データベース開発者ガイド

Amazon Redshift

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Redshift: データベース開発者ガイド

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスと関連付けてはならず、また、お客様に混乱を招くような形や Amazon の信用を傷つけたり失わせたりする形で使用することはできません。Amazon が所有しない商標はすべてそれぞれの所有者に所属します。所有者は必ずしも Amazon と提携していたり、関連しているわけではありません。また、Amazon 後援を受けているとはかぎりません。

Table of Contents

Amazon Redshift の概要

Amazon Redshift データベースデベロッパーガイドにようこそ。このガイドでは、Amazon Redshift を使用してデータウェアハウスを作成および管理する方法を理解することに重点を置いています。このガイドは、データベースに対して設計者、ソフトウェア開発者、または管理者として作業する方を対象として、データウェアハウスの設計、構築、クエリ、および保守に必要となる情報を示しています。

Amazon Redshift は、 クラウド内でのフルマネージド型、ペタバイト規模のデータウェアハウスサービスです。Amazon Redshift Serverless を使用すると、プロビジョニングされたデータウェアハウスの通常の設定がなくても、データにアクセスして分析することができます。リソースは自動的にプロビジョニングされて、データウェアハウス容量はインテリジェントにスケーリングされ、要求が厳しく、予測不可能なワークロードであっても高速なパフォーマンスを実現します。データウェアハウスがアイドル状態のときには課金されず、使用した分のみ支払います。データセットのサイズに関係なく、Amazon Redshift クエリエディタ v2 またはお好みのビジネスインテリジェンス (BI) ツールで、データをロードしてクエリを直ちに開始することができます。使いやすい管理不要の環境で、最高のコストパフォーマンスと使い慣れた SQL 機能をお楽しみください。

トピック

Amazon Redshift を使用するための前提条件

このトピックでは、Amazon Redshift を使用するために必要な前提条件について説明します。

このガイドを使用する前に、以下のタスクを完了する方法について説明している「Redshift Serverless データウェアハウスの使用を開始」をお読みください。

- Amazon Redshift Serverless でデータウェアハウスを作成します。
- Amazon Redshift クエリエディタ v2 でサンプルデータをロードする
- Amazon S3 からデータをロードする。

また、その SQL クライアントの使い方を知っていることと、SQL 言語の基本を理解していることも必要です。

Amazon Redshift アーキテクチャ

このトピックは、Amazon Redshift を構成するコンポーネントを理解するのに役立ちます。

Amazon Redshift データウェアハウスは、エンタープライズクラスのリレーショナルデータベースのクエリおよび管理を行うためのシステムです。

Amazon Redshift では、ビジネスインテリジェンス (BI) や、レポート作成、データ処理、分析用のツールなど、多種類のアプリケーションとのクライアント接続をサポートしています。

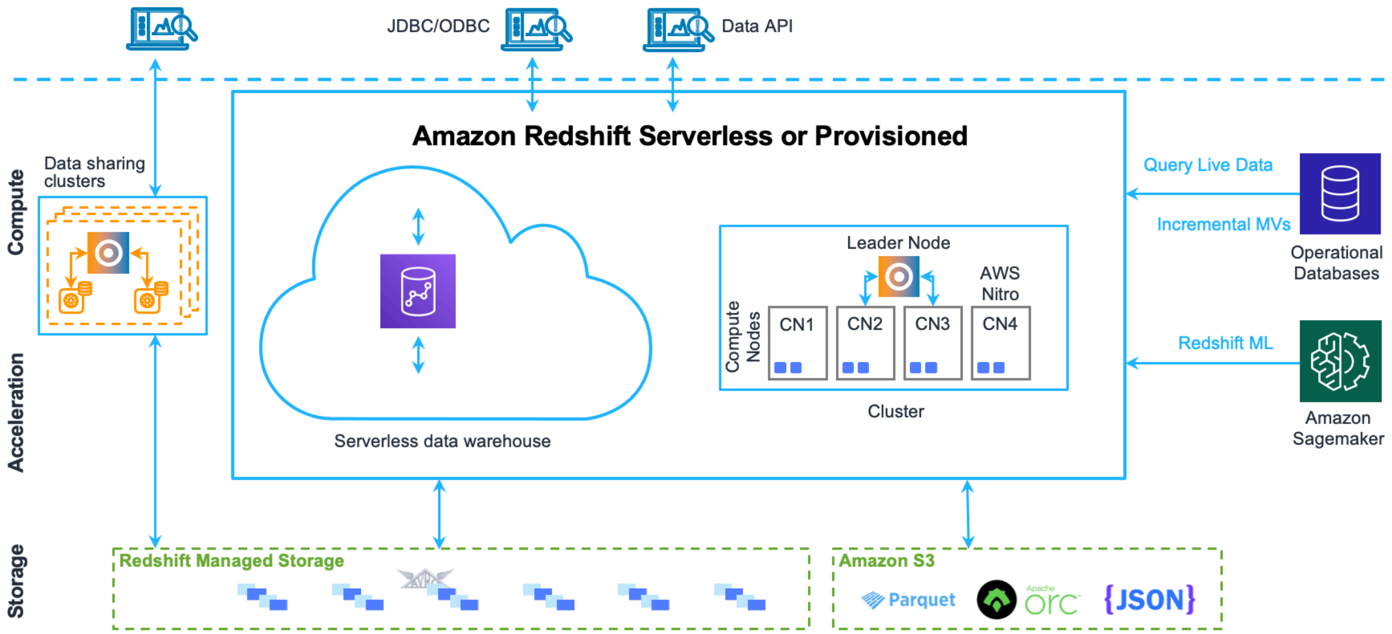
分析クエリを実行するときは、大量のデータを複数のステージからなる操作で取得、比較、および評価して、最終的な結果を生成します。

Amazon Redshift は、超並列処理、列指向データストレージ、および非常に効率的で対象を限定したデータ圧縮とエンコードのスキームの組み合わせによって、効率的なストレージと最善のクエリパフォーマンスを実現します。このセクションでは、Amazon Redshift でのシステムアーキテクチャの概要を説明します。

トピック

データウェアハウスシステムのアーキテクチャ

このセクションでは、次の図で示すような、Amazon Redshift データウェアハウスアーキテクチャを構成するコンポーネントについて説明します。



クライアントアプリケーション

Amazon Redshift は、さまざまなデータロードおよび ETL (抽出、変換、およびロード) ツールや、ビジネスインテリジェンス (BI) レポート、データマイニング、そして分析ツールなどと統合されています。Amazon Redshift はオープンスタンダードの PostgreSQL をベースにしているため、ほとんどの既存 SQL クライアントアプリケーションを、最小限の変更のみで動作させられます。Amazon Redshift SQL と PostgreSQL の間での重要な相違点については、「[Amazon Redshift および PostgreSQL]」を参照してください。

クラスター

Amazon Redshift データウェアハウスの中核となるインフラストラクチャコンポーネントは、クラスターです。

クラスターは、1 つまたは複数のコンピューティングノードで構成されます。クラスターが 2 つ以上のコンピューティングノードでプロビジョニングされている場合、追加のリーダーノードがコンピューティングノードを調整して、外部の通信を処理します。クライアントアプリケーションはリーダーノードとのみ直接通信します。コンピューティングノードは外部アプリケーションに対して透過的です。

リーダーノード

リーダーノードは、クライアントプログラムおよびコンピューティングノードとのすべての通信を管理します。リーダーノードは、データベース操作を遂行するための実行計画、特に複雑なクエリの

結果を取得するために必要な一連の手順を解析および作成します。リーダーノードは、実行計画に基づいて、コードをコンパイルし、コンパイル済みのコードをコンピューティングノードに配布してから、データの一部分を各コンピューティングノードに割り当てます。

リーダーノードは、コンピューティングノードに格納されているテーブルがクエリで参照されている場合にのみ、コンピューティングノードに SQL ステートメントを配布します。他のすべてのクエリは、リーダーノードで排他的に実行されます。Amazon Redshift は、特定の SQL 機能をリーダーノードのみに実装するように設計されています。これらの機能を使用するクエリが、コンピューティングノードに存在するテーブルを参照する場合、エラーが返されます。詳細については、「リーダーノードでサポートされる SQL 関数」を参照してください。

コンピューティングノード

リーダーノードは、実行計画の個別要素のコードをコンパイルし、コードを個々のコンピューティングノードに割り当てます。コンピューティングノードは、コンパイル済みのコードを実行し、最終的な集計のために中間結果をリーダーノードに返送します。

各コンピューティングノードの専用 CPU およびメモリは、ノードのタイプによって異なります。ワークロードが増えるに従って、ノードの数を増やすかノードの種類をアップグレードして、または両方を行って、クラスターのコンピューティング能力を拡張できます。

Amazon Redshift には、コンピューティングのニーズに合わせて複数のノードタイプが用意されています。各ノードタイプの詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift クラスター」を参照してください。

Redshift マネージドストレージ

データウェアハウスのデータは、Redshift マネージドストレージ (RMS) に保存されます。RMS では、Amazon S3 ストレージを使用してストレージをペタバイト規模にスケーリングできます。RMS では、コンピューティングとストレージを個別にスケールして料金を支払うことができるため、コンピューティングのニーズのみに基づいてクラスターのサイズを設定できます。高性能 SSD ベースのローカルストレージを Tier 1 キャッシュとして自動的に使用します。また、データブロックの温度、データブロックの有効期間、ワークロードパターンなどの最適化を利用し、操作を行わなくても必要に応じて自動的にストレージを Amazon S3 にスケーリングすることができます。

ノードスライス

コンピューティングノードはスライスに分割されています。各スライスは、ノードのメモリとディスク容量の一部を割り当てられ、ノードに割り当てられたワークロードの一部分を処理します。リーダーノードは、スライスへのデータの分散を管理し、クエリまたは他のデータベース操作のワークロードをスライスに分配します。スライスは、並列処理を行って操作を完了します。

ノードあたりのスライスの数は、クラスターのノードサイズによって決まります。各ノードサイズに含まれるスライス数の詳細については、「Amazon Redshift 管理ガイド」の「クラスターとノードについて」を参照してください。

テーブルを作成するときは、必要に応じて 1 つの列を分散キーとして指定できます。テーブルにデータがロードされるときは、テーブルに対して定義されている分散キーに従って行がノードスライスに分配されます。適切な分散キーを選択することにより、Amazon Redshift は並列処理を使用してデータのロードやクエリを効率的に実行できようになります。分散キーの選択については、「最適な分散スタイルの選択」を参照してください。

内部ネットワーク

Amazon Redshift は、高帯域幅接続、近接性、およびカスタム通信プロトコルを活用することで、リーダーノードとコンピューティングノードの間に、プライベートかつ非常に高速なネットワーク通信を実現します。コンピューティングノードは、クライアントアプリケーションが直接アクセスすることのない別の独立したネットワークで動作します。

データベース

クラスターは、1 つ以上のデータベースで構成されます。ユーザーデータはコンピューティングノードに格納されます。SQL クライアントはリーダーノードと通信し、リーダーノードはクエリの実行をコンピューティングノードと調整します。

Amazon Redshift はリレーショナルデータベース管理システム (RDBMS) なので、他の RDBMS アプリケーションとも互換性があります。Amazon Redshift は、標準的な RDBMS と同じ機能 (データの挿入や削除といったオンライントランザクション処理 (OLTP) 機能など) を提供しますが、この機能は、非常に大きいデータセットに関する高パフォーマンスの分析やレポート向けに最適化されています。

Amazon Redshift は PostgreSQL に基づいています。Amazon Redshift と PostgreSQL の間には非常に重要な相違点がいくつかあり、データウェアハウスアプリケーションの開発と設計の際には、それらを考慮に入れる必要があります。Amazon Redshift SQL と PostgreSQL の違いについては、「Amazon Redshift および PostgreSQL」 を参照してください。

Amazon Redshift のパフォーマンス

このトピックでは、パフォーマンスを向上させる Amazon Redshift のコンポーネントについて説明します。これらのコンポーネントを理解することで、Amazon Redshift のパフォーマンスを調整し、パフォーマンスの低下をトラブルシューティングするのに役立ちます。

Amazon Redshift では、以下のパフォーマンス機能を採用することによって、極めて高速のクエリ実行を実現しています。

トピック

- 超並列処理
- 列指向データストレージ
- データ圧縮
- クエリオプティマイザ
- 結果のキャッシュ
- コンパイル済みコード

超並列処理

超並列処理 (MPP) により、大量のデータに対して非常に複雑なクエリを高速で実行できます。複数のコンピューティングノードがすべてのクエリ処理を行って、最終的に結果が集計されます。各ノードの各コアは、同じコンパイル済みクエリセグメントをデータ全体の一部分に対して実行します。

Amazon Redshift では、テーブルの行がコンピューティングノードに分散され、これによりデータの並列処理が可能となっています。各テーブルに対して適切な分散キーを選択することにより、データの分配を最適化して、ワークロードを分散し、ノード間のデータの移動を低減できます。詳細については、「最適な分散スタイルの選択」を参照してください。

フラットファイルからのデータのロードでは、複数のノードにワークロードを分散させることで並列処理を活用しながら、複数のファイルから同時に読み取ります。テーブルへのデータのロード方法の詳細については、「データをロードするための Amazon Redshift のベストプラクティス」を参照してください。

列指向データストレージ

データベーステーブルの列指向ストレージはディスク全体の必要な I/O を大幅に減らすため、分析クエリのパフォーマンスの最適化の重要な要因となっています。詳細については、「列指向ストレージ」を参照してください。

列が適切にソートされていると、クエリプロセッサは大きいデータブロックのサブセットをすばやく除外できます。詳細については、「最良のソートキーの選択」を参照してください。

データ圧縮

データ圧縮により必要なストレージが減るので、ディスク I/O が減少し、クエリのパフォーマンスが向上します。クエリを実行すると、圧縮されたデータがメモリに読み込まれ、クエリの実行時に圧縮解除されます。メモリにロードするデータを少なくできるので、Amazon Redshift は、より多くのメモリをデータ分析のために割り当てられるようになります。列指向ストレージでシーケンシャルに格納される類似のデータに対して、Amazon Redshift では、列指向のデータ型と明示的に結び付けられた適応型圧縮エンコーディングを利用しています。テーブルの列でデータの圧縮を有効にする最良の方法は、テーブルにデータをロードする際に、Amazon Redshift で最適な圧縮エンコーディングが適用されるようにすることです。自動データ圧縮の詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

クエリオプティマイザ

Amazon Redshift のクエリ実行エンジンには、MPP 対応で、列指向データストレージも利用するクエリオプティマイザが組み込まれています。Amazon Redshift のクエリオプティマイザでは、複数テーブルの結合、サブクエリ、および集計処理が含まれることが多い、複雑な分析クエリを処理するための、多くの強化機能と拡張機能が実装されています。クエリの最適化の詳細については、「クエリパフォーマンスのチューニング」を参照してください。

結果のキャッシュ

クエリの実行時間を短縮し、システムパフォーマンスを向上させるために、Amazon Redshift は特定の種類のクエリの結果をリーダーノード上のメモリにキャッシュします。ユーザーがクエリを送信すると、Amazon Redshift は結果キャッシュ内で、有効性がありキャッシュされているクエリ結果のコピーをチェックします。結果のキャッシュに一致するものが見つかった場合、Amazon Redshift はキャッシュ済みの結果を使用して、クエリは実行しません。結果のキャッシュはユーザーに対して透過的です。

結果のキャッシュはデフォルトでオンになっています。現在のセッションで結果のキャッシュをオフにするには、enable_result_cache_for_session パラメータを `off` に設定します。

Amazon Redshift は、次の条件がすべて満たされた場合に、新しいクエリに対してキャッシュ内の結果を適用します。

- クエリを発行したユーザーは、クエリで使用されるオブジェクトへのアクセス許可を持っています。
- クエリ内のテーブルまたはビューは変更されていません。

- クエリでは、GETDATE のような、実行するたびに評価する必要がある関数は使用されません。
- クエリは Amazon Redshift Spectrum の外部テーブルを参照しません。
- クエリの結果に影響を与える構成パラメータは変更されません。
- クエリは、キャッシュされたクエリと構文的に一致します。

キャッシュの有効性を最大化し、リソースを最も効率的に使用をするため、Amazon Redshift では、サイズの大きなクエリ結果セットの一部をキャッシュしていません。Amazon Redshift は、クエリ結果をキャッシュするかどうかを、さまざまな要因に基づいて決定します。それらの要因とは、キャッシュ内のエントリ数や、Amazon Redshift クラスターのインスタンスタイプなどです。

クエリで結果のキャッシュを使用するかどうかを決定するには、SVL_QLOG システムビューをクエリします。クエリで結果キャッシュが使用された場合、source_query 列はソースクエリのクエリ ID を戻します。結果のキャッシュが使用されない場合、source_query 列の値は Null になります。

次の例は、ユーザー ID 104 およびユーザー ID 102 によって送信されたクエリが、ユーザー ID 100 によって実行されるクエリからの結果キャッシュを使用することを示しています。

```
select userid, query, elapsed, source_query from svl_qlog
where userid > 1
order by query desc;

userid | query  | elapsed  | source_query
-------+--------+----------+-------------
   104 | 629035 |       27 |       628919
   104 | 629034 |       60 |       628900
   104 | 629033 |       23 |       628891
   102 | 629017 |  1229393 |
   102 | 628942 |       28 |       628919
   102 | 628941 |       57 |       628900
   102 | 628940 |       26 |       628891
   100 | 628919 | 84295686 |
   100 | 628900 | 87015637 |
   100 | 628891 | 58808694 |
```

コンパイル済みコード

リーダーノードは、完全に最適化されたコンパイル済みコードをクラスターのすべてのノードに分配します。クエリをコンパイルすると、インタープリタに伴うオーバーヘッドが削減されるので、実行速度 (特に複雑なクエリの場合) が向上します。コンパイル済みのコードはキャッシュされ、同じク

ラスターのセッション間で共有されます。その結果、同じクエリを将来的に実行した場合、パラメータを変えても高速化されることが多くなります。

クエリ実行エンジンは、JDBC と ODBC 接続プロトコルでは異なるコードをコンパイルするので、異なるプロトコルを使用する 2 つのクライアントで、コードの初回コンパイル時のコストが発生します。ただし、同じプロトコルを使用するクライアントには、キャッシュされたコードの共有によるメリットがあります。

列指向ストレージ

このセクションでは、Amazon Redshift が表形式のデータを効率的に保存するために使用する方法である列指向ストレージについて説明します。

データベーステーブルの列指向ストレージは、ディスク全体の必要な I/O を大幅に減らすため、分析クエリのパフォーマンスの最適化の重要な要因となっています。これにより、ディスクからロードする必要のあるデータの量が減ります。

以下の一連の図では、列指向データストレージにおいて効率性がどのように実装され、データがメモリに読み込まれるときの効率性にどのように変換されるのかについて説明します。

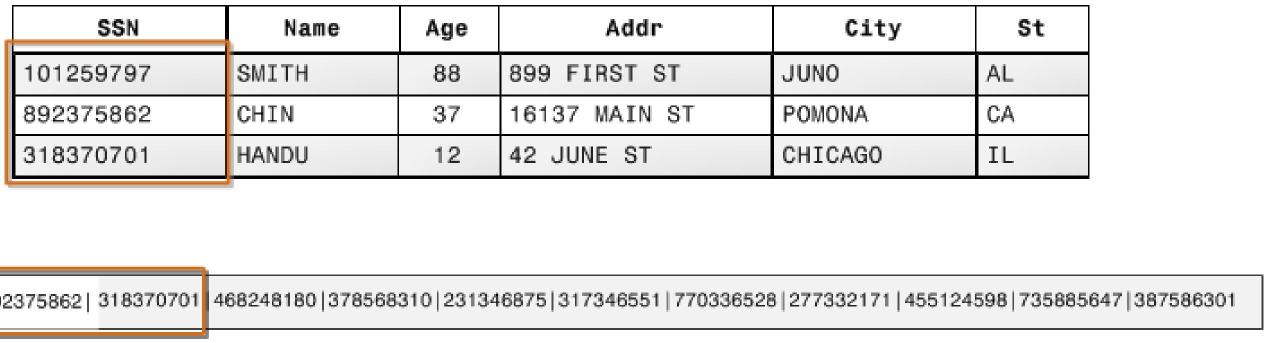
この最初の図では、データベーステーブルのレコードが行ごとにディスクブロックに格納される一般的な方法を示します。

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

標準的なリレーショナルデータベーステーブルでは、各行には 1 つのレコードのフィールド値が含まれます。行対応のデータベースストレージでは、データブロックには、行全体を構成する連続した各列の値がシーケンシャルに格納されます。ブロックサイズがレコードのサイズより小さい場合、1 レコード全体のストレージに複数のブロックが使用される場合があります。ブロックサイズがレコードのサイズより大きい場合、1 レコード全体のストレージが 1 ブロックより小さくなり、ディスク容量の使用効率が悪くなる場合があります。オンライントランザクション処理 (OLTP) アプリケーションでは、ほとんどのトランザクションに、通常は一度に 1 レコードまたは少数のレコードの、

レコード全体の値の頻繁な読み取りおよび書き込みが含まれます。そのため、行対応のストレージは OLTP データベースに最適です。

次の図は、列指向ストレージにおいて、各列の値がディスクブロックにシーケンシャルに格納されることを示しています。



Block 1

列指向ストレージを使用すると、各データブロックには複数の行の 1 つの列の値が格納されます。レコードがシステムに入ると、Amazon Redshift は各列のデータを、列指向ストレージ向けに透過的に変換します。

この単純化した例では、列指向ストレージを使用することで、行ベースのストレージと比較して 3 倍の数のレコードの列フィールド値を各データブロックに保持しています。つまり、同じ数のレコードの同じ数の列フィールド値を読み取るのに必要な I/O 操作は、行対応ストレージの 3 分の 1 です。実際には、列数と行数が非常に多いテーブルでは、ストレージ効率がいっそう向上します。

利点としては、各ブロックが同じ型のデータを保持するので、ブロックデータは列のデータ型に対して最適な圧縮方式を使用でき、ディスク容量と I/O をさらに削減できることが挙げられます。データ型に基づく圧縮エンコーディングの詳細については、「圧縮エンコード」を参照してください。

ディスクでのデータ保存容量の減少は、データの取得とメモリへのデータの格納にも引き継がれます。多くのデータベース操作では一度に 1 つまたは少数の列にのみアクセスして処理する必要があるだけなので、クエリに実際に必要な列のブロックだけを取得することでメモリ容量を節約できます。OLTP トランザクションでは、通常、少数のレコードの行の大部分または全部の列が関係するのに対し、データウェアハウスクエリでは、一般に、大量の行のほんの数列だけが読み取られます。つまり、同じ数の行の同じ数の列フィールド値を読み取るのに必要な I/O 操作は、ほんのわずかです。行単位のブロックの処理に必要なメモリのほんの一部しか使用しません。実際には、列数と行数が非常に多いテーブルでは、効率が比例的に向上します。例えば、100 列のテーブルがあるとします。5 列を使用するクエリは、テーブルに含まれるデータの約 5 パーセントだけを読み取る必要があります。大きいデータベースの場合、この節約は数十億または数兆のレコードに対して行われます。一方、行対応のデータベースでは、必要のない 95 列も含まれるブロックを読み取ります。

データベースでの一般的なブロックサイズは、2 KB ~ 32 KB の範囲です。Amazon Redshift では、より効率的な 1 MB のブロックサイズを使用しているため、クエリの実行に伴うデータベースの負荷や、他の操作の実行に必要な I/O 要求の数はさらに減少しています。

ワークロード管理

このセクションでは、Amazon Redshift がクエリを準備して実行する方法を理解するのに役立つワークロード管理 (WLM) について説明します。

Amazon Redshift のワークロード管理 (WLM) により、ユーザーはワークロード内の柔軟な優先順位が可能になります。これにより、実行速度が高く処理時間の短いクエリが、処理時間の長いクエリの後に滞らないようにできます。Amazon Redshift は、サービスクラスに従ってランタイム時にクエリキューを作成します。サービスクラスでは、内部システムキューやユーザーからアクセスが可能なキューなどの、さまざまな種類のキューに対する設定パラメータが定義されています。ユーザーから見た場合、ユーザーアクセス可能サービスクラスとキューは機能的に同じものです。一貫性を保つため、このドキュメントでは、ユーザーアクセス可能サービスクラスとランタイムキューは、キューという用語を使用して表します。

Redshift は、自動 WLM と呼ばれる自動ワークロード管理を提供します。これは、さまざまなワークロードを処理するように調整され、推奨されるデフォルトです。自動 WLM を使用すると、Redshift はクエリが到着したときにリソース使用率を判断し、それらをメインクラスターで実行するか、通貨スケーリングクラスターで実行するか、それぞれをキューに送信するかを動的に決定します。(クエリがキューに入れられると、自動 WLM はより期間の短いクエリを優先します)。自動 WLM は合計スループットを最大化し、効率的なデータウェアハウスリソースを維持できます。ワークロードは、サイズやスケジュールを心配することなく実行できます。自動 WLM は、プロビジョニングされたクラスターのデフォルトです。詳細については、「自動 WLM の実装」を参照してください。

> **Note**
>
> Amazon Redshift Serverless ワークグループは常に自動 WLM を使用します。

大量のクエリやリソースを大量に消費するクエリが実行されている場合、ワークロード管理は、ワークロードがローカルリソースでキューに入っているときに、追加のコンピューティングリソースにスケールできます。同時実行スケーリングと自動 WLM を一緒に使用すると、事実上無制限の同時ユーザーと同時クエリに対して一貫したパフォーマンスでサポートできます。

Redshift プロビジョニングクラスターは、きめ細かな手動での最適化が必要な場合に手動 WLM を提供します。ここでは、カスタマーはリソースの割り当て、クエリの同時実行、キューイングを管理し

ます。クエリを実行すると、WLM は、ユーザーのユーザーグループに従ってクエリをキューに割り当てるか、キューの設定でリストされているキューグループを照合することによってクエリをキューに割り当てます。これは、ユーザーが設定したクエリグループラベルで設定されます。詳細については、「手動 WLM を実装する」を参照してください。

手動 WLM はワークロードパターンに合わせて時間の経過とともに微調整できますが、ほとんどの場合、その静的な性質のため、1 日または長期間にわたって変化するワークロードに適応することが困難になる可能性があるため、使用はお勧めしません。より多くのモニタリングと継続的な調整が必要となります。また、多くの場合、手動 WLM は自動 WLM ほど効率的にコンピューティングリソースを使用しません。例えば、キューが手動で設定され、割り当てられたメモリが制限される場合などです。

ワークロード管理設定の成功を測定するための重要なメトリクスは、システムスループットです。つまり、これは正常に完了したクエリの数のことです。システムスループットは 1 秒あたりのクエリ数で測定されます。詳細については、「Amazon Redshift クラスターパフォーマンスのモニタリング」を参照してください。

WLM の設定を管理するには、Amazon Redshift マネジメントコンソールを使用することが最も簡単です。さらに、Amazon Redshift コマンドラインインターフェイス (CLI) または Amazon Redshift API を使用することもできます。ワークロード管理の実装と使用の詳細については、「ワークロード管理の実装」を参照してください。

他のサービスと Amazon Redshift を併用する

このセクションでは、Amazon Redshift のデータソースおよびデータ送信先として他のサービスを使用する方法について説明します。

Amazon Redshift を他の AWS サービスと統合し、データセキュリティ機能を使用することにより、ユーザーは確実かつ安全に、すばやくデータを移動、変換、ロードできるようになります。

S3

Amazon Simple Storage Service (Amazon S3) は、クラウドにデータを保存するためのウェブサービスです。Amazon Redshift は、Amazon S3 バケットに保存されている複数のデータファイルからデータの読み取りとロードを行うために、並列処理を活用します。詳細については、「Amazon S3 からデータをロードする」を参照してください。

また、Amazon Redshift データウェアハウスから Amazon S3 上の複数のデータファイルに対するデータのエクスポートに、並列処理を使用することもできます。詳細については、「Amazon Redshift でのデータのアンロード」を参照してください。

DynamoDB

Amazon DynamoDB は、フルマネージド型の NoSQL データベースサービスです。1 つの Amazon DynamoDB テーブルからのデータを含む Amazon Redshift テーブルを、COPY コマンドによりロードすることができます。詳細については、「Amazon DynamoDB テーブルからのデータのロード」を参照してください。

SSH

Amazon Redshift の COPY コマンドは、Amazon EMR クラスター、Amazon EC2 インスタンス、あるいは他のコンピュータなど、1 つ以上のリモートホストからデータをロードするために使用できます。COPY では SSH を使用してリモートホストに接続し、リモートホストでコマンドを実行してデータを生成します。Amazon Redshift は複数の同時接続をサポートしています。COPY コマンドでは、複数のホストのソースからの出力を並列で読み取ってロードします。詳細については、「リモートホストからデータをロードする」を参照してください。

AWS Data Pipeline

AWS Data Pipeline を使用して、Amazon Redshift との間のデータの移動や変換を自動化できます。AWS Data Pipeline に組み込まれたスケジューリング機能を使用することで、データ転送や変換のための複雑なロジックを記述することなく、繰り返し行うジョブをスケジュールして実行することができます。例えば、繰り返しのジョブを設定して、データを Amazon DynamoDB から Amazon Redshift へ自動的にコピーすることができます。Amazon S3 から Amazon Redshift に対し、データを定期的に移動するパイプラインを作成するプロセスに関するチュートリアルは、AWS Data Pipeline デベロッパーガイド の「AWS Data Pipelineを使用して Amazon Redshift にデータをコピーする」を参照してください。

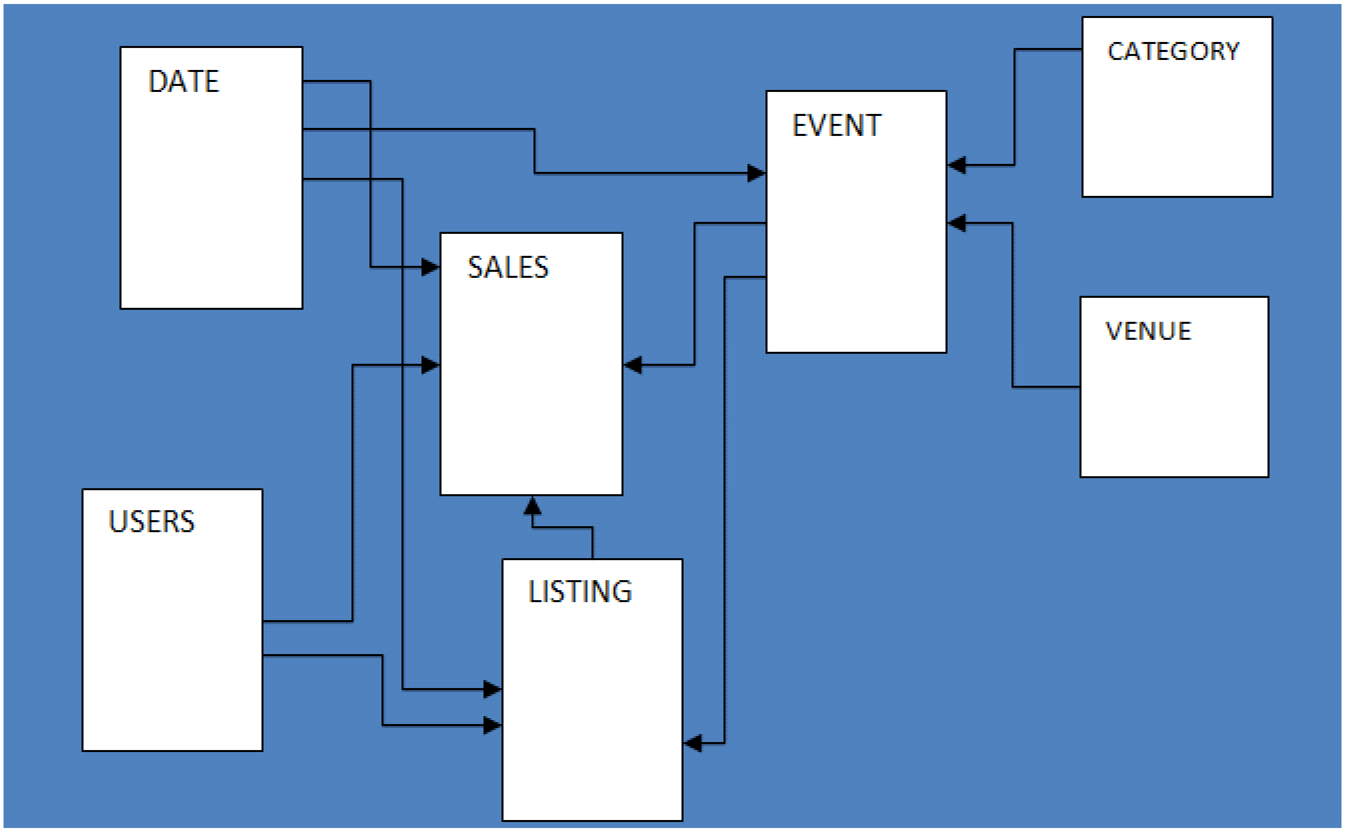
AWS DMS

Amazon Redshift へのデータの移行に AWS Database Migration Service を使用できます。AWS DMS は、Oracle、PostgreSQL、Microsoft SQL Server、Amazon Redshift、Aurora DB クラスター、DynamoDB、Amazon S3、MariaDB、MySQL など、最も広く使用されている商用およびオープンソースのデータベースとの間で相互にデータを移行できます。詳細については、「AWS Database Migration Service のターゲットとしての Amazon Redshift データベースの使用」を参照してください。

サンプルデータベース

このセクションでは、Amazon Redshift ドキュメントの例で使用されている、TICKIT というサンプルデータベースについて説明します。

この小規模のデータベースは、7 個のテーブルで構成されています。そのうち 2 個はファクトテーブル、5 個はディメンションです。「Amazon Redshift 入門ガイド」の「ステップ 4: Amazon S3 から Amazon Redshift にデータをロードする」の手順に従って、TICKIT データセットをロードできます。



このサンプルデータベースアプリケーションでは、アナリストが、架空の TICKIT ウェブサイトの販売アクティビティを追跡できます。ユーザーはこのサイトで、スポーツイベント、ショー、およびコンサートのチケットをオンラインで購入および販売します。アナリストは特に、長期間にわたってチケットの動向を確認でき、販売者の成功率に加えて、人気の高いイベント、施設、シーズンを確認できます。アナリストは、この情報を使用して、頻繁にサイトにアクセスする購入者および販売者にインセンティブを提供したり、新規ユーザーを誘致したり、広告やプロモーションを推進したりすることができます。

例えば、次のクエリは、2008 年に販売されたチケット数に基づき、サンディエゴで最も販売数の多かった販売者 5 名を検出します。

```
select sellerid, username, (firstname ||' '|| lastname) as name,
city, sum(qtysold)
from sales, date, users
where sales.sellerid = users.userid
and sales.dateid = date.dateid
```

```
and year = 2008
and city = 'San Diego'
group by sellerid, username, name, city
order by 5 desc
limit 5;

sellerid | username |       name        |   city    | sum
----------+----------+-------------------+-----------+-----
49977 | JJK84WTE | Julie Hanson      | San Diego |  22
19750 | AAS23BDR | Charity Zimmerman | San Diego |  21
29069 | SVL81MEQ | Axel Grant        | San Diego |  17
43632 | VAG08HKW | Griffin Dodson    | San Diego |  16
36712 | RXT40MKU | Hiram Turner      | San Diego |  14
(5 rows)
```

このガイドで例示しているデータベースには、小規模のデータセットが含まれています。このデータセットでは、2 つのファクトテーブルそれぞれに含まれる行が 200,000 行未満で、ディメンションの範囲は CATEGORY テーブルの 11 行から USERS テーブルの 50,000 行までです。

特に、このガイドに記載するデータベースの例では、Amazon Redshift テーブル設計における以下の主要機能をデモンストレーションしています。

- データのディストリビューション
- データのソート
- 列指向の圧縮

TICKIT データベース内のテーブルのスキーマについては、以下のタブを選択してください。

CATEGORY table

列名	データ型	説明
CATID	SMALLINT	プライマリキー。各行の一意の ID 値。各行は、チケットが購入および販売される特定のタイプのイベントを表します。
CATGROUP	VARCHAR(10)	イベントのグループの記述名 (例: **Shows** および **Sports**)。

列名	データ型	説明
CATNAME	VARCHAR(10)	グループ内のイベントのタイプの短い記述名 (例: **Opera** および **Musicals**)。
CATDESC	VARCHAR(50)	イベントのタイプの長い記述名 (例: **Musical theatre**)。

DATE table

列名	データ型	説明
DATEID	SMALLINT	プライマリキー。各行の一意の ID 値。各行は、1 暦年の中の各日を表します。
CALDATE	DATE	カレンダー日付 (例: **2008-06-24**)。
DAY	CHAR(3)	曜日 (短縮形) (例: **SA**)。
WEEK	SMALLINT	週番号 (例: **26**)。
MONTH	CHAR(5)	月名 (短縮形) (例: **JUN**)。
QTR	CHAR(5)	四半期番号 (**1**～**4**)。
YEAR	SMALLINT	4 桁の年 (**2008**)。
HOLIDAY	BOOLEAN	その日が祝日 (米国) であるかどうかを示すフラグ。

EVENT table

列名	データ型	説明
EVENTID	INTEGER	プライマリキー。各行の一意の ID 値。各行は、特定の時刻に特定の施設で開催された各イベントを表します。
VENUEID	SMALLINT	VENUE テーブルの外部キー参照。

列名	データ型	説明
CATID	SMALLINT	CATEGORY テーブルの外部キー参照。
DATEID	SMALLINT	DATE テーブルの外部キー参照。
EVENTNAME	VARCHAR(200)	イベントの名前 (例: **Hamlet** または **La Traviata**)。
STARTTIME	TIMESTAMP	イベントの日付と開始時刻 (例: **2008-10-10 19:30:00**)。

VENUE table

列名	データ型	説明
VENUEID	SMALLINT	プライマリキー。各行の一意の ID 値。各行は、イベントが開催された特定の施設を表します。
VENUENAME	VARCHAR(100)	施設の正式名称 (例: **Cleveland Browns Stadium**)。
VENUECITY	VARCHAR(30)	市町村名 (例: **Cleveland**)。
VENUESTATE	CHAR(2)	州または地域の 2 文字の略称 (米国およびカナダ) (例: **OH**)。
VENUESEATS	INTEGER	施設で利用できる座席の最大数 (ただし、確認済みの場合) (例: **73200**)。デモンストレーションの目的から、この列には null 値と 0 がいくつか含まれています。

USERS table

列名	データ型	説明
USERID	INTEGER	プライマリキー。各行の一意の ID 値。各行は、1 つ以上のイベントでチケットをリストまたは購入した登録済みのユーザー (購入者、販売者、または両方) を表します。
USERNAME	CHAR(8)	英数字 8 文字のユーザー名 (例: **PGL08LJI**)。
FIRSTNAME	VARCHAR(30)	ユーザーの名 (例: **Victor**)。
LASTNAME	VARCHAR(30)	ユーザーの姓 (例: **Hernandez**)。
CITY	VARCHAR(30)	ユーザーの自宅住所の市町村 (例: **Naperville**)。
STATE	CHAR(2)	ユーザーの自宅住所の州 (例: **GA**)。
EMAIL	VARCHAR(100)	ユーザーの E メールアドレス (この列にはランダムなラテン文字値が格納される) (例: **turpis@accumsanlaoreet.org**)。
PHONE	CHAR(14)	ユーザーの 14 文字の電話番号 (例: **(818) 765-4255**)。
LIKESPORTS, ..。	BOOLEAN	ユーザーの好き嫌いを **true** 値および **false** 値で表す一連の 10 列。

LISTING table

列名	データ型	説明
LISTID	INTEGER	プライマリキー。各行の一意の ID 値。各行は、特定のイベントの一連のチケットをリストで表します。
SELLERID	INTEGER	USERS テーブルの外部キー参照 (チケットを販売するユーザーを表す)。

列名	データ型	説明
EVENTID	INTEGER	EVENT テーブルの外部キー参照。
DATEID	SMALLINT	DATE テーブルの外部キー参照。
NUMTICKETS	SMALLINT	販売可能なチケット数 (例: `2` または `20`)。
PRICEPERTICKET	DECIMAL(8,2)	各チケットの定価 (例: `27.00` または `206.00`)。
TOTALPRICE	DECIMAL(8,2)	このリストの定価総額 (NUMTICKETS*PRICEPERTICKET)。
LISTTIME	TIMESTAMP	リストが投稿された日時 (例: `2008-03-18 07:19:35`)。

SALES table

列名	データ型	説明
SALESID	INTEGER	プライマリキー。各行の一意の ID 値。各行は、特定のイベントの 1 枚以上のチケットの各販売を表します (特定のリストで提供)。
LISTID	INTEGER	LISTING テーブルの外部キー参照。
SELLERID	INTEGER	USERS テーブルの外部キー参照 (チケットを販売したユーザー)。
BUYERID	INTEGER	USERS テーブルの外部キー参照 (チケットを購入したユーザー)。
EVENTID	INTEGER	EVENT テーブルの外部キー参照。
DATEID	SMALLINT	DATE テーブルの外部キー参照。
QTYSOLD	SMALLINT	販売されたチケット数 (`1`～`8`)。(1 回の取引で最大 8 枚のチケットを販売できます。)

列名	データ型	説明
PRICEPAID	DECIMAL(8,2)	チケットの合計支払額 (例: **75.00** または **488.00**)。チケットの個別の価格は PRICEPAID/QTYSOLD です。
COMMISSION	DECIMAL(8,2)	そのビジネスで販売価格から徴収される 15% のコミッション (例: **11.25** または **73.20**)。販売者は PRICEPAID 値の 85% を受け取ります。
SALETIME	TIMESTAMP	販売が完了した日時 (例: **2008-05-24 06:21:47**)。

Amazon Redshift のベストプラクティス

以下では、概念実証の計画、テーブルの設計、テーブルへのデータのロード、Amazon Redshif のクエリの書き込みのベストプラクティスと、Amazon Redshift Advisor の操作の説明を示します。

Amazon Redshift は、他の SQL データベースシステムとは異なります。Amazon Redshift アーキテクチャを最大限に活用するには、超並列処理、列指向データストレージ、および列指向データ圧縮を使用するようにテーブルを設計し、構築して、ロードする必要があります。データロード時間とクエリ実行時間が予想以上または必要以上に長い場合は、重要な情報を見落としている可能性があります。

経験豊富な SQL データベースデベロッパーには、Amazon Redshift データウェアハウスの開発を開始する前に、このトピックを読むことを強くお勧めします。

SQL データベース開発の初心者には、このトピックから開始することをお勧めしません。まず、「Amazon Redshift 入門ガイド」の「コマンドを実行して、データウェアハウス内のデータベースを定義して使用する」を読み、いくつかの例を実際に試してください。

このトピックでは、最も重要な開発の原則の概要と、これらの原則を実装するために役立つ具体的なヒント、例、およびベストプラクティスを示します。1 つのベストプラクティスをあらゆる状況に適用できるわけではありません。データベース設計を確定する前に、あらゆるオプションを評価してください。詳細については、「自動テーブル最適化」、「Amazon Redshift でのデータのロード」、「クエリパフォーマンスのチューニング」、および参照章を参照してください。

トピック

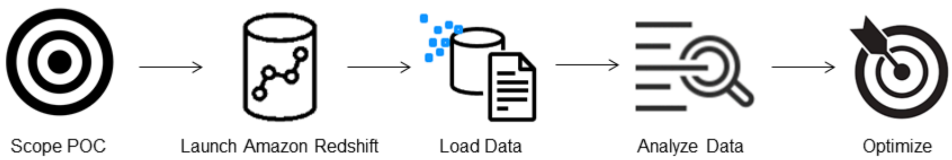
Amazon Redshift の概念実証 (POC) を実施する

Amazon Redshift は、広く利用されているクラウドデータウェアハウスであり、組織の Amazon Simple Storage Service データレイク、リアルタイムストリーム、機械学習 (ML) ワークフロー、トランザクションワークフローなどと連携するクラウドベースのフルマネージドサービスを提供しま

す。以降のセクションでは、Amazon Redshift で概念実証 (POC) を実施するプロセスについて説明します。ここで紹介する情報は、POC の目標を設定する際に役立ちます。ここでは、POC のサービスのプロビジョンと設定を自動化できるツールを活用します。

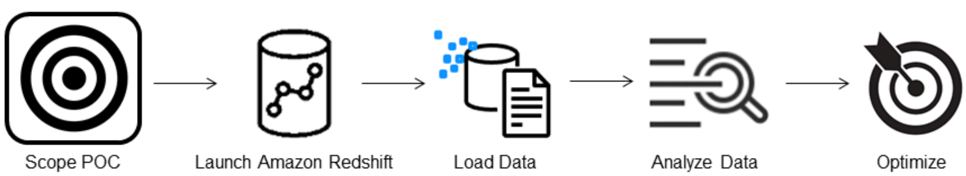
> **Note**
>
> この情報を PDF としてコピーするには、「Amazon Redshift のリソース」ページで [独自の Redshift 概念実証 (POC) を実行] リンクを選択します。

Amazon Redshift の POC を実施する場合、クラス最高レベルのセキュリティ機能、伸縮自在なスケーリング、容易な統合と取り込み、柔軟な分散型データアーキテクチャのオプションなど、さまざまな機能をテスト、実証、採用できます。



POC を成功に導くには、以下のステップを実施します。

ステップ 1: POC の範囲を定義する



POC を実施する際は、独自のデータを使用するか、ベンチマークデータセットを使用するかを選択できます。独自のデータを選択する場合、そのデータに対して独自のクエリを実行します。ベンチマークデータの場合、ベンチマークとともにサンプルクエリが提供されます。独自のデータで POC を実施する準備がまだできていない場合は、「サンプルデータセットを使用する」で詳細を確認してください。

Amazon Redshift の POC については、一般に 2 週間分のデータを使用することをお勧めします。

開始するには、以下を実施します。

1. ビジネス要件と機能要件を特定し、完成形から逆向きに作業します。一般的な例には、パフォーマンスの向上、コスト低減、新しいワークロードや機能のテスト、Amazon Redshift と別のデータウェアハウスの比較などがあります。
2. POC の成功基準となる具体的な目標を設定します。例えば、パフォーマンスの向上の場合、高速化する上位 5 つのプロセスをリストアップして、現在の実行時間と必要となる実行時間を目標に含めます。プロセスは、レポート、クエリ、ETL プロセス、データインジェストなど、現在抱えている課題のいずれかで構いません。
3. テストの実行に必要となる具体的な範囲とアーティファクトを特定します。Amazon Redshift に移行または継続的に取り込む必要があるデータセットと、テストを実行して成功基準と照らし合わせるうえで必要となるクエリとプロセスを特定します。これには、以下の 2 つの方法があります。

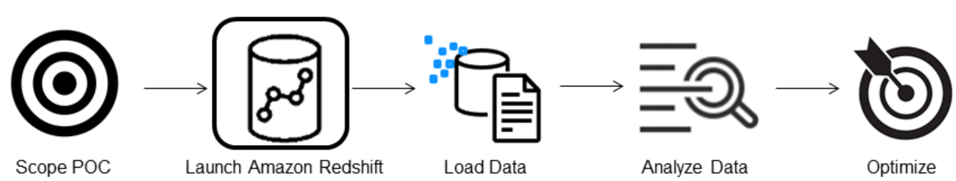
   独自のデータを使用する

   - 独自のデータをテストするには、成功基準を満たすテストに必要となる、実行可能な最小限のデータアーティファクトのリストを作成します。例えば、現在のデータウェアハウスに 200 のテーブルがあり、テストするレポートで必要となるのが 20 のテーブルのみの場合、テーブルの小型のサブセットのみを使用すると POC の実行を迅速化できます。

   サンプルデータセットを使用する

   - 独自のデータセットの準備ができていない場合でも、TPC-DS や TPC-H などの業界標準のベンチマークデータを使用して Amazon Redshift で POC を開始し、サンプルベンチマーククエリを実行し、Amazon Redshift の機能を活用できます。上記のデータセットには、作成後に Amazon Redshift データウェアハウス内からアクセスできます。上記データセットとサンプルクエリにアクセスする方法の詳細な手順については、「ステップ 2: Amazon Redshift を起動する」を参照してください。

ステップ 2: Amazon Redshift を起動する



Amazon Redshift を使用すると、高速、簡単、安全な大規模なクラウドデータウェアハウスを使って、インサイト取得までの時間を短縮できます。Redshift Serverless コンソールでウェアハウスを起動すると、すぐに使用を開始でき、数秒でデータからインサイトを取得できます。Redshift Serverless を使用すると、データウェアハウスの管理にわずらわされることなく、ビジネス成果の実現に集中できます。

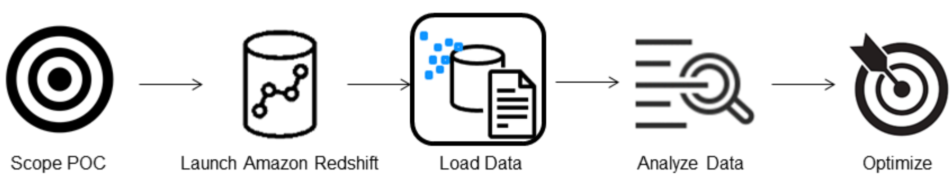
Amazon Redshift Serverless を設定する

Redshift Serverless を初めて使用する場合、コンソールがウェアハウスの起動に必要となる手順をガイドします。また、アカウントでの Redshift サーバーレスの使用状況に対するクレジットの対象となる場合もあります。無料トライアルの選択に関する詳細については、Amazon Redshift 無料トライアルを参照してください。「Amazon Redshift の開始方法」の「Amazon Redshift Serverless によるデータウェアハウスの作成」の手順に従って、Redshift Serverless でデータウェアハウスを作成します。ロードするデータセットがない場合は、このガイドにサンプルデータセットをロードする手順も記載されています。

既にアカウントで Redshift Serverless を起動したことがある場合は、「Amazon Redshift 管理ガイド」の「名前空間を伴うワークグループの作成」の手順に従ってください。ウェアハウスが利用できるようになったら、Amazon Redshift で利用可能なサンプルデータのロードを選択できます。Amazon Redshift クエリエディタ v2 を使用してデータをロードする方法の詳細については、「Amazon Redshift 管理ガイド」の「サンプルデータのロード」を参照してください。

サンプルデータセットをロードする代わりに独自のデータを使用する場合は、「ステップ 3: サンプルデータをロードする」を参照してください。

ステップ 3: サンプルデータをロードする



Redshift Serverless 起動後の次のステップは、POC のためのデータのロードです。シンプルな CSV ファイルをアップロードする場合でも、S3 から半構造化データを取り込む場合でも、データを直接ストリーミングする場合でも、Amazon Redshift は、データをソースから Amazon Redshift テーブルに迅速かつ簡単に移動する柔軟性を提供します。

データのロードには、次のいずれかの方法を選択します。

ローカルファイルをアップロードする

手早くデータを取り込み、分析するには、Amazon Redshift クエリエディタ v2 を使用して、ローカルのデスクトップからデータファイルを簡単にロードできます。Redshift は、CSV、JSON、AVRO、PARQUET、ORC など、さまざまな形式のファイルを処理する機能を備えています。ユーザーが管理者としてクエリエディタ v2 を使用してローカルデスクトップからデータをロードできるようにするには、共有の Amazon S3 バケットを指定し、ユーザーアカウントに適切なアクセス許可を設定する必要があります。ステップバイステップガイダンスについては、「クエリエディタ V2 を使用して Amazon Redshift でのデータロードを簡単かつ安全に行う方法」に従ってください。

Amazon S3 ファイルをロードする

Amazon S3 バケットから Amazon Redshift にデータをロードするには、まず COPY コマンドを使用します。コマンドでは、ソースの Amazon S3 の場所とターゲットの Amazon Redshift テーブルを指定します。指定した Amazon S3 バケットに Amazon Redshift がアクセスできるように、IAM ロールとアクセス許可が適切に設定されていることを確認します。ステップバイステップのガイダンスについては、「チュートリアル: Amazon S3 からデータをロードする」を参照してください。クエリエディタ v2 で [データのロード] オプションを選択して、S3 バケットから直接データをロードすることもできます。

継続的なデータインジェスト

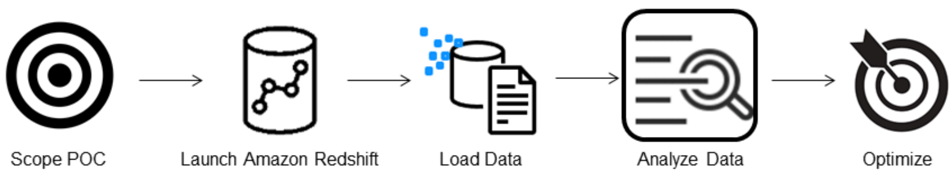
Autocopy (プレビュー中) は、COPY コマンドの拡張機能であり、Amazon S3 バケットからの継続的なデータのロードを自動化します。COPY ジョブを作成すると、Amazon Redshift は、指定されたパスに新しい Amazon S3 ファイルが作成されたことを検出し、ユーザーの操作なしで自動的にロードします。Amazon Redshift は、ロードされたファイルを記録して、ロードされたのは 1 回だけであることを確認します。COPY ジョブの作成手順については、「COPY JOB (プレビュー)」を参照してください。

> **Note**
>
> Autocopy は現時点ではプレビュー段階であり、特定の AWS リージョン でプロビジョンされたクラスターでのみサポートされています。Autocopy 向けのプレビュークラスターを作成するには、「Amazon S3 からの継続的なファイル取り込みによるテーブルのロード (プレビュー)」を参照してください。

ストリーミングデータをロードする

ストリーミング取り込み機能を使用すると、Amazon Kinesis Data Streams と Amazon Managed Streaming for Apache Kafka から Amazon Redshift へのストリームデータの低レイテンシーかつ高速な取り込みができます。Amazon Redshift のストリーミング取り込みは、マテリアライズドビューを使用します。マテリアライズドビューは、自動更新を利用してストリームから直接更新されます。マテリアライズドビューは、ストリームのデータソースにマッピングされています。マテリアライズドビューの定義を行う際には、ストリームデータをフィルタリングしたり集計したりできます。ストリームからデータをロードするステップバイステップのガイダンスについては、「Amazon Kinesis Data Streams の開始方法」または「Amazon Managed Streaming for Apache Kafka からのストリーミング取り込みを開始する」を参照してください。

ステップ 4: データを分析する



Redshift Serverless ワークグループと名前空間を作成し、データをロードしたら、Redshift Serverless コンソールのナビゲーションパネルから [クエリエディタ v2] を開いて、直ちにクエリを実行できます。クエリエディタ v2 を使用して、独自のデータセットに対してクエリ機能をテストしたり、クエリのパフォーマンスをテストしたりできます。

Amazon Redshift クエリエディタ v2 を使用してクエリを実行する

Amazon Redshift コンソールからクエリエディタ v2 にアクセスできます。クエリエディタ v2 を使用してクエリを設定、接続、実行する方法に関する完全なガイドについては、「Amazon Redshift クエリエディタ v2 でデータ分析を簡素化する」を参照してください。

別の方法として、POC の一環で負荷テストを実行する場合は、以下の手順で Apache JMeter をインストールして実行することもできます。

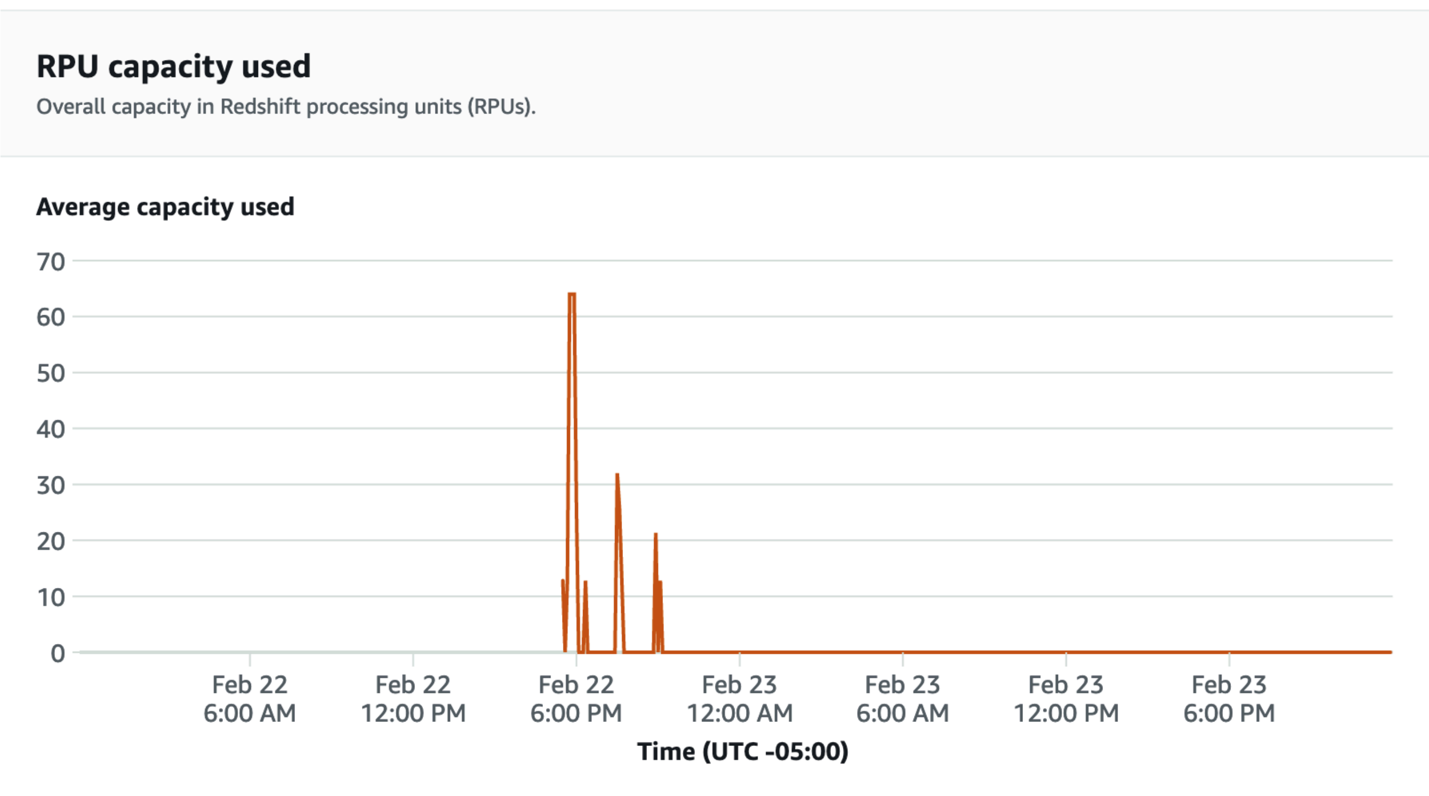
Apache JMeter を使用して負荷テストを実行する

「N」人のユーザーが Amazon Redshift に同時にクエリを送信するシミュレーションを行う負荷テストを実行するには、オープンソースの Java ベースのツールである Apache JMeter を使用できます。

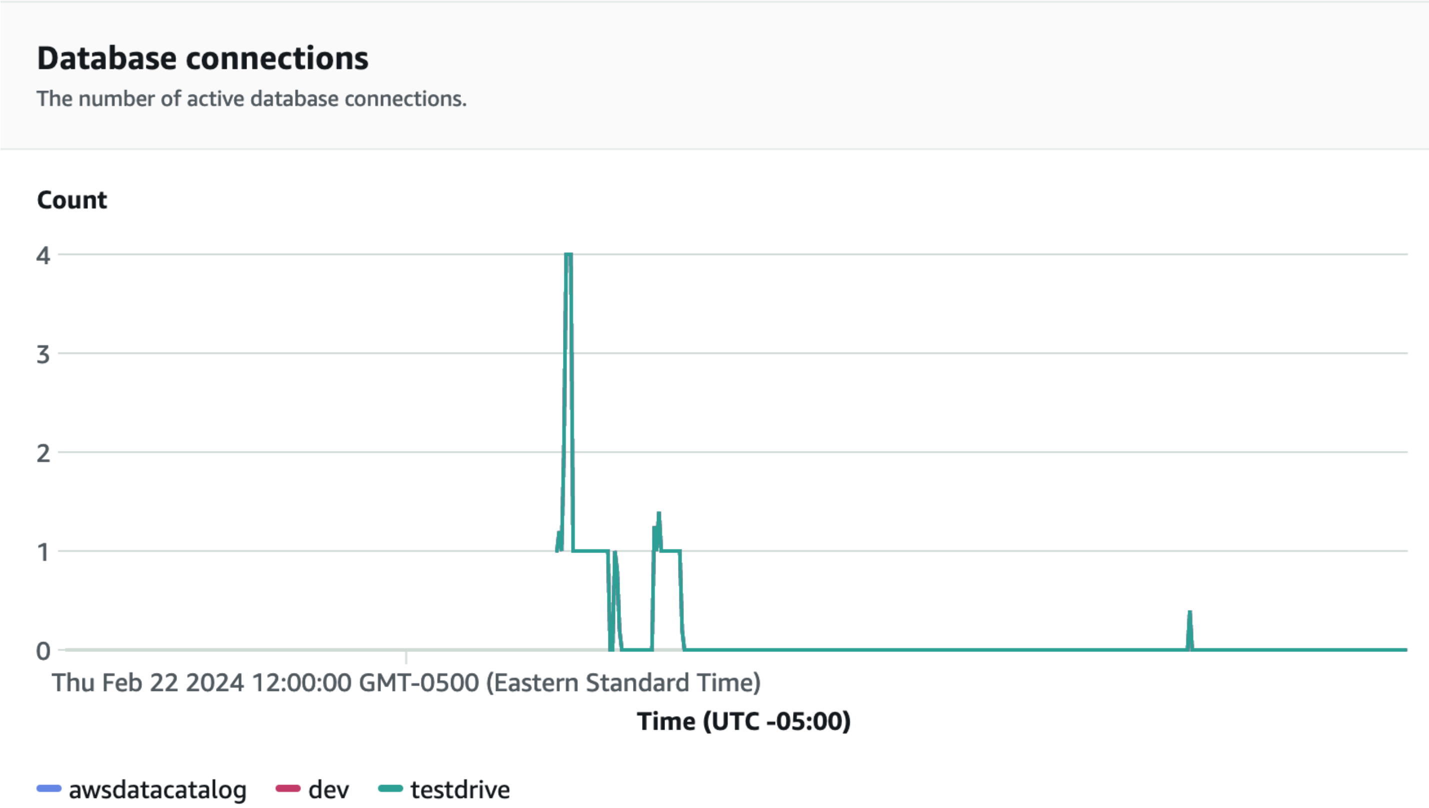
Apache JMeter をインストールして Redshift Serverless ワークグループに対して実行させるように設定するには、「AWS Analytics Automation Toolkit を使用して Amazon Redshift の負荷テストを自動化する」の手順に従ってください。これは、Redshift ソリューションを動的にデプロイするためのオープンソースユーティリティである AWS Analytics Automation toolkit (AAA) を使用して、上記リソースを自動的に起動します。独自のデータを Amazon Redshift にロードした場合は、必ず「ステップ 5 – SQL オプションをカスタマイズする」に従い、テーブルに対してテストすべき適切な SQL ステートメントを明確に指定して実行します。クエリエディタ v2 を使用してこのような SQL ステートメントを各 1 回ずつテストして、エラーなく実行されることを確認します。

SQL ステートメントのカスタマイズを完了し、テスト計画を最終化したら、テスト計画を保存して、Redshift Serverless ワークグループに対して実行します。テストの進行状況をモニタリングするには、Redshift Serverless コンソール を開いて [クエリとデータベースモニタリング] に移動し、[クエリ履歴] タブをクリックして、クエリについての情報を確認します。

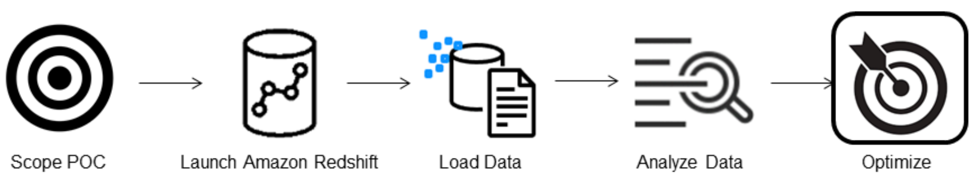
パフォーマンスメトリクスについては、Redshift Serverless コンソールの [データベースパフォーマンス] タブをクリックして、[データベース接続] や [CPU 使用率] などのメトリクスをモニタリングします。このタブでは、使用されている RPU 容量をモニタリングするグラフを表示して、ワークグループでの負荷テスト実行中に、Redshift Serverless が同時ワークロードの需要を満たすために自動的にスケールしているのを確認できます。



負荷テストの実行中にモニタリングすべきもう 1 つの有用なメトリクスに、データベース接続があります。増加するワークロード需要に対応するために、ワークグループが特定の時点で一度に多数の同時接続を処理しているのを確認できます。



ステップ 5: 最適化する



Amazon Redshift は、個別のユースケースをサポートするさまざまな設定と機能を提供しており、毎日数万人ものユーザーがエクサバイトのデータを処理することができます。Amazon Redshift は、このような分析ワークロードを支える基盤となっています。さまざまなオプションの中から選択する際、お客様は Amazon Redshift ワークロードをサポートするための最適なデータウェアハウス設定を決定するうえで役立つツールを求めています。

Test Drive

Test Drive を利用すると、既存のワークロードに対して考えうる設定で自動再生を行い、対応する出力を分析し、ワークロードの移行先として最適なターゲットを評価できます。Test Drive を使用してさまざまな Amazon Redshift の設定を評価する方法については、「Redshift Test Drive を使用してワークロードに最適な Amazon Redshift の設定を判断する」を参照してください。

Amazon Redshift テーブル設計のベストプラクティス

データベースをプランニングする際、テーブル設計に関して、全体的なクエリパフォーマンスに多大な影響を与える重要な決定があります。これらの設計上の選択により、ストレージ要件にも重大な影響があります。その結果、I/O 操作の数が減少し、クエリの処理に必要なメモリが最小限に抑えられるので、クエリパフォーマンスにも影響します。

このセクションでは、最も重要な設計上の決定の概要と、クエリパフォーマンスを最適化するためのベストプラクティスについて説明します。テーブル設計のオプションの詳細な説明と例については、「自動テーブル最適化」を参照してください。

トピック

最良のソートキーの選択

Amazon Redshift は、ソートキーに応じたソート順でデータをディスクに保存します。Amazon Redshift クエリオプティマイザは、最適なクエリプランを決定する際にソート順を使用します。

> **Note**
>
> 自動テーブル最適化を使用する場合、テーブルのソートキーを選択する必要はありません。詳細については、「自動テーブル最適化」を参照してください。

最良のアプローチのために、以下にいくつかの提案を示します。

- Amazon Redshift で適切なソート順を選択するには、ソートキーとして `AUTO` と入力します。
- 最新のデータが最も頻繁にクエリ処理される場合は、タイムスタンプ列をソートキーの主要な列として指定します。

  クエリは時間範囲外のブロック全体をスキップできるので、効率性が高まります。
- 1 つの列に対して範囲フィルタリングまたは等価性フィルタリングを頻繁に実行する場合は、その列をソートキーとして指定します。

  Amazon Redshift は、その列のブロック全体のデータを読み込む必要がありません。これは、各ブロックに保存された列の最小値と最大値を追跡し、述語範囲に当てはまらないブロックをスキップできるためです。
- テーブルを頻繁に結合する場合は、結合列をソートキーとディストリビューションキーの両方として指定します。

  これにより、クエリオプティマイザは、より時間のかかるハッシュ結合ではなくソートマージ結合を選択できるようになります。データが結合キーですでにソートされているので、クエリオプティマイザはソートマージ結合のソートフェーズをバイパスできます。

最適な分散スタイルの選択

クエリを実行すると、必要に応じて結合と集計を実行するために、クエリオプティマイザによって行がコンピューティングノードに再分散されます。テーブル分散スタイルの選択は、クエリを実行する前にデータを必要な場所に配置しておくことによって、再分散ステップの影響を最小限に抑えるために行われます。

> **Note**
>
> 自動テーブル最適化を使用する場合、テーブルのディストリビューションスタイルを選択する必要はありません。詳細については、「自動テーブル最適化」を参照してください。

最良のアプローチのために、以下にいくつかの提案を示します。

1. ファクトテーブルと 1 つのディメンションテーブルをそれらの共通の列に基づいて分散させます。

ファクトテーブルに指定できる分散キーは 1 つだけです。別のキーを使用して結合しているテーブルは、ファクトテーブルとコロケーションされません。結合の頻度と結合列のサイズに基づいてコロケーションする 1 つのディメンションを選択します。ディメンションテーブルのプライマリキーとそれに対応するファクトテーブルの外部キーをいずれも DISTKEY として指定します。

2. フィルタリングされたデータセットのサイズに基づいて最大ディメンションを選択します。

   結合で使用されている行のみが分散される必要があるため、テーブルのサイズではなく、フィルタリング後のデータセットのサイズを考慮します。

3. フィルタリングされた結果セットで高い濃度の行を選択します。

   日付列の販売テーブルを配信する場合、例えば、ほとんどの販売が季節的でなければ、おそらくほぼ均等なデータ分散を取得します。ただし、通常、範囲が制限された述語を使用して狭い日付期間をフィルタリングする場合は、フィルタリングされた行のほとんどはスライス限られたセットになり、クエリのワークロードもスキューされます。

4. ALL 分散を使用するように一部のディメンションテーブルを変更します。

   ファクトテーブルやその他の重要な結合テーブルとディメンションテーブルをコロケーションできない場合は、テーブル全体を全ノードに分散させることによってパフォーマンスを大幅に向上させることができます。ALL 分散を使用すると、ストレージ領域要件の増大、ロード時間の長期化、メンテナンス操作の増加を招くため、ALL 分散を選択する前にすべての要因を比較検討しておく必要があります。

Amazon Redshift が適切なディストリビューションスタイルを選択できるようにするには、ディストリビューションスタイルに `AUTO` を指定します。

ディストリビューションスタイルの選択の詳細については、クエリ最適化のためのデータのディストリビューション を参照してください。

COPY による圧縮エンコードの選択

テーブルを作成するときに圧縮エンコードを指定できますが、ほとんどの場合、自動圧縮が最も適切な結果を生み出します。

ENCODE AUTO は、テーブルのデフォルトです。テーブルが ENCODE AUTO に設定されると、Amazon Redshift は、テーブル内のすべての列の圧縮エンコードを自動的に管理します。詳細については、CREATE TABLEおよびALTER TABLEを参照してください。

COPY コマンドは、データを分析し、ロード操作の一部として自動的に空のテーブルに圧縮エンコードを適用します。

自動圧縮では、圧縮エンコードを選択する際に全体的なパフォーマンスの負荷を分散させます。ソートキー列が、同じクエリ内の他の列よりもかなり高度に圧縮される場合、範囲が制限されたスキャンはパフォーマンスが低下する可能性があります。その結果、自動圧縮では、効率が低い圧縮エンコードを選択してソートキー列が他の列とのバランスの取れた状態を維持します。

テーブルのソートキーが日付またはタイムスタンプで、テーブルが多くの大きな varchar 列を使用しているとします。この場合、ソートキーをまったく圧縮しないことによって、より高いパフォーマンスを得ることができる可能性があります。テーブルで ANALYZE COMPRESSION のコマンドを実行し、エンコーディングを使用して新しいテーブルを作成します。ただし、ソートキーの圧縮エンコードを省いてください。

自動的な圧縮エンコードではパフォーマンスが低下しますが、これが発生するのはテーブルが空で、すでに圧縮エンコードがない場合のみです。存続期間の短いテーブルおよび頻繁に作成するテーブル (ステージングテーブルなど) の場合は、自動圧縮でテーブルを 1 回ロードするか、ANALYZE COMPRESSION コマンドを実行します。次にそれらのエンコードを使用して新しいテーブルを作成します。CREATE TABLE ステートメントにエンコードを追加するか、CREATE TABLE LIKE を使用して同じエンコードで新しいテーブルを作成できます。

詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

プライマリキーおよび外部キーの制約の定義

必要に応じて、テーブル間でのプライマリキーおよび外部キーの制約を定義します。これらの制約は情報提供のみを目的としているものの、クエリオプティマイザはこれらを使用して、より効率的なクエリプランを生成します。

アプリケーションが制約を適用しない限り、プライマリキーおよび外部キーの制約を定義しないでください。Amazon Redshift は、一意性、プライマリキー、および外部キーの制約を強要することはありません。

Amazon Redshift が制約を使用する方法の詳細については、テーブルの制約 を参照してください。

最小列サイズの使用

便宜上の理由で列の最大サイズを使用しないようにしてください。

代わりに、列に保存する可能性が高い最大値を考慮し、それに応じて列のサイズを決定します。例えば、郵便局で使用する米国の州や地域の略語を保存するための CHAR 列に必要なのは、CHAR(2) のみです。

日付列での日時データ型の使用

Amazon Redshift は、DATE および TIMESTAMP データを CHAR または VARCHAR よりも効率的に保存するので、クエリパフォーマンスが向上します。必要な解決策に応じて、日時情報を格納するときに文字型の代わりに DATE または TIMESTAMP データ型を使用します。詳細については、「日時型」を参照してください。

データをロードするための Amazon Redshift のベストプラクティス

大量のデータセットのロードには時間がかかり、大量のコンピューティングリソースを消費する場合があります。データのロード方法はクエリパフォーマンスにも影響を与える可能性があります。このセクションでは、COPY コマンド、一括挿入、ステージングテーブルを使って効率的にデータをロードするベストプラクティスを紹介します。

トピック

データのロードに関するチュートリアル

チュートリアル: Amazon S3 からデータをロードする は、データを Amazon S3 バケットにアップロードしてから COPY コマンドを使用してデータをテーブルにロードするまでの手順を紹介します。このチュートリアルは、ロードエラーをトラブルシューティングするためのヘルプも提供し、また 1 つのファイルのロードと複数のファイルのロードのパフォーマンスの比較も示します。

COPY コマンドを使用したデータのロード

COPY コマンドは、Amazon S3、Amazon EMR、Amazon DynamoDB、またはリモートホスト上の複数のデータソースから同時にデータをロードします。COPY コマンドは INSERT ステートメントを使う場合よりもはるかに効率的に大量のデータをロードし、データを保存します。

COPY コマンドの使用に関する詳細については、「Amazon S3 からデータをロードする」と「Amazon DynamoDB テーブルからのデータのロード」を参照してください。

単一の COPY コマンドを使用した複数のファイルからのロード

Amazon Redshift は、複数の圧縮データファイルからの並列的なデータロードを、自動的に実行します。Amazon S3 オブジェクトプレフィックスまたはマニフェストファイルを利用し、ロードするファイルを指定できます。

ただし、複数の COPY コマンドを同時に使用して複数のファイルから 1 つのテーブルにデータをロードする場合には、Amazon Redshift はそれらの読み込みを直列的に実行します。この種類のロードはかなり低速で、テーブルにソート列が定義されている場合は、最後に VACUUM プロセスが必要になります。COPY の使用によるデータのロードの詳細については、「Amazon S3 からデータをロードする」を参照してください。

データファイルをロードする

ソースデータファイルにはさまざまな形式があり、さまざまな圧縮アルゴリズムが使用されます。COPY コマンドを使用してデータをロードすると、Amazon Redshift は Amazon S3 バケットプレフィックスで参照されるすべてのファイルをロードします。プレフィックスは、オブジェクトキー名の先頭にある文字列です。プレフィックスが複数のファイルまたは分割可能なファイルを指している場合、Amazon Redshift は Amazon Redshift の MPP アーキテクチャを利用してデータを並列でロードします。これにより、クラスター内のノード間でワークロードが分割されます。一方、分割できないファイルのデータをロードする場合、Amazon Redshift は直列的なロードを実行します。この処理はかなり低速になります。以下のセクションでは、フォーマットと圧縮に応じて、さまざまなファイルタイプを Amazon Redshift にロードする推奨方法について説明します。

分割可能なファイルからのデータのロード

次のファイルは、データがロードされると自動的に分割できます。

- 非圧縮の CSV ファイル
- BZIP で圧縮された CSV ファイル
- 列形式ファイル (Parquet/ORC)

Amazon Redshift は、128 MB 以上のファイルを自動的にチャンクに分割します。列形式ファイル (特に Parquet と ORC) では、128MB 未満の場合は分割されません。Redshift は、並列動作するスライスを利用してデータをロードします。これにより、ローディングのパフォーマンスが高速化されます。

分割できないファイルからのデータのロード

JSON や CSV などのファイルタイプを GZIP などの他の圧縮アルゴリズムで圧縮しても、自動的に分割されません。この場合、圧縮されたデータを、ほぼ等しいサイズ (1 MB から 1 GB 程度) の複数の小さいファイルに手動で分割することをお勧めします。ファイルの数はクラスター内のスライス数の倍数である必要があります。データを複数のファイルに分割する方法と COPY を使ってデータをロードする例の詳細については、「Amazon S3 からデータをロードする」を参照してください。

データファイルを圧縮する

大きなロードファイルを圧縮するときは、gzip、lzop、bzip2、または Zstandard を使用してそれらを圧縮し、データを複数の小さなファイルに分割することをお勧めします。

COPY コマンドとともに GZIP、LZOP、BZIP2、ZSTD オプションを指定します。この例では、パイプで区切られた LZOP ファイルから TIME テーブルをロードしています。

```
copy time
from 's3://amzn-s3-demo-bucket/data/timerows.lzo'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
lzop
delimiter '|';
```

非圧縮データファイルを、分割しなくても良い場合もあります。データの分割、ならびに COPY を使用したデータのロード例については、「Amazon S3 からデータをロードする」を参照してください。

ロードの前後におけるデータファイルの検証

Amazon S3 からデータをロードする前に、最初に Amazon S3 バケットに正しいファイルのみがすべて含まれることを確認します。詳細については、「必要なファイルがバケットにあることを確認する」を参照してください。

ロード操作が完了したら、STL_LOAD_COMMITS システムテーブルにクエリし、ファイルが予定どおりロードされたことを確認します。詳細については、「データが正しくロードされたことを確認する」を参照してください。

複数行の挿入の使用

COPY コマンドを選択できないときに SQL 挿入が必要な場合、可能であれば、複数行の挿入を使用します。一度にたった 1 行または数行ずつデータを追加するとき、データ圧縮は効率的ではありません。

複数行の挿入を使用すれば、一連の挿入を一括処理することでパフォーマンスが改善します。次の例では、シングル INSERT ステートメントを使い、3 つの行を 4 列のテーブルに挿入しています。これは、複数行の挿入の構文を図示するためだけの小規模な挿入例にすぎません。

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);
```

詳細と例については、「INSERT」を参照してください。

一括挿入の使用

SELECT 句とともに一括挿入操作を使用すれば、データ挿入のパフォーマンスが向上します。

テーブル間でデータまたはデータのサブセットを移動する必要があるとき、INSERT および CREATE TABLE AS コマンドを使用します。

例えば、次の INSERT ステートメントでは、CATEGORY テーブルからすべての行が選択され、CATEGORY_STAGE テーブルに挿入されます。

```
insert into category_stage
(select * from category);
```

次の例では、CATEGORY_STAGE が CATEGORY のコピーとして作成され、CATEGORY のすべての行が CATEGORY_STAGE に挿入されます。

```
create table category_stage as
select * from category;
```

ソートキー順序でのデータのロード

ソートキー順序でデータをロードし、バキュームの必要性をなくします。

新しいデータの各バッチがテーブルの既存の行に続く場合、データはソート順序で適切に保存され、バキュームを実行する必要がありません。COPY により入ってくるデータの各バッチがロード時にソートされるため、各ロードの行を事前にソートする必要はありません。

例えば、本日のアクティビティに基づいて毎日データをロードするとします。ソートキーがタイムスタンプ列の場合、データはソート順に保存されます。この順序が発生するのは、現在の日付のデータは常に前日のデータの末尾に付加されるためです。詳細については、「ソートキー順序でデータをロードする」を参照してください。バキュームオペレーションの詳細については、「テーブルのバキューム処理」を参照してください。

順次ブロックでのデータのロード

大量のデータを追加する必要がある場合、ソート順序に基づいて順次ブロックでデータをロードすればバキュームの必要性がなくなります。

例えば、2017 年 1 月から 2017 年 12 月までのイベントのあるテーブルをロードする必要があるとします。各月が 1 つのファイルに含まれていると仮定して、1 月、2 月などと、行をロードします。ロードが完了したとき、テーブルは完全にソートされています。バキュームを実行する必要はありません。詳細については、「時系列テーブルの使用」を参照してください。

非常に大量のデータセットをロードするとき、ソートに必要な領域が利用可能な領域の合計を超えることがあります。小さなブロックでデータをロードすれば、各ロードの際、中間のソート領域が少なくて済みます。また、小さなブロックをロードする場合、COPY が失敗してロールバックされるときに簡単に再開できます。

時系列テーブルの使用

データの保存期間が固定されている場合、時系列テーブルの順序でデータを整理することができます。このような順序では、各テーブルは同一であっても、さまざまな時間範囲のデータが含まれます。

該当するテーブルで DROP TABLE コマンドを実行することで、古いデータを簡単に削除できます。この手法は、大規模な DELETE プロセスを実行するよりもはるかに高速で、スペースを回復するためにそれ以降の VACUUM プロセスを実行する手間が省けます。データが異なるテーブルに保存されているというファクトを非表示にするために、UNION ALL ビューを作成できます。古いデータを削除するとき、UNION ALL ビューを微調整し、ドロップしたテーブルを削除します。同様に、新しい期間を新しいテーブルにロードするとき、新しいテーブルをこのビューに追加します。クエリフィルタと一致しないテーブルでスキャンをスキップするようオプティマイザにシグナルを送信するには、ビュー定義で各テーブルに対応する日付範囲をフィルタします。

UNION ALL ビューでのテーブルが多くなりすぎないようにします。テーブルを追加するたびに、クエリにわずかな処理時間が追加されます。テーブルは、同じタイムフレームを使用する必要はありません。例えば、日次、月次、年次など、さまざまな期間のテーブルがあるとします。

ソートキーのタイムスタンプ列のある時系列テーブルを使用する場合、ソートキー順序でデータを効果的にロードします。それにより、バキュームでデータを再ソートする必要がなくなります。詳細については、「ソートキー順序でデータをロードする」を参照してください。

保守管理の時間枠を回避したスケジュール計画

スケジュール計画された保守管理がクエリの実行中に発生した場合、クエリは終了し、ロールバックされます。クエリをやり直す必要があります。大規模なデータロードや VACUUM 操作などの長時間実行される操作のスケジュールは、保守管理の時間枠を避けて設定します。小規模の増分でデータをロードし、VACUUM 操作の規模を管理することで、リスクを最小限に抑え、やり直しが必要な場合、簡単にやり直すこともできます。詳細については、「順次ブロックでのデータのロード」および「テーブルのバキューム処理」を参照してください。

Amazon Redshift クエリの設計のベストプラクティス

クエリパフォーマンスを最大化するには、クエリの作成時に次の推奨事項に従います。

- ベストプラクティスに従ってテーブルを設計し、クエリパフォーマンスの強固な基盤を提供します。詳細については、「Amazon Redshift テーブル設計のベストプラクティス」を参照してください。
- `select *` は使用しないでください。明確に必要な列のみを含めます。
- 同じテーブルから複数回選択するのではなく、CASE 条件式 を使用して複雑な集計を実行します。

- どうしても必要でなければクロス結合を使用しないでください。これらの結合に結合条件がないと、2 つのテーブルのデカルト積が求められます。クロス結合は通常ネステッドループ結合として実行されますが、これは使用可能な結合タイプで最も低速な結合です。
- クエリのテーブル 1 つが述語条件のみで使用されている場合には、サブクエリを使用します。この場合、サブクエリはわずかな数の行を返します (ほぼ 200 未満)。次の例では、LISTING テーブルへの結合を回避するためにサブクエリを使用します。

```
select sum(sales.qtysold)
from sales
where salesid in (select listid from listing where listtime > '2008-12-26');
```

- 述語を使用してデータセットをできるだけ制限します。
- 述語で、できる限りコストのかからない演算子を使用します。比較条件 演算子には、LIKE 演算子が推奨されます。SIMILAR TO または POSIX 演算子 には、引き続き LIKE 演算子が推奨されます。
- クエリ述語で関数を使用しないでください。それらを使用すると、クエリの中間ステップの解決に多くの行が必要になるため、クエリのコストが上昇する可能性があります。
- 可能な場合、データセットを制限するために、WHERE 句を使用します。次に、クエリプランナーは行の順序を使用して、条件に一致するレコードを判断できるため、多数のディスクブロックのスキャンをスキップできます。これを行わない場合、クエリ実行エンジンは該当する列を完全にスキャンする必要があります。
- 述語が同じフィルタを適用する場合でも、述語を追加して結合に関与するテーブルをフィルタリングします。クエリは同じ結果セットを返しますが、Amazon Redshift はスキャンステップの前に結合テーブルをフィルタリングし、それらのテーブルで効率的にブロックのスキャンを省略できます。結合条件で使用された列をフィルタリングする場合、冗長フィルタは必要ありません。

  例えば、`SALES` と `LISTING` に結合して、12 月以降にリストされ、販売者別に分類されたチケットを検索するとします。両方のテーブルは日付でソートされています。次のクエリは、共通キーのテーブルを結合し、12 月 1 日より後の `listing.listtime` 値をフィルタリングします。

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
group by 1 order by 1;
```

sales.saletime の述語が WHERE 句に含まれないため、実行エンジンは SALES テーブル全体をスキャンしなければならなくなります。フィルタによって結合に関与する行が少なくなることがわかっている場合は、そのフィルタも追加します。次の例では、実行時間が大幅に減ります。

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
and sales.saletime > '2008-12-01'
group by 1 order by 1;
```

- GROUP BY 句でソートキーを使用すると、クエリプランナーがより効率的な集計を使用できます。GROUP BY リストにソートキー列のみが含まれており、そのうち 1 つが分散キーでもある場合、クエリは 1 フェーズ集計の対象となる可能性があります。GROUP BY リストのソートキー列には、1 番目のソートキーの後に、ソートキー順序で使用する他のソートキーが含まれている必要があります。例えば、1 番目のソートキーを使用する、1 番目と 2 番目のソートキーを使用する、1 番目、2 番目、3 番目のソートキーを使用する、などが有効です。1 番目と 3 番目のソートキーを使用することは有効ではありません。

  EXPLAIN コマンドを実行し、クエリの集計ステップで XN GroupAggregate を探すことにより、1 フェーズ集計の使用を確認できます。

- GROUP BY 句と ORDER BY 句の両方を使用する場合、必ずどちらにも同じ順序で列を配置してください。つまり、次の方法を使用します。

```
group by a, b, c
order by a, b, c
```

  次の方法は使用しないでください。

```
group by b, c, a
order by a, b, c
```

Amazon Redshift Advisor からのレコメンデーションの使用

Amazon Redshift クラスターのパフォーマンスを向上させて運用コストを減らすため、Amazon Redshift Advisor は変更に関する具体的なレコメンデーションを提供します。Advisor は、クラスターのパフォーマンスおよび使用状況メトリクスを分析して、カスタマイズされたレコメンデーショ

ンを作成します。これらのカスタマイズされたレコメンデーションは、運用とクラスターの設定に関連しています。最適化に優先順位を付けられるように、影響度の順にレコメンデーションがランク付けされます。

Advisor は、パフォーマンス統計または運用データに関する確認に基づいてレコメンデーションを作成します。クラスターでテストを実行して確認を行い、テスト値が指定範囲内にあるかどうか判断されます。テスト結果がその範囲外の場合、クラスターの確認が生成されます。同時に、確認された値をベストプラクティスの範囲にするためのレコメンデーションが作成されます。表示されるのは、パフォーマンスや運用に大きな影響を与えるレコメンデーションのみです。レコメンデーションが対応済みであると判断されると、レコメンデーションのリストからその事項が削除されます。

例えば、データウェアハウスに多数の非圧縮テーブル列が含まれているとします。この場合は、`ENCODE` パラメータを使用して列の圧縮を指定してテーブルを再構築することで、クラスターのストレージコストを低減できます。別の例として、クラスターに含まれている非圧縮テーブルのデータが膨大であると Advisor が確認したとします。この場合、圧縮の候補であるテーブルの列と、それらの列を圧縮する方法を示すリソースを見つけるための SQL コードブロックが提供されます。

Advisor がサポートされている Amazon Redshift リージョン

Amazon Redshift Advisor の機能は以下の AWS リージョンでのみご利用いただけます。

- 米国東部 (バージニア北部) リージョン (us-east-1)
- 米国東部 (オハイオ) リージョン (us-east-2)
- 米国西部 (北カリフォルニア) リージョン (us-west-1)
- 米国西部 (オレゴン) リージョン (us-west-2)
- アフリカ (ケープタウン) リージョン (af-south-1)
- アジアパシフィック (香港) リージョン (ap-east-1)
- アジアパシフィック (ハイデラバード) リージョン (ap-south-2)
- アジアパシフィック (ジャカルタ) リージョン (ap-southeast-3)
- アジアパシフィック (メルボルン) リージョン (ap-southeast-4)
- アジアパシフィック (ムンバイ) リージョン (ap-south-1)
- アジアパシフィック (大阪) リージョン (ap-northeast-3)
- アジアパシフィック (ソウル) リージョン (ap-northeast-2)
- アジアパシフィック (シンガポール) リージョン (ap-southeast-1)
- アジアパシフィック (シドニー) リージョン (ap-southeast-2)

- アジアパシフィック (東京) リージョン (ap-northeast-1)
- カナダ (中部) リージョン (ca-central-1)
- カナダ西部 (カルガリー) — リージョン (ca-west-1)
- 中国 (北京) リージョン (cn-north-1)
- 中国 (寧夏) リージョン (cn-northwest-1)
- 欧州 (フランクフルト) リージョン (eu-central-1)
- 欧州 (アイルランド) リージョン (eu-west-1)
- 欧州 (ロンドン) リージョン (eu-west-2)
- 欧州 (ミラノ) リージョン (eu-south-1)
- 欧州 (パリ) リージョン (eu-west-3)
- 欧州 (スペイン) リージョン (eu-south-2)
- 欧州 (ストックホルム) リージョン (eu-north-1)
- 欧州 (チューリッヒ) リージョン (eu-central-2)
- イスラエル (テルアビブ) リージョン (il-central-1)
- 中東 (バーレーン) リージョン (me-south-1)
- 中東 (UAE) リージョン (me-central-1)
- 南米 (サンパウロ) リージョン (sa-east-1)

トピック

Amazon Redshift Advisor レコメンデーションの表示

Amazon Redshift Advisor レコメンデーションにアクセスするには、Amazon Redshift コンソール、Amazon Redshift API、または AWS CLI を使用できます。レコメンデーションにアクセスするには、IAM ロールまたは ID にアクセス許可 `redshift:ListRecommendations` がアタッチされている必要があります。

Amazon Redshift プロビジョンドコンソールでの Amazon Redshift Advisor レコメンデーションの表示

AWS Management Console で Amazon Redshift Advisor レコメンデーションを表示できます。

コンソールで Amazon Redshift クラスターに関する Amazon Redshift Advisor レコメンデーションを表示するには

1. AWS Management Consoleにサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Advisor] (アドバイザー) を選択します。
3. 各レコメンデーションを展開して、詳細情報を確認します。このページでは、レコメンデーションをソートしたりグループ化したりすることができます。

Amazon Redshift API オペレーションを使用した Amazon Redshift Advisor レコメンデーションの表示

Amazon Redshift API を使用して、Amazon Redshift クラスターに関する Amazon Redshift Advisor レコメンデーションを一覧表示できます。通常、選択したプログラミング言語でアプリケーションを開発するには、AWS SDK を使用して `redshift:ListRecommendations` API を呼び出します。詳細については、「Amazon Redshift API リファレンス」の「ListRecommendations」を参照してください。

AWS Command Line Interface オペレーションを使用した Amazon Redshift Advisor レコメンデーションの表示

AWS Command Line Interface を使用して、Amazon Redshift クラスターに関する Amazon Redshift Advisor レコメンデーションを一覧表示できます。詳細については、「AWS CLI コマンドリファレンス」の「list-recommendations」を参照してください。

Amazon Redshift Advisor のレコメンデーション

Amazon Redshift Advisor は、Amazon Redshift クラスターを最適化してパフォーマンスを高め、運用コストを低減するための方法に関するレコメンデーションを提供します。前述したように、各レコメンデーションの説明はコンソールに表示されます。これらのレコメンデーションの詳細は、以下のセクションで説明します。

トピック

COPY によってロードされた Amazon S3 ファイルオブジェクトを圧縮する

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを活用して、データを並列で読み込み、ロードします。このコマンドは Amazon S3 のファイル、DynamoDB テーブル、および 1 つ以上のリモートホストからのテキスト出力を読み込むことができます。

大量のデータをロードするときは、COPY コマンドを使用して、S3 から圧縮されたデータファイルをロードすることを強くお勧めします。大きなデータセットを圧縮することで、ファイルを Amazon S3 にアップロードする時間を短縮できます。また、COPY を使えば、読み込み時にファイルが解凍され、ロードプロセスが高速化されます。

分析

大きな非圧縮データセットをよくロードする、長時間実行される COPY コマンドでは、かなりパフォーマンスが向上する可能性があります。Advisor の分析では、大きな非圧縮データセットをロードする COPY コマンドが識別されます。このような場合、Advisor は Amazon S3 のソースファイルで圧縮を実装するためのレコメンデーションを生成します。

レコメンデーション

大量のデータをロードするか、かなり長時間実行される各 COPY で、Amazon S3 から圧縮されたデータオブジェクトが取り込まれることを確認します。スーパーユーザーとして次の SQL コマンドを実行して、Amazon S3 から大きな非圧縮データセットをロードする COPY コマンドを識別できます。

```
SELECT
    wq.userid, query, exec_start_time AS starttime, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
```

```
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY 1, 2, 3, 7
HAVING SUM(transfer_size) = SUM(data_size)
AND SUM(transfer_size)/(1024*1024) >= 5
ORDER BY 6 DESC, 5 DESC;
```

ロード後に、ステージングされたデータが Amazon S3 に残る場合 (データレイクアーキテクチャでは一般的です)、圧縮形式でこのデータを保存すると、ストレージコストを削減できます。

実装のヒント

- 最適なオブジェクトサイズは、圧縮後で 1～128 MB の間です。
- gzip、lzop、または bzip2 形式でファイルを圧縮できます。

複数のアクティブなデータベースを分離する

ベストプラクティスとして、Amazon Redshift のデータベースを相互に分離することをお勧めします。特定のデータベースで実行されるクエリは、クラスターの他のデータベースからのデータにアクセスできません。ただし、クラスターのすべてのデータベースで実行するクエリは、基盤となる同じストレージスペースとコンピューティングリソースを共有します。1 つのクラスターに複数のアクティブなデータベースが含まれる場合、そのワークロードは通常は無関係です。

分析

Advisor の分析では、同時に実行中のアクティブなワークロードに対するクラスター上のすべてのデータベースが確認されます。同時に実行中のアクティブなワークロードがある場合、Advisor は別の Amazon Redshift クラスターへのデータベースの移行を検討するためのレコメンデーションを生成します。

レコメンデーション

アクティブにクエリが実行されている各データベースを、別の専用クラスターに移動することを検討します。別のクラスターを使用するとリソースの競合が減少し、クエリのパフォーマンスが向上します。これが可能であるのは、各ワークロードのストレージ、コスト、およびパフォーマンスのニーズ

に対して各クラスターのサイズを設定できるためです。また、多くの場合、関連しないワークロードでさまざまなワークロード関連設定の利点を得ることができます。

アクティブに使用されているデータベースを識別するには、スーパーユーザーとしてこの SQL コマンドを実行できます。

```
SELECT database,
  COUNT(*) as num_queries,
  AVG(DATEDIFF(sec,starttime,endtime)) avg_duration,
  MIN(starttime) as oldest_ts,
  MAX(endtime) as latest_ts
FROM stl_query
WHERE userid > 1
GROUP BY database;
```

実装のヒント

- ユーザーは各データベースに接続する必要があり、クエリは 1 つのデータベースのみにしかアクセスできないため、別のクラスターにデータベースを移動しても、ユーザーへの影響は最小限で済みます。
- データベースを移動する 1 つのオプションは、次のステップを実行することです。
  1. 現在のクラスターのスナップショットを、同じサイズのクラスターに一時的に復元します。
  2. 移動対象のデータベースを除き、新しいクラスターからすべてのデータベースを削除します。
  3. データベースのワークロードに合わせて、クラスターのサイズを適切なノードタイプと数に変更します。

ワークロード管理 (WLM) メモリを再割り当てする

Amazon Redshift はユーザークエリを処理のために 手動 WLM を実装する にルーティングします。ワークロード管理 (WLM) は、クエリがキューにルーティングされる方法を定義します。Amazon Redshift は、各キューにクラスターの使用可能なメモリの一部を割り当てます。キューのメモリは、キューのクエリスロットに分けられます。

ワークロードが必要とするよりも多くのスロットでキューが設定されると、それらの未使用スロットに割り当てられたメモリの使用率が低くなります。設定されたスロットをワークロードの要件に合わせて減らすと、使用率の低いメモリがアクティブなスロットに再分散され、クエリのパフォーマンスが向上します。

分析

Advisor の分析では、ワークロードの同時実行要件を確認し、未使用スロットがあるクエリキューを識別します。Advisor は、以下が見つかると、キューのスロット数を減らすためのレコメンデーションを生成します。

- 分析全体で完全に無効なスロットがあるキュー
- 分析全体で少なくとも 2 つの無効なスロットがあった、4 つを超えるスロットがあるキュー。

レコメンデーション

設定されたスロットをピーク時のワークロード要件に合わせて減らすと、使用率が低いメモリがアクティブなスロットに再分散されます。スロットが十分に利用されていないキューに対して設定されたスロット数を減らすことを検討します。それらのキューを確認するには、スーパーユーザーとして次の SQL コマンドを実行することにより、各キューの 1 時間あたりのピークスロット要件を比較できます。

```
WITH
 generate_dt_series AS (select sysdate - (n * interval '5 second') as dt from (select
 row_number() over () as n from stl_scan limit 17280)),
 apex AS (
     SELECT iq.dt, iq.service_class, iq.num_query_tasks, count(iq.slot_count) as
 service_class_queries, sum(iq.slot_count) as service_class_slots
     FROM
         (select gds.dt, wq.service_class, wscc.num_query_tasks, wq.slot_count
         FROM stl_wlm_query wq
         JOIN stv_wlm_service_class_config wscc ON (wscc.service_class =
 wq.service_class AND wscc.service_class > 5)
         JOIN generate_dt_series gds ON (wq.service_class_start_time <= gds.dt AND
 wq.service_class_end_time > gds.dt)
         WHERE wq.userid > 1 AND wq.service_class > 5) iq
     GROUP BY iq.dt, iq.service_class, iq.num_query_tasks),
     maxes as (SELECT apex.service_class, trunc(apex.dt) as d, date_part(h,apex.dt) as
 dt_h, max(service_class_slots) max_service_class_slots
                     from apex group by apex.service_class, apex.dt,
 date_part(h,apex.dt))
SELECT apex.service_class - 5 AS queue, apex.service_class, apex.num_query_tasks AS
 max_wlm_concurrency, maxes.d AS day, maxes.dt_h || ':00 - ' || maxes.dt_h || ':59' as
 hour, MAX(apex.service_class_slots) as max_service_class_slots
FROM apex
```

```
JOIN maxes ON (apex.service_class = maxes.service_class AND apex.service_class_slots =
 maxes.max_service_class_slots)
GROUP BY  apex.service_class, apex.num_query_tasks, maxes.d, maxes.dt_h
ORDER BY apex.service_class, maxes.d, maxes.dt_h;
```

max_service_class_slots 列はその時間のクエリキューの WLM クエリスロットの最大数を表します。使用率の低いキューが存在する場合は、「Amazon Redshift 管理ガイド」に記載されているようにパラメータグループを変更して、スロット削減の最適化を実装します。

実装のヒント

- ワークロードのボリュームが非常に変わりやすい場合は、分析でピークの使用期間をキャプチャしたことを確認します。そうでない場合は、前の SQL を繰り返し実行し、ピークの同時実行要件をモニタリングします。
- 前の SQL コードからのクエリ結果の解釈の詳細については、GitHub で wlm_apex_hourly.sql スクリプトについて参照してください。

COPY 中に圧縮分析をスキップする

COPY コマンドで宣言された圧縮エンコードで空のテーブルにデータをロードすると、Amazon Redshift はストレージの圧縮を適用します。この最適化により、エンドユーザーによってロードされた場合でも、クラスターのデータが効率的に保存されます。圧縮を適用するために必要な分析には、かなり時間がかかる場合があります。

分析

Advisor の分析では、自動圧縮分析で遅れた COPY 操作をチェックします。この分析では、ロード中のデータをサンプリングして、圧縮エンコードが決定されます。このサンプリングは、ANALYZE COMPRESSION コマンドで実行されるサンプリングに似ています。

夜間の抽出、変換、ロード (ETL) バッチなどで、構造化プロセスの一部としてデータをロードする場合は、事前に圧縮を定義できます。また、悪影響なくこのフェーズを完全にスキップするようにテーブル定義を最適化することもできます。

レコメンデーション

圧縮分析フェーズをスキップして COPY の応答性を向上させるには、次の 2 つのオプションのいずれかを実装します。

- COPY コマンドを使用してロードするテーブルを作成するときに、列の ENCODE パラメータを使用します。
- COPY コマンドで COMPUPDATE OFF パラメータを指定して、圧縮を完全に無効にします。

通常、最適なソリューションはテーブルの作成中に列のエンコーディングを使用することです。この手法により、圧縮されたデータをディスクに保存する利点も維持できます。ANALYZE COMPRESSION コマンドを使用して圧縮エンコーディングを提案できますが、これらのエンコーディングを適用するには、テーブルを再作成する必要があります。このプロセスを自動化するには、GitHub にある AWS の ColumnEncodingUtility を使用できます。

自動圧縮分析をトリガーした最近の COPY 操作を特定するには、次の SQL コマンドを実行します。

```
  WITH xids AS (
    SELECT xid FROM stl_query WHERE userid>1 AND aborted=0
    AND querytxt = 'analyze compression phase 1' GROUP BY xid
    INTERSECT SELECT xid FROM stl_commit_stats WHERE node=-1)
SELECT a.userid, a.query, a.xid, a.starttime, b.complyze_sec,
    a.copy_sec, a.copy_sql
FROM (SELECT q.userid, q.query, q.xid, date_trunc('s',q.starttime)
    starttime, substring(querytxt,1,100) as copy_sql,
    ROUND(datediff(ms,starttime,endtime)::numeric / 1000.0, 2) copy_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt ilike 'copy %from%' OR querytxt ilike '% copy %from%')
    AND querytxt not like 'COPY ANALYZE %') a
LEFT JOIN (SELECT xid,
    ROUND(sum(datediff(ms,starttime,endtime))::numeric / 1000.0,2) complyze_sec
    FROM stl_query q JOIN xids USING (xid)
    WHERE (querytxt like 'COPY ANALYZE %'
    OR querytxt like 'analyze compression phase %')
    GROUP BY xid ) b ON a.xid = b.xid
WHERE b.complyze_sec IS NOT NULL ORDER BY a.copy_sql, a.starttime;
```

実装のヒント

- ETL プロセス (ステージングテーブル、一時テーブルなど) 中に作成される、サイズが非常に大きいすべてのテーブルで、最初のソートキーを除くすべての列に対して必ず圧縮エンコーディングを宣言します。

- 前の SQL コマンドによって識別された COPY コマンドごとに、ロード中のテーブルで予想される存続期間の長さを予測します。テーブルが非常に小さいまま維持されることが確実である場合は、`COMPUPDATE OFF` パラメータを使用して圧縮を完全にオフにします。それ以外の場合は、COPY コマンドを使用して、ロードする前に明示的な圧縮を使用したテーブルを作成します。

COPY によってロードされた Amazon S3 オブジェクトを分割する

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを活用して、Amazon S3 のファイルからデータを読み込み、ロードします。COPY コマンドは、複数のファイルから同時にデータをロードして、クラスターのノード間でワークロードを分散します。最適なスループットを達成するには、データを複数のファイルに分割し、並列処理の長所を最大限に活用することをお勧めします。

分析

Advisor の分析では、Amazon S3 でステージングされた少数のファイルに含まれる大きなデータセットをロードする COPY コマンドが識別されます。少数のファイルから大きなデータセットをよくロードする、長時間実行される COPY コマンドでは、かなりパフォーマンスが向上する可能性があります。これらの COPY コマンドにかなり長い時間がかかっていると Advisor が判断した場合、Amazon S3 の追加のファイルにデータを分割して、並列処理性を高めるためのレコメンデーションが作成されます。

レコメンデーション

この場合は、次のアクションを実行することをお勧めします。これらは優先順位が高い順に示されています。

1. クラスターのノード数よりも少ないファイルをロードする COPY コマンドを最適化します。
2. クラスターのスライス数よりも少ないファイルをロードする COPY コマンドを最適化します。
3. ファイルの数がクラスタースライス数の倍数でない場合は、COPY コマンドを最適化します。

特定の COPY コマンドは、大量のデータをロードするか、かなり長時間実行されます。これらのコマンドについては、クラスターのスライス数の倍数と等しい数のデータオブジェクトを Amazon S3 からロードすることをお勧めします。COPY コマンドでロードされた S3 オブジェクトの数を確認するため、スーパーユーザーとして次の SQL コードを実行します。

```
SELECT
    query, COUNT(*) num_files,
    ROUND(MAX(wq.total_exec_time/1000000.0),2) execution_secs,
    ROUND(SUM(transfer_size)/(1024.0*1024.0),2) total_mb,
    SUBSTRING(querytxt,1,60) copy_sql
FROM stl_s3client s
JOIN stl_query q USING (query)
JOIN stl_wlm_query wq USING (query)
WHERE s.userid>1 AND http_method = 'GET'
    AND POSITION('COPY ANALYZE' IN querytxt) = 0
    AND aborted = 0 AND final_state='Completed'
GROUP BY query, querytxt
HAVING (SUM(transfer_size)/(1024*1024))/COUNT(*) >= 2
ORDER BY CASE
WHEN COUNT(*) < (SELECT max(node)+1 FROM stv_slices) THEN 1
WHEN COUNT(*) < (SELECT COUNT(*) FROM stv_slices WHERE node=0) THEN 2
ELSE 2+((COUNT(*) % (SELECT COUNT(*) FROM stv_slices))/(SELECT COUNT(*)::DECIMAL FROM
 stv_slices))
END, (SUM(transfer_size)/(1024.0*1024.0))/COUNT(*) DESC;
```

実装のヒント

- ノードのスライスの数は、クラスターのノードサイズによって決まります。さまざまなノードタイプのスライス数については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクラスターとノード」を参照してください。
- セットに共通プレフィックスまたはプレフィックスキーを指定するか、マニフェストファイルにファイルのリストを明示的に指定することで、複数のファイルをロードできます。ファイルのロードについて詳しくは、「圧縮および非圧縮のファイルからのデータのロード」を参照してください。
- Amazon Redshift はワークロードを分割するときにファイルサイズを考慮しません。ロードデータファイルを分割して大体同じサイズにし、圧縮後に 1 MB ～ 1 GB になるようにします。

テーブル統計の更新

Amazon Redshift はコストベースのクエリオプティマイザを使用して、クエリに対して最適な実行プランを選択します。コストの予測は、ANALYZE コマンドを使用中に収集されたテーブル統計に基づいています。統計が古いか、ない場合、データベースではクエリを実行するために効率の低いプラン

が選択される可能性があります (特に複雑なクエリの場合)。現在の統計を維持すると、可能な限り短時間で複雑なクエリを実行するうえで役立ちます。

分析

Advisor の分析では、統計が古いか、欠落しているテーブルが追跡されます。複雑なクエリに関連するテーブルアクセスのメタデータが確認されます。複雑なパターンで頻繁にアクセスされるテーブルに統計がない場合、Advisor は ANALYZE を実行するための重要なレコメンデーションを作成します。複雑なパターンで頻繁にアクセスされるテーブルに古い統計がある場合、Advisor は ANALYZE を実行するためのレコメンデーション提案を作成します。

レコメンデーション

テーブルの内容が大きく変わるたびに、ANALYZE で統計を更新します。COPY または INSERT コマンドで、既存のテーブルにかなりの数の新しいデータ行をロードするたびに、ANALYZE を実行することをお勧めします。また、UPDATE または DELETE コマンドを使用してかなりの数の行を変更するたびに、ANALYZE を実行することをお勧めします。統計がない、または統計が古いテーブルを識別するには、スーパーユーザーとして次の SQL コマンドを実行します。結果は、最も大きなテーブルから最も小さいテーブルの順にソートされます。

統計がない、または統計が古いテーブルを識別するには、スーパーユーザーとして次の SQL コマンドを実行します。結果は、最も大きなテーブルから最も小さいテーブルの順にソートされます。

```
SELECT
   ti.schema||'.'||ti."table" tablename,
   ti.size table_size_mb,
   ti.stats_off statistics_accuracy
 FROM svv_table_info ti
 WHERE ti.stats_off > 5.00
 ORDER BY ti.size DESC;
```

実装のヒント

デフォルトの ANALYZE のしきい値は 10 パーセントです。これは、最後の ANALYZE から変更されたテーブルの行が 10 パーセント未満の場合、ANALYZE コマンドは指定されたテーブルをスキップすることを意味します。その結果、各 ETL プロセスの最後に ANALYZE コマンドを発行する選択ができます。この手法では、ANALYZE は頻繁にスキップされますが、必要なときには ANALYZE が確実に実行されます。

ANALYZE の統計は、結合で使用される列 (例: `JOIN tbl_a ON col_b`)、または述語として使用される列 (例: `WHERE col_b = 'xyz'`) に対して最も大きな影響があります。デフォルトでは、ANALYZE は指定されたテーブルのすべての列の統計を収集します。必要に応じて、最も影響が大きい列に対してのみ ANALYZE を実行して、ANALYZE の実行に必要な時間を短縮できます。述語として使用される列を識別するには、次の SQL コマンドを実行できます。また、`ANALYZE PREDICATE COLUMNS` を指定して、分析する列を Amazon Redshift で自動的に選択することもできます。

```
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

詳細については、「テーブルを分析する」を参照してください。

ショートクエリアクセラレーションの有効化

ショートクエリアクセラレーション (SQA) は、実行時間が短い一部のクエリを、実行時間が長いクエリよりも優先します。SQA では実行時間が短いクエリを専用領域で実行します。このため SQA クエリは、実行時間が長いクエリをキューで待機するよう強制されません。SQA は、実行時間が短く、ユーザー定義のキュー内にあるクエリのみを優先します。SQA によって実行時間が短いクエリの実行開始が早くなり、ユーザーへの結果表示も早くなります。

SQA を有効にすると、ショートクエリの実行に割り当てられるワークロード管理 (WLM) キューを減らす、またはなくすことができます。さらに、キュー内のスロットに対する実行時間が長いクエリとショートクエリの競合が不要になるため、WLM キューが使用するクエリスロットの数を少なく設定できます。同時実行数が減るとクエリのスループットが向上し、大部分のワークロードに関するシステム全体のパフォーマンスも向上します。詳細については、「ショートクエリアクセラレーション」を参照してください。

分析

Advisor はワークロードパターンを確認し、SQA がレイテンシーと SQA 対象クエリの日次キュー時間を減らす最近のクエリの数を報告します。

レコメンデーション

WLM 設定を変更して SQA を有効にします。Amazon Redshift は、機械学習アルゴリズムを使用して適格な各クエリを分析します。クエリのパターンを SQA が学習するため、予測精度は向上します。詳細については、「ワークロード管理の設定」を参照してください。

SQA を有効にすると、WLM は、ショートクエリの最大実行時間をデフォルトで動的に設定します。SQA の最大実行時間の動的設定を保持することをお勧めします。

実装のヒント

SQA が有効になっているかどうか確認するには、以下のクエリを実行します。クエリが 1 行を返した場合、SQA は有効です。

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

詳細については、「SQA のモニタリング」を参照してください。

テーブルの分散キーを変更する

Amazon Redshift は、テーブルディストリビューションスタイルに従ってクラスター全体にテーブル行を分散します。キー分散を含むテーブルでは、分散キー (DISTKEY) として列が必要です。テーブル行は、その DISTKEY 列の値に基づいてクラスターのノードスライスに割り当てられます。

適切な DISTKEY によって、各ノードスライスに同様の数の行が配置され、結合条件で頻繁に参照されます。結合は、テーブルが DISTKEY 列で結合されるときに最適化され、その結果クエリのパフォーマンスは向上します。

分析

Advisor は、クラスターのワークロードを分析して、キー分散スタイルから大きなメリットが得られるテーブルに最適な分散キーを識別します。

レコメンデーション

Advisor には、分析に基づいてテーブルの DISTSTYLE および DISTKEY を変更する ALTER TABLE ステートメントが用意されています。パフォーマンスのメリットを大幅に向上させるには、レコメンデーショングループ内の SQL ステートメントをすべて実装する必要があります。

ALTER TABLE を使用して大きなテーブルを再分散すると、クラスターリソースが消費され、一時的なテーブルロックがさまざまな時点で必要になります。他のクラスターのワークロードが軽いときに、レコメンデーションの各グループを実装します。テーブル分散プロパティの最適化の詳細については、Amazon Redshift エンジニアリングの高度なテーブル設計の計画: ディストリビューションスタイルとディストリビューションキーを参照してください。

ALTER DISTSYLE および DISTKEY の詳細については、ALTER TABLE を参照してください。

> **Note**
>
> レコメンデーションが表示されないとしても、必ずしも現在の分散スタイルが最も適切であるとは限りません。Advisor は、十分なデータがない場合や、再分散で期待されるメリットが少ない場合にはレコメンデーションを提供しません。
> また、Advisor のレコメンデーションは特定のテーブルに適用され、必ずしも、同じ名前の列を含むテーブルに適用されるとは限りません。列名を共有するテーブルは、テーブル内のデータが同じでない限り、それらの列に対して異なる特性を含めることができます。
> ETL ジョブによって作成または削除されるステージングテーブルのレコメンデーションが表示される場合は、ETL プロセスを変更して、Advisor が推奨する分散キーを使用します。

テーブルのソートキーを変更する

Amazon Redshift は、テーブルのソートキーに従ってテーブル行をソートします。テーブル行のソートは、ソートキー列の値に基づいています。

テーブルを適切なソートキーでソートすると、ディスクから読み取るテーブルブロックが少なくなるため、特に範囲が制限された述語を持つクエリのパフォーマンスが向上します。

分析

Advisor は数日にわたってクラスターのワークロードを分析し、テーブルにとって有益なソートキーを特定します。

レコメンデーション

Advisor は、分析に基づいてテーブルのソートキーを変更するために、以下の 2 グループの ALTER TABLE ステートメントを提供します。

- COMPOUND ソートキーを追加するために、現在ソートキーを保持していないテーブルを変更するステートメント。
- ソートキーを INTERLEAVED から COMPOUND に変更するか、ソートキーを使用しなくするためのステートメント。

  複合 (COMPOUND) ソートキーを使用すると、メンテナンス時のオーバーヘッドが大幅に削減されます。複合ソートキーがあるテーブルは、インターリーブソートに必要な、高価な VACUUM REINDEX 操作を必要としません。実際には、大半の Amazon Redshift ワークロードでインターリーブソートキーより複合ソートキーの方が効率的です。ただし、テーブルが小さい場合は、ソートキーを格納するためのオーバーヘッドを回避するために、ソートキーを使用しない方が効率的です。

ALTER TABLE を使用して大きなテーブルをソートすると、クラスターリソースが消費され、さまざまなタイミングでテーブルロックが必要になります。クラスターのワークロードが中程度の場合は、各レコメンデーションを実装します。テーブルソートキー設定の最適化の詳細については、[Amazon Redshift エンジニアリングの高度なテーブル設計プレイブック: 複合ソートキーおよびインターリーブソートキー]を参照してください。

ALTER SORTKEY の詳細については、[ALTER TABLE] を参照してください。

> **Note**
>
> テーブルのレコメンデーションが表示されない場合は、必ずしも現在の設定が最適であるとは限りません。Advisor は、十分なデータがない場合や、ソートで期待されるメリットが少ない場合にはレコメンデーションを提供しません。
> Advisor のレコメンデーションは特定のテーブルに適用され、必ずしも、同じ名前やデータ型の列を含むテーブルに適用されるとは限りません。列名を共有するテーブルには、テーブル内のデータとワークロードに基づいて異なるレコメンデーションがある場合があります。

列の圧縮エンコードを変更する

圧縮は、データの保存時にそのサイズを小さくする列レベルの操作です。Amazon Redshift では、ディスクの I/O 量を減らすことによってストレージ領域を節約し、クエリパフォーマンスを向上させるために、圧縮を使用します。各列のデータ型とクエリパターンに基づいて、最適な圧縮エンコードを行うことをお勧めします。最適な圧縮を行えば、クエリをより効率的に実行でき、データベースが占めるストレージ領域を最小限に抑えることができます。

分析

Advisor は、クラスターのワークロードとデータベーススキーマの分析を継続的に実行し、各テーブルの列に最適な圧縮エンコードを特定します。

レコメンデーション

Advisor は、分析に基づいて特定の列の圧縮エンコードを変更する ALTER TABLE ステートメントを提供します。

ALTER TABLE で列圧縮エンコードを変更すると、クラスターのリソースが使用され、さまざまな時点でテーブルロックが必要になります。クラスターのワークロードが軽い場合は、レコメンデーションを実装することをお勧めします。

参考までに、ALTER TABLE の例 は列のエンコードを変更するステートメントをいくつか示します。

> **Note**
>
> Advisor は、十分なデータがない場合や、エンコードを変更することで期待されるメリットが少ない場合にはレコメンデーションを提供しません。

データ型のレコメンデーション

Amazon Redshift には、さまざまなユースケースに対応する SQL データ型のライブラリがあります。これらのデータ型には、`INT` のような整数型や、`VARCHAR` のような文字を保存する型などがあります。Redshift は、迅速なアクセスと優れたクエリパフォーマンスを実現するために、最適化された方法でデータ型を保存します。Redshift は特定のデータ型に対する関数も提供し、これらはクエリ結果のフォーマットや計算に使用できます。

分析

Advisor は、クラスターのワークロードとデータベーススキーマの分析を継続的に実行して、データ型の変更によって大きなメリットが得られる列を特定します。

レコメンデーション

Advisor は、提案されたデータ型を使用した新しい列を追加する `ALTER TABLE` ステートメントを提供します。それに付随する `UPDATE` ステートメントは、既存の列のデータを新しい列にコピーします。新しい列を作成してデータをロードしたら、新しい列にアクセスするようにクエリと取り込みスクリプトを変更します。次に、新しいデータ型に特化した機能と関数を活用します。これらは「SQL 関数リファレンス」に記載されています。

既存のデータを新しい列にコピーするには、時間がかかる場合があります。Advisor のレコメンデーションは、それぞれクラスターのワークロードが軽いときに実装することをお勧めします。「データ型」で、利用可能なデータ型のリストを参照してください。

Advisor は、十分なデータがない場合、またはデータ型を変更することで期待されるメリットが少ない場合にはレコメンデーションを提供しません。

Amazon Redshift のチュートリアル

これらのチュートリアルのステップに従って、Amazon Redshift の特徴について学習します。

チュートリアル: Amazon S3 からデータをロードする	このチュートリアルでは、S3 バケット内のデータファイルから Amazon Redshift データベースのテーブルに、データを最初から最後までロードする手順を説明します。
チュートリアル: Amazon Redshift Spectrum を使用したネストデータのクエリ	このチュートリアルでは、Redshift Spectrum を使用して、ネストデータにクエリを実行します。Redshift Spectrum では、ファイル形式が Parquet、ORC、JSON、Ion のデータにクエリを実行できます。
チュートリアル: 手動ワークロード管理 (WLM) キューの設定	このチュートリアルでは、手動ワークロード管理 (WLM) キューを使用するように Amazon Redshift を設定します。Amazon Redshift は、同時実行クエリを実行するためにリソースの分割方法を管理する場合は、WLM キューを使用します。複数の WLM WLM キューを使用する必要がある場合は、手動 WLM を使用するように Amazon Redshift を設定する必要があります。
チュートリアル: Amazon Redshift での空間 SQL 関数の使用	このチュートリアルでは、空間関数を使用してデータをクエリします。空間関数を使用して、ジオメトリとジオグラフィデータをクエリします。
Amazon Redshift ML のチュートリアル	これらのチュートリアルでは、機械学習モデルを作成して使用します。
チュートリアル: RBAC でのロールの作成とクエリ	このチュートリアルでは、ロールベースのアクセスコントロール (RBAC) を使用して、作成するデータベースでアクセス許可を作成して使用します。RBAC では、読み取り専用アクセス

	許可や読み取り/書き込みアクセス許可など、特定のアクセス許可を持つロールを作成できます。次に、これらのロールをユーザーに割り当てて、指定されたアクセス許可を付与することができます。

自動テーブル最適化

自動テーブル最適化は、管理者の介入を必要とせずに、ソートキーとディストリビューションキーを適用することにより、テーブルの設計を自動的に最適化するセルフチューニング機能です。自動化を使用してテーブルの設計を調整することで、テーブルの最適化を手動で調整して実装するために時間を費やすことなく、より簡単に開始し、最速のパフォーマンスをすばやく得ることができます。

自動テーブル最適化は、クエリがテーブルとどのように相互作用するかを継続的に監視します。高度な人工知能メソッドを使用して、ソートキーとディストリビューションキーを選択し、クラスターのワークロードパフォーマンスを最適化します。Amazon Redshift がキーを適用することでクラスターのパフォーマンスが向上すると判断した場合、テーブルは、クラスターが作成されてから数時間以内に自動的に変更され、クエリへの影響は最小限に抑えられます。

この自動化を利用するために、Amazon Redshift 管理者が新しいテーブルを作成するか、既存のテーブルを変更して自動最適化を使用できるようにします。ディストリビューションスタイルまたはソートキーがある AUTO の既存のテーブルでは、すでに自動化が有効になっています。これらのテーブルに対してクエリを実行すると、Amazon Redshift はソートキーまたはディストリビューションキーによってパフォーマンスが向上するかどうかを判断します。その場合、Amazon Redshift は管理者の介入を必要とせずにテーブルを自動的に変更します。最小数のクエリが実行されると、クラスターの起動から数時間以内に最適化が適用されます。

Amazon Redshift が、ディストリビューションキーによってクエリのパフォーマンスが向上すると判断した場合、ディストリビューションスタイルが AUTO のテーブルでは、ディストリビューションスタイルを KEY に変更できます。

トピック

- 自動テーブル最適化の有効化、無効化、モニタリング
- 保存データのサイズを削減するための列圧縮
- クエリ最適化のためのデータのディストリビューション
- ソートキー
- テーブルの制約

自動テーブル最適化の有効化、無効化、モニタリング

デフォルトでは、ソートキーまたはディストリビューションキーを明示的に定義せずに作成されたテーブルは AUTO に設定されています。テーブルの作成時に、ソートキーまたはディストリビュー

ションキーを手動で明示的に設定することもできます。ソートキーまたはディストリビューションキーを設定した場合、テーブルは自動的に管理されません。

自動テーブル最適化の有効化

既存のテーブルを自動的に最適化できるようにするには、ALTER ステートメントオプションを使用して、テーブルを AUTO に変更します。ソートキーには自動化を定義できますが、ディストリビューションキーには自動化を定義することができません (逆も同様)。ALTER ステートメントを実行してテーブルを自動テーブルに変換する場合、既存のソートキーとディストリビューションスタイルは保持されます。

```
ALTER TABLE table_name ALTER SORTKEY AUTO;
```

```
ALTER TABLE table_name ALTER DISTSTYLE AUTO;
```

詳細については、「ALTER TABLE」を参照してください。

最初は、テーブルにはディストリビューションキーまたはソートキーがありません。ディストリビューションスタイルは、テーブルのサイズに応じて EVEN または ALL のいずれかに設定されます。テーブルのサイズが大きくなるにつれて、Amazon Redshift は最適なディストリビューションキーとソートキーを適用します。最適化は、最小数のクエリが実行されてから数時間以内に適用されます。ソートキーの最適化を決定するとき、Amazon Redshift はテーブルスキャン中にディスクから読み込んだデータブロックを最適化しようとします。ディストリビューションスタイルの最適化を決定するとき、Amazon Redshift はクラスターノード間で転送されるバイト数を最適化しようとします。

テーブルからの自動テーブル最適化の削除

自動最適化からテーブルを削除できます。オートメーションからテーブルを削除するには、ソートキーまたはディストリビューションスタイルを選択します。ディストリビューションスタイルを変更するには、特定のディストリビューションスタイルを指定します。

```
ALTER TABLE table_name ALTER DISTSTYLE EVEN;
```

```
ALTER TABLE table_name ALTER DISTSTYLE ALL;
```

```
ALTER TABLE table_name ALTER DISTSTYLE KEY DISTKEY c1;
```

ソートキーを変更するには、ソートキーを定義するか、[none (なし)] を選択します。

```
ALTER TABLE table_name ALTER SORTKEY(c1, c2);
```

```
ALTER TABLE table_name ALTER SORTKEY NONE;
```

自動テーブル最適化のモニタリング

システムビュー SVV_ALTER_TABLE_RECOMMENDATIONS は、テーブルに対する現在の Amazon Redshift Advisor のレコメンデーションを記録します。このビューには、自動最適化用に定義されているテーブルと定義されていないテーブルのすべてのレコメンデーションが表示されます。

テーブルが自動最適化用に定義されているかどうかを表示するには、システムビュー SVV_TABLE_INFO にクエリを実行します。エントリは、現在のセッションのデータベースに表示されているテーブルに対してのみ表示されます。レコメンデーションは、クラスターの作成時刻から数時間以内に開始され、1 日に 2 回ビューに挿入されます。レコメンデーションが利用可能になると、1 時間以内に開始されます。(Amazon Redshift またはユーザーのいずれかによって) レコメンデーションが適用されると、ビューに表示されなくなります。

システムビュー SVL_AUTO_WORKER_ACTION には、Amazon Redshift によって実行されたすべてのアクションに関する監査ログとテーブルの以前の状態が表示されます。

システムビュー SVV_TABLE_INFO には、システム内のすべてのテーブルが、テーブルのソートキーとディストリビューションスタイルが AUTO に設定されているかどうかを示す列とともに一覧表示されます。

これらのシステムビューの詳細については、[システムモニタリング (プロビジョニングのみ)] を参照してください。

保存データのサイズを削減するための列圧縮

圧縮は、データの保存時にそのサイズを小さくする列レベルのオペレーションです。圧縮によってストレージスペースが節約され、ストレージから読み込まれるデータのサイズが小さくなり、ディスク I/O の量が減少するので、クエリパフォーマンスが向上します。

ENCODE AUTO は、テーブルのデフォルトです。テーブルが ENCODE AUTO に設定されると、Amazon Redshift は、テーブル内のすべての列の圧縮エンコードを自動的に管理します。詳細については、「CREATE TABLE」および「ALTER TABLE」を参照してください。

ただし、テーブル内のいずれかの列に圧縮エンコードを指定すると、テーブルは ENCODE AUTO に設定されなくなります。Amazon Redshift は、テーブルにあるすべての列の圧縮エンコードを自動的に管理しないようになりました。

テーブルを作成するときに、テーブルの列に圧縮タイプまたはエンコードを手動で適用できます。または、COPY コマンドを使用すると、自動的に圧縮を分析および適用できます。詳細については、「COPY による圧縮エンコードの選択」を参照してください。自動圧縮の適用の詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

> **Note**
>
> COPY コマンドを使用して自動圧縮を適用することを強くお勧めします。

新しいテーブルが別のテーブルと同じデータ特性を共有している場合は、圧縮エンコードを手動で適用することを選択できます。または、自動圧縮時に適用される圧縮エンコードがデータにとって最適ではないことがテストで判明した場合は、ユーザーが管理することもできます。圧縮エンコードを手動で適用する場合は、すでに入力されているテーブルに対して ANALYZE COMPRESSION コマンドを再実行し、その結果を使用して圧縮エンコードを選択することができます。

圧縮を手動で適用するには、CREATE TABLE ステートメントの一部として個々の列に対して圧縮エンコードを指定します。構文は次のとおりです。

```
CREATE TABLE table_name (column_name
data_type ENCODE encoding-type)[, ...]
```

ここで、encoding-type には次のセクションのキーワード表から選択します。

例えば、次のステートメントは 2 列のテーブル PRODUCT を作成します。データがテーブルにロードされるときに、PRODUCT_ID 列は圧縮されませんが、PRODUCT_NAME 列はバイトディクショナリエンコード (BYTEDICT) を使用して圧縮されます。

```
create table product(
product_id int encode raw,
product_name char(20) encode bytedict);
```

ALTER TABLE コマンドを使用して列をテーブルに追加する際には、列のエンコードを指定できます。

```
ALTER TABLE table-name ADD [ COLUMN ] column_name column_type ENCODE encoding-type
```

トピック

- 圧縮エンコード
- 圧縮エンコードのテスト

圧縮エンコード

圧縮エンコードは、行がテーブルに追加されるときにデータ値の列に適用される圧縮のタイプを指定します。

ENCODE AUTO は、テーブルのデフォルトです。テーブルが ENCODE AUTO に設定されると、Amazon Redshift は、テーブル内のすべての列の圧縮エンコードを自動的に管理します。詳細については、「CREATE TABLE」および「ALTER TABLE」を参照してください。

ただし、テーブル内のいずれかの列に圧縮エンコードを指定すると、テーブルは ENCODE AUTO に設定されなくなります。Amazon Redshift は、テーブルにあるすべての列の圧縮エンコードを自動的に管理しないようになりました。

CREATE TABLE を使用すると、テーブル内の列の圧縮エンコードを指定するとき、ENCODE AUTO は無効です。ENCODE AUTO が無効なとき、Amazon Redshift は、ユーザーが ENCODE タイプを指定していない列について、次のように圧縮エンコードを自動的に割り当てます。

- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、または DOUBLE PRECISION データ型として定義されている列には、RAW 圧縮が割り当てられます。
- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP、または TIMESTAMPTZ データ型として定義された列には AZ64 圧縮が割り当てられます。
- CHAR または VARCHAR データ型として定義された列には、LZO 圧縮が割り当てられます。

テーブルのエンコードは、作成後に ALTER TABLE を使用して変更できます。ALTER TABLE を使用して ENCODE AUTO を無効にした場合、Amazon Redshift は列の圧縮エンコードを自動的に管理しなくなります。すべての列の圧縮エンコードタイプは、ユーザーが変更するか、ENCODE AUTO を再び有効にするまで、ENCODE AUTO を無効にしたときの圧縮エンコードタイプのままです。

Amazon Redshift は、以下の圧縮エンコーディングをサポートしています。

Raw

Raw エンコードは、ソートキー、および BOOLEAN、REAL、または DOUBLE PRECISION データ型として定義された列として指定された列のエンコードのデフォルトです。raw エンコードでは、データは非圧縮の raw 形式で格納されます。

AZ64

AZ64 は、高い圧縮率とクエリ処理能力の改善を実現するために Amazon によって設計された独自の圧縮エンコードアルゴリズムです。Z64 アルゴリズムは、より小さなデータ値のグループを圧縮し、並列処理に SIMD (Single Instruction Multiple Data) 命令を使用します。AZ64 を使用すると、数値、日付、および時刻データ型のストレージを大幅に節約し、高いパフォーマンスを実現できます。

次のデータ型の CREATE TABLE および ALTER TABLE ステートメントを使用して列を定義する場合、AZ64 を圧縮エンコードとして使用できます。

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- DATE
- TIMESTAMP
- TIMESTAMPTZ

Byte-dictionary

バイトディクショナリエンコードでは、ディスク上の列値のブロックごとに、一意の値の個別のディクショナリが作成されます (Amazon Redshift のディスクブロックは 1 MB を占有します。) ディクショナリには、元のデータ値のインデックスとして格納されている最大 256 個の 1 バイト値が含まれます。単一のブロックに 256 個を超える値が格納されている場合は、余分な値が非圧縮の raw 形式でブロックに書き込まれます。ディスクブロックごとに、このプロセスが繰り返されます。

このエンコードは、低カーディナリティ文字列の列に対して非常に効果的です。このエンコードは、列のデータドメインが一意の値 256 個未満である場合に最適です。

BYTEDICT でエンコードされた文字列データ型 (CHAR と VARCHAR) の列の場合、Amazon Redshift は圧縮されたデータを直接処理するベクタースキャンと述語評価を実行します。これら

のスキャンでは、ハードウェア固有の単一命令複数データ (SIMD) 命令が使用されて、並列処理が行われます。これにより、文字列列のスキャンが大幅に加速されます。バイトディクショナリエンコードは、CHAR/VARCHAR 列に長い文字列が含まれる場合に特にスペース効率が高まります。

テーブルに、CHAR(30) データ型の COUNTRY 列があるとします。データがロードされると、Amazon Redshift はディクショナリを作成し、COUNTRY 列にインデックス値を入力します。ディクショナリには、インデックス作成された一意の値が含まれ、テーブル自体には、対応する値の 1 バイトのサブスクリプトのみが含まれます。

> **Note**
>
> 固定長文字の列には末尾の空白が格納されます。したがって CHAR(30) 列では、バイトディクショナリエンコードを使用すると、圧縮値ごとに 29 バイトのストレージが節約されます。

次の表は、COUNTRY 列のディクショナリを示しています。

一意のデータ値	ディクショナリインデックス	サイズ (固定長、30 バイト/値)
England	0	30
United States of America	1	30
Venezuela	2	30
Sri Lanka	3	30
Argentina	4	30
Japan	5	30
Total		180

次の表は、COUNTRY 列の値を示しています。

元のデータ値	元のサイズ (固定長、30 バイト/値)	圧縮値 (インデックス)	新しいサイズ (バイト)
England	30	0	1
England	30	0	1
United States of America	30	1	1
United States of America	30	1	1
Venezuela	30	2	1
Sri Lanka	30	3	1
Argentina	30	4	1
Japan	30	5	1
Sri Lanka	30	3	1
Argentina	30	4	1
Total	300		10

この例の合計圧縮サイズは、次のように計算されます。ディクショナリに 6 つの異なるエントリが格納され (6 * 30 = 180)、テーブルに 10 個の 1 バイト圧縮値が含まれて、合計 190 バイトとなります。

Delta

デルタエンコードは、日時列にとって非常に有用です。

デルタエンコードは、列内の連続する値間の差を記録することにより、データを圧縮します。この差は、ディスク上の列値の各ブロックに対する個別のディクショナリに記録されます (Amazon Redshift のディスクブロックは 1 MB を占有します。) 例えば、列に 1 から 10 までの 10 個の整数が順番に含まれているとします。最初の値は 4 バイトの整数 (+ 1 バイトのフラグ) として保存

されます。次の 9 つはそれぞれ値 1 のバイトとして保存され、前の値より 1 大きいことを示します。

デルタエンコードには、次の 2 つのバージョンがあります。

- 差を 1 バイト値 (8 ビット整数) として記録する DELTA
- 差を 2 バイト値 (16 ビット整数) として記録する DELTA32K

1 バイトを使用して列内のほとんどの値を圧縮できる場合は、1 バイトの変形が非常に効果的です。しかし、最悪の場合、デルタがより大きいと、このエンコードは非圧縮データを保存する場合よりも多少効果が低くなる可能性もあります。16 ビットバージョンにも同様の論理が当てはまります。

2 つの値間の差が 1 バイトの範囲 (DELTA) または 2 バイトの範囲 (DELTA32K) を超える場合は、先頭に 1 バイトのフラグが付けられて元の完全な値が格納されます。1 バイトの範囲は -127～127、2 バイトの範囲は -32K～32K です。

次の表は、デルタエンコードが数値列に対してどのように機能するかを示しています。

元のデータ値	元のサイズ (バイト)	差 (デルタ)	圧縮値	圧縮サイズ (バイト)
1	4		1	1+4 (フラグ + 実際の値)
5	4	4	4	1
50	4	45	45	1
200	4	150	150	1+4 (フラグ + 実際の値)
185	4	-15	-15	1
220	4	35	35	1
221	4	1	1	1
合計	28			15

LZO

LZO エンコードは、非常に高い圧縮率と良好なパフォーマンスを実現します。LZO エンコードは、非常に長い文字列を格納する CHAR 列および VARCHAR 列に対して使用すると特に効果的です。製品説明、ユーザーコメント、JSON 文字列などの自由形式テキストに適しています。

Mostly

Mostly エンコードは、列のデータ型が、格納された大部分の値で必要なサイズより大きい場合に有用です。このタイプの列に Mostly エンコードを指定して、列内の大部分の値を、より小さい標準ストレージサイズに圧縮することができます。圧縮できない残りの値は、raw 形式で格納されます。例えば、INT2 列などの 16 ビット列を 8 ビットストレージに圧縮できます。

一般的に、Mostly エンコードは次のデータ型に対して使用します。

- SMALLINT/INT2 (16 ビット)
- INTEGER/INT (32 ビット)
- BIGINT/INT8 (64 ビット)
- DECIMAL/NUMERIC (64 ビット)

列のデータ型のサイズに見合う Mostly エンコードの適切なバージョンを選択します。例えば、16 ビット整数列として定義された列に MOSTLY8 を適用します。MOSTLY16 を 16 ビットデータ型の列に適用したり、MOSTLY32 を 32 ビットデータ型の列に適用したりすることはできません。

列内の比較的多くの値を圧縮できない場合、Mostly エンコードは非圧縮の場合よりも効果が低くなります。これらのエンコードの 1 つを列に適用する前に、確認してください。現在ロードしようとしている (そして今後ロードする可能性が高い) 値のほとんどは、次のテーブルに示す範囲に収まるはずです。

エンコード	圧縮ストレージサイズ	圧縮できる値の範囲 (範囲外の値は raw として格納される)
MOSTLY8	1 バイト (8 ビット)	-128 ~ 127
MOSTLY16	2 バイト (16 ビット)	-32768 ~ 32767
MOSTLY32	4 バイト (32 ビット)	-2147483648 ~ +2147483647

> **Note**
>
> 10 進値では、値が範囲内にあるかどうかを判断する場合に小数点を無視します。例えば、1,234.56 は 123,456 として扱われ、MOSTLY32 列で圧縮できます。

例えば、VENUE テーブルの VENUEID 列が raw 整数列として定義されている場合は、その値が 4 バイトのストレージを消費することを意味します。しかし、列の値の現在の範囲は **0**～**309** です。したがって、VENUEID に対して MOSTLY16 エンコードを使用してこのテーブルを再作成および再ロードすると、その列の各値のストレージが 2 バイトに減少します。

別のテーブルで参照される VENUEID 値のほとんどが 0～127 の範囲内にある場合、その外部キー列を MOSTLY8 としてエンコードすることは妥当と言えます。選択を行う前に、参照テーブルデータに対していくつかのクエリを実行して、ほとんどの値が 8 ビット、16 ビット、または 32 ビットの範囲内にあるかどうかを確認する必要があります。

次の表は、MOSTLY8、MOSTLY16、および MOSTLY32 エンコードが使用される場合の特定の数値の圧縮サイズを示しています。

元の値	元の INT または BIGINT (バイト)	MOSTLY8 圧縮サイズ (バイト)	MOSTLY16 圧縮サイズ (バイト)	MOSTLY32 圧縮サイズ (バイト)
1	4	1	2	4
10	4	1	2	4
100	4	1	2	4
1000	4	raw データサイズと同じ	2	4
10000	4		2	4
20000	4		2	4
40000	8		raw データサイズと同じ	4
100000	8			4
2000000000	8			4

Run length

ランレングスエンコードは、連続して繰り返される値を、値と連続発生数 (実行の長さ) から成るトークンに置き換えます。ディスク上の列値のブロックごとに、一意の値の個別のディクショナリが作成されます (Amazon Redshift のディスクブロックは 1 MB を占有します。) このエンコードは、データ値が連続して頻繁に繰り返されるテーブル (例えば、テーブルがこれらの値でソートされる場合) に最も適しています。

例えば、大きなディメンションテーブルの列に、予測どおりに小さなドメイン (10 個未満の可能な値を持つ COLOR 列など) があるとします。これらの値は、データがソートされていない場合でも、テーブル全体で長いシーケンスに分類される可能性があります。

ソートキーとして指定された列に、ランレングスエンコードを適用することは推奨されません。範囲が制限されたスキャンは、ブロックに同様の数の行が含まれる場合にパフォーマンスが向上します。ソートキー列が、同じクエリ内の他の列よりもかなり高度に圧縮される場合、範囲が制限されたスキャンはパフォーマンスが低下する可能性があります。

次の表は、COLOR 列の例を使用して、ランレングスエンコードがどのように機能するかを示しています。

元のデータ値	元のサイズ (バイト)	圧縮値 (トークン)	圧縮サイズ (バイト)
Blue	4	{2,Blue}	5
Blue	4		0
Green	5	{3,Green}	6
Green	5		0
Green	5		0
Blue	4	{1,Blue}	5
Yellow	6	{4,Yellow}	7
Yellow	6		0
Yellow	6		0
Yellow	6		0

元のデータ値	元のサイズ (バイト)	圧縮値 (トークン)	圧縮サイズ (バイト)
合計	51		23

Text255 and Text32k

text255 および text32k エンコードは、同じ単語が頻繁に出現する VARCHAR 列を圧縮する場合に有用です。ディスク上の列値のブロックごとに、一意の単語の個別のディクショナリが作成されます (Amazon Redshift のディスクブロックは 1 MB を占有します。) ディクショナリには、列内の最初の 245 個の一意の単語が含まれます。これらの単語は、ディスク上で、245 個の値の 1 つを表す 1 バイトインデックス値に置き換えられ、ディクショナリに表されていないすべての単語は非圧縮で格納されます。このプロセスは、1 MB のディスクブロックごとに繰り返されます。インデックス作成された単語が列内に頻繁に出現する場合、列の圧縮率は非常に高くなります。

text32k エンコードでも原理は同じですが、各ブロックのディクショナリは特定数の単語をキャプチャすることはありません。代わりにディクショナリは、結合されたエントリが、32K から多少のオーバーヘッドを減算した長さに達するまで、検出した一意の各単語のインデックスを作成します。インデックス値は 2 バイトで格納されます。

例えば、VENUE テーブルの VENUENAME 列を考えてみます。この列には **Arena**、**Center**、**Theatre** などの単語が繰り返し出現し、text255 圧縮が適用された場合、各ブロックに出現する最初の 245 個の単語内にこれらが含まれると考えられます。その場合、この列は圧縮の利点を利用できます。その場合、これらの単語が出現するたびに 1 バイトのストレージ (それぞれ 5、6、または 7 バイトではなく) を占めるにすぎないのが理由です。

ZSTD

Zstandard (ZSTD) エンコードは、多様なデータセット間で非常にパフォーマンスのいい高圧縮比率を提供します。ZSTD は、製品説明、ユーザーのコメント、ログ、JSON 文字列など、長さがさまざまな文字列を保存する CHAR および VARCHAR 列に対して、特に効果を発揮します。一部のアルゴリズム (デルタエンコードや Mostly エンコードなど) では非圧縮時よりもストレージスペースの使用量が増える場合がありますが、ZSTD ではディスク使用量が増えることはありません。

ZSTD は、SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、および TIMESTAMPTZ データ型をサポートします。

次の表は、サポートされる圧縮エンコード、およびエンコードをサポートするデータ型を示しています。

エンコードタイプ	CREATE TABLE および ALTER TABLE のキーワード	データ型
raw (非圧縮)	RAW	すべて
AZ64	AZ64	SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP、TIMESTAMPTZ
バイトディクショナリ	BYTEDICT	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ
デルタ	DELTA DELTA32K	SMALLINT、INT、BIGINT、DATE、TIMESTAMP、DECIMAL INT、BIGINT、DATE、TIMESTAMP、DECIMAL
LZO	LZO	SMALLINT、INTEGER、BIGINT、DECIMAL、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ、SUPER
Mostlyn	MOSTLY8 MOSTLY16 MOSTLY32	SMALLINT、INT、BIGINT、DECIMAL INT、BIGINT、DECIMAL BIGINT、DECIMAL

エンコードタイプ	CREATE TABLE および ALTER TABLE のキーワード	データ型
ランレングス	RUNLENGTH	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ
テキスト	TEXT255 TEXT32K	VARCHAR のみ VARCHAR のみ
Zstandard	ZSTD	SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、TIMESTAMPTZ、SUPER

圧縮エンコードのテスト

列のエンコードを手動で指定する場合は、データに対してさまざまなエンコードをテストできます。

> **Note**
>
> できる限り COPY コマンドを使用してデータをロードし、データに基づいて最適なエンコードを COPY コマンドが選択できるようにすることをお勧めします。または、ANALYZE COMPRESSION コマンドを使用して、既存のデータに対して提案されるエンコードを表示できます。自動圧縮の適用の詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

データ圧縮の有意義なテストを実行するには、多数の行が必要になります。この例では、2 つのテーブル VENUE と LISTING から選択を行うステートメントを使用して、テーブルを作成し行を挿入

します。通常は 2 つのテーブルを結合する WHERE 句は省略します。その結果、VENUE テーブルの各行が LISTING テーブルのすべての行に結合されて、合計 3200 万以上の行になります。これはデカルト結合と呼ばれ、通常はお勧めしません。ただし、この目的においては、多数の行を作成する便利な方法です。テストするデータを含む既存のテーブルがある場合は、このステップをスキップできます。

サンプルデータを含むテーブルを作成したら、7 列のテーブルを作成します。それぞれ異なる圧縮エンコード (raw、bytedict、lzo、run length、text255、text32k、および zstd) が適用されます。最初のテーブルからデータを選択する INSERT コマンドを実行することにより、各列に全く同じデータを入力します。

圧縮エンコードをテストするには、以下を実行します。

1. (オプション) 最初に、デカルト結合を使用して、多数の行を含むテーブルを作成します。既存のテーブルをテストする場合は、このステップをスキップします。

```
create table cartesian_venue(
venueid smallint not null distkey sortkey,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer);

insert into cartesian_venue
select venueid, venuename, venuecity, venuestate, venueseats
from venue, listing;
```

2. 次に、比較する各エンコードを含むテーブルを作成します。

```
create table encodingvenue (
venueraw varchar(100) encode raw,
venuebytedict varchar(100) encode bytedict,
venuelzo varchar(100) encode lzo,
venuerunlength varchar(100) encode runlength,
venuetext255 varchar(100) encode text255,
venuetext32k varchar(100) encode text32k,
venuezstd varchar(100) encode zstd);
```

3. INSERT ステートメントを SELECT 句とともに使用して、すべての列に同じデータを挿入します。

```
insert into encodingvenue
```

```
select venuename as venueraw, venuename as venuebytedict, venuename as venuelzo,
 venuename as venuerunlength, venuename as  venuetext32k, venuename as  venuetext255,
 venuename as venuezstd
from cartesian_venue;
```

4. 新しいテーブルの行数を確認します。

```
select count(*) from encodingvenue

  count
----------
 38884394
(1 row)
```

5. STV_BLOCKLIST システムテーブルをクエリ処理して、各列で使用される 1 MB のディスクブロックの数と比較します。

   MAX 集計関数が、各列のブロックの最高数を返します。STV_BLOCKLIST テーブルには、3 つのシステム生成列の詳細が含まれます。この例では、WHERE 句で `col < 6` を使用して、システム生成列を除外します。

```
select col, max(blocknum)
from stv_blocklist b, stv_tbl_perm p
where (b.tbl=p.id) and name ='encodingvenue'
and col < 7
group by name, col
order by col;
```

   クエリは次の結果を返します。列には 0 から始まる番号が付けられています。クラスターの設定方法によっては結果の数が異なる場合がありますが、相対的なサイズは同様のものになります。このデータセットについては、2 列目の BYTEDICT エンコードから、最良の結果が得られました。このアプローチの圧縮比は 20:1 よりも優れています。LZO および ZSTD エンコードからも良好な結果が得られました。もちろん、異なるデータセットからは異なる結果が得られます。より長い文字列が列に含まれる場合は、LZO から最良の圧縮結果が得られることが多くなります。

```
 col | max
-----+-----
   0 | 203
   1 |  10
   2 |  22
   3 | 204
```

```
    4 |  56
    5 |  72
    6 |  20
(7 rows)
```

既存のテーブルにデータが存在する場合は、ANALYZE COMPRESSION コマンドを使用して、テーブルに対して提案されるエンコードを表示できます。例えば、次の例は、3800 万行を含む VENUE テーブル CARTESIAN_VENUE のコピーに対して推奨されるエンコードを示しています。ANALYZE COMPRESSION により VENUENAME 列には LZO エンコードが推奨されています。ANALYZE COMPRESSION は減少率を含む複数の要素に基づいて最適な圧縮を選択します。この特定のケースでは、BYTEDICT の方がより圧縮できますが、LZO でも 90 パーセントを超える圧縮になります。

```
analyze compression cartesian_venue;

Table          | Column     | Encoding | Est_reduction_pct
---------------+------------+----------+------------------
reallybigvenue | venueid    | lzo      | 97.54
reallybigvenue | venuename  | lzo      | 91.71
reallybigvenue | venuecity  | lzo      | 96.01
reallybigvenue | venuestate | lzo      | 97.68
reallybigvenue | venueseats | lzo      | 98.21
```

例

次の例は、さまざまなデータ型の列を含む CUSTOMER テーブルを作成します。この CREATE TABLE ステートメントは、これらの列に対する圧縮エンコードの数ある可能な組み合わせの 1 つを示しています。

```
create table customer(
custkey int encode delta,
custname varchar(30) encode raw,
gender varchar(7) encode text255,
address varchar(200) encode text255,
city varchar(30) encode text255,
state char(2) encode raw,
zipcode char(5) encode bytedict,
start_date date encode delta32k);
```

次の表は、CUSTOMER テーブルに対して選択された列エンコードを示し、それぞれの選択内容について説明しています。

列	データ型	エンコード	説明
CUSTKEY	int	delta	CUSTKEY は、連続した一意の整数値から成ります。差は 1 バイトとなるので、DELTA を選択するのが適切です。
CUSTNAME	varCHAR(30)	raw	CUSTNAME には、繰り返し値がほとんどない大きなドメインがあります。いずれの圧縮エンコードも効果的でないと考えられます。
GENDER	varchar(7)	text255	GENDER は、多数の繰り返し値を含む非常に小さなドメインです。同じ単語が繰り返し出現する VARCHAR 列には、text255 が適しています。
ADDRESS	varchar(200)	text255	ADDRESS は大きなドメインですが、Street、Avenue、North、South など、繰り返しの単語が多数含まれます。同じ単語が繰り返し出現する VARCHAR 列を圧縮するには、text255 と text32k が

列	データ型	エンコード	説明
			有用です。列が短いので、text255 を選択するのが適切です。
CITY	varCHAR(30)	text255	CITY は、いくつかの繰り返し値を含む大きなドメインです。特定の市の名前が、他の市の名前よりもかなり頻繁に使用されます。ADDRESS と同じ理由により、text255 を選択するのが適切です。
STATE	char(2)	raw	米国では、STATE は 50 個の 2 文字値の正確なドメインです。bytedict エンコードによって多少圧縮されるものの、列サイズが 2 文字のみであるため、データの解凍時のオーバーヘッドを考慮すると圧縮する必要はそれほどないと考えられます。

列	データ型	エンコード	説明
ZIPCODE	char(5)	bytedict	ZIPCODE は、50,000 個未満の一意の値を含む既知のドメインです。特定の郵便番号が、他の郵便番号よりもかなり頻繁に出現します。bytedict エンコードは、数に制限のある一意の値が列に含まれる場合に非常に効果的です。
START_DATE	date	delta32k	デルタエンコードは、特に行が日付順にロードされる場合に、日時列にとって非常に有用です。

クエリ最適化のためのデータのディストリビューション

テーブルにデータをロードすると、そのテーブルの分散スタイルに従って、Amazon Redshift がテーブルの行を各コンピューティングノードに分散します。クエリを実行すると、必要に応じて結合と集計を実行するために、クエリオプティマイザによって行がコンピューティングノードに再分散されます。テーブル分散スタイルの選択は、クエリを実行する前にデータを必要な場所に配置しておくことによって、再分散ステップの影響を最小限に抑えるために行われます。

> **Note**
>
> このセクションでは、Amazon Redshift データベースにおけるデータディストリビューションの原則について説明します。`DISTSTYLE AUTO` を使用してテーブルを作成することをお勧めします。その場合、Amazon Redshift は自動テーブル最適化を使用してデータディストリビューションスタイルを選択します。詳細については、「[自動テーブル最適化]」を参照し

てください。このセクションの残りの部分では、ディストリビューションスタイルについて詳しく説明します。

トピック

データ分散の概念

以下に、Amazon Redshift のデータディストリビューションの概念をいくつか示します。

ノードとスライス

Amazon Redshift クラスターは一連のノードです。クラスター内の各ノードには、独自のオペレーティングシステム、専用メモリ、および専用のディスクストレージが備わっています。ノードの 1 つはリーダーノードであり、コンピューティングノードへのデータの分散およびクエリ処理タスクを管理します。コンピューティングノードは、これらのタスクを実行するためのリソースを提供します。

コンピューティングノードのディスクストレージは複数のスライスに分割されています。ノードあたりのスライスの数は、クラスターのノードサイズによって決まります。ノードはいずれも並列クエリの実行に関与し、スライス全体でできるだけ均等に分散されたデータを処理します。各ノードサイズに含まれるスライスの数の詳細については、「Amazon Redshift 管理ガイド」の「クラスターおよびノードについて」を参照してください。

データの再分散

テーブルにデータをロードすると、そのテーブルの分散スタイルに従って、Amazon Redshift がテーブルの行を各ノードスライスに分散します。クエリプランの一環として、クエリオプティマイザは、

クエリの実行を最適化するために必要なデータブロックの配置場所を決定します。クエリの実行中にデータが物理的に移動または再配信されます。再分散では、結合する特定の行を送信するか、またはテーブル全体をすべてのノードにブロードキャストすることが必要となる場合があります。

データの再分散は、クエリプランコストの相当な部分を占める可能性があり、また、再分散によって生成されるネットワークトラフィックは、他のデータベース処理に影響を及ぼし、システム全体のパフォーマンスを低速化する可能性があります。データを最初にどこに配置すれば最も効果的かを予測すると、データ再分散の影響を最小限に抑えることができます。

データ分散の目標

テーブルにデータをロードすると、そのテーブルを作成したときに選択した分散スタイルに従って、Amazon Redshift がテーブルの行をコンピューティングノードとスライスに分散します。データ分散では、次の 2 つの主要目標が設定されます。

- クラスター内のノード間でワークロードを均等に分散させること。不均等な分散が行われると (データ分散スキュー)、一部のノードの作業量が他のノードよりも多くなり、クエリのパフォーマンスが低下します。
- クエリの実行中にデータ移動を最小限に抑えるため。結合または集計に関与する行が、他のテーブルの結合列とともにノード上ですでにコロケーションされている場合、クエリ実行中に再分散する必要のあるデータの量はそれほど多くなりません。

データベースにどの分散戦略を選択するかは、クエリのパフォーマンス、ストレージ要件、データロード、およびメンテナンスに重大な影響を及ぼします。各テーブルに最良の分散スタイルを選択することにより、データを均等に分散し、システム全体のパフォーマンスを大幅に向上させることができます。

分散スタイル

テーブルを作成する場合は、以下の AUTO、EVEN、KEY、または ALL という分散スタイルのいずれかを指定します。

分散スタイルを指定しない場合、Amazon Redshift は AUTO 分散を使用します。

AUTO 分散

AUTO 分散では、Amazon Redshift はテーブルデータのサイズに基づいて最適な分散スタイルを割り当てます。例えば、AUTO 分散スタイルが指定された場合、Amazon Redshift ではまず、ALL 分散スタイルを小さなテーブルに割り当てます。テーブルが大きくなると、Amazon Redshift は分散スタイ

ルを KEY に変更し、プライマリキー (または複合プライマリキーの列) を分散キーとして選択する場合があります。テーブルが大きくなり、分散キーに適した列がない場合、Amazon Redshift は分散スタイルを EVEN に変更します。ディストリビューションスタイルの変更は、ユーザークエリへの影響を最小限に抑え、バックグラウンドで発生します。

テーブルのディストリビューションキーを変更するために Amazon Redshift が自動的に実行したアクションを表示するには、SVL_AUTO_WORKER_ACTION を参照してください。テーブルのディストリビューションキーの変更に関する現在の推奨事項を表示するには、SVV_ALTER_TABLE_RECOMMENDATIONS を参照してください。

テーブルに適用された分散スタイルを表示するには、PG_CLASS_INFO システムカタログビューに対してクエリを実行します。詳細については、「分散スタイルの表示」を参照してください。CREATE TABLE ステートメントで分散スタイルを指定しない場合、Amazon Redshift は AUTO 分散を適用します。

EVEN 分散

リーダーノードは、特定の列の値に含まれている値にかかわらず、ラウンドロビン方式によって複数のスライス間で行を分散させます。テーブルが結合に参加しない場合は、EVEN ディストリビューションが適切です。KEY ディストリビューションと ALL ディストリビューションのどちらかを明確に選択できない場合にも適しています。

キー分散

行の分散は、特定の列に含まれている値に従って行われます。リーダーノードは、複数の一致する値を同じノードスライスに配置します。結合キーに基づいてテーブルのペアを分散する場合、リーダーノードは、結合列に含まれている値に従って行をスライスにコロケーションします。このようにして、共通の列から一致する値が物理的に一緒に保存されます。

ALL 分散

テーブル全体のコピーがすべてのノードに分散されます。EVEN 分散または KEY 分散によってテーブルの一部の行のみが各ノードに配置されている場合、ALL 分散を行うと、テーブルが関与しているあらゆる結合ですべての行が確実にコロケーションされるようになります。

ALL 分散では、クラスタ内のノードの数だけ必要なストレージが増えるため、データをロードまたは更新したり、複数のテーブルに挿入したりするのに時間がかかります。ALL 分散は、比較的移動の少ないテーブル、つまり、更新頻度が低く、更新範囲が広くないテーブルに適しています。クエリ中に小さなテーブルを再分散するコストは低いため、小さいディメンションテーブルを DISTSTYLE ALL として定義しても大きな利点はありません。

> **Note**
>
> 列のディストリビューションスタイルを指定した後、Amazon Redshift はクラスターレベルでデータ配信を処理します。Amazon Redshift は、データベースオブジェクト内でのデータをパーティション化するという概念を必要とせず、サポートしていません。テーブルスペースを作成したり、テーブルのパーティション化方式を定義したりする必要はありません。

シナリオによっては、テーブルの作成後にそのテーブルの分散スタイルを変更することができます。詳細については、「ALTER TABLE」を参照してください。テーブルの作成後にテーブルの分散スタイルを変更できない場合は、テーブルを再作成して新しいテーブルにディープコピーを作成できます。詳細については、ディープコピーを実行する を参照してください。

分散スタイルの表示

テーブルの分散スタイルを表示するには、PG_CLASS_INFO ビューまたは SVV_TABLE_INFO ビューに対してクエリを実行します。

テーブルの現在の分散スタイルは、PG_CLASS_INFO の RELEFFECTIVEDISTSTYLE 列に示されます。テーブルが自動分散を使用する場合、RELEFFECTIVEDISTSTYLE は 10、11、または 12 です。これは、効率的な分散スタイルが AUTO (ALL)、AUTO (EVEN)、または AUTO (KEY) のどれであるかを示します。テーブルが自動分散を使用する場合、分散スタイルには当初 AUTO (ALL) が表示され、その後テーブルが大きくなると AUTO (EVEN) または AUTO (KEY) に変更される場合があります。

次の表は、RELEFFECTIVEDISTSTYLE 列に含まれる各値の分散スタイルを示しています。

RELEFFECTIVEDISTSTYLE	現在の分散スタイル
0	EVEN
1	KEY
8	ALL
10	AUTO (ALL)
11	AUTO (EVEN)

RELEFFECTIVEDISTSTYLE	現在の分散スタイル
12	AUTO (KEY)

テーブルの現在の分散スタイルは、SVV_TABLE_INFO の DISTSTYLE 列に示されます。テーブルが分散スタイルを使用する場合、DISTSTYLE は AUTO (ALL)、AUTO (EVEN)、または AUTO (KEY) になります。

次の例では、3 つの分散スタイルと自動分散を使用して 4 つのテーブルを作成した後で、分散スタイルを表示するために SVV_TABLE_INFO に対するクエリを実行します。

```
create table public.dist_key (col1 int)
diststyle key distkey (col1);

insert into public.dist_key values (1);

create table public.dist_even (col1 int)
diststyle even;

insert into public.dist_even values (1);

create table public.dist_all (col1 int)
diststyle all;

insert into public.dist_all values (1);

create table public.dist_auto (col1 int);

insert into public.dist_auto values (1);

select "schema", "table", diststyle from SVV_TABLE_INFO
where "table" like 'dist%';

        schema   |    table        | diststyle
     ------------+-----------------+------------
      public     | dist_key        | KEY(col1)
      public     | dist_even       | EVEN
      public     | dist_all        | ALL
      public     | dist_auto       | AUTO(ALL)
```

クエリパターンの評価

分散スタイルの選択は、データベース設計の 1 つの側面にすぎません。分散スタイルをシステム全体のコンテキストの中で検討し、クラスターサイズ、圧縮エンコード方法、ソートキー、テーブル制約など、他の重要な要因と分散の間でバランスを取ることが必要です。

できる限り実際のデータに近いデータを使用してシステムをテストします。

分散スタイルについて適切な選択を行うには、Amazon Redshift アプリケーションのクエリパターンを理解しておく必要があります。システム内で最もコストの大きいクエリを特定し、それらのクエリの要求に基づいてデータベースの初期設計を行います。クエリの総コストを決定する要因には、クエリの実行に要する時間、およびクエリによって使用されるコンピューティングリソースの量などがあります。クエリのコストを決定するその他の要因には、クエリの実行頻度、他のクエリやデータベースのオペレーションに及ぼす影響の度合いなどがあります。

最もコストの大きいクエリによって使用されるテーブルを特定し、クエリランタイムにおけるそれらのテーブルの役割を評価します。テーブルの結合および集計方法を検討します。

このセクションに示すガイドラインに従って、各テーブルの分散スタイルを選択します。それが完了したら、テーブルを作成し、実際のデータにできるだけ近いデータをテーブルにロードします。次に、使用する予定のクエリの種類についてテーブルのテストを行います。クエリの説明プランを評価して、調整の余地を特定できます。ロード時間、ストレージスペース、およびクエリランタイムを比較して、システムの全体的な要件のバランスを取ります。

分散スタイルの指定

このセクションでは、分散スタイルの指定に関する検討事項と推奨事項にスタースキーマを例として使用しています。データベース設計は、スタースキーマに基づいている場合もあれば、スタースキーマの変形または完全に異なるスキーマに基づいている場合もあります。Amazon Redshift は、選択したスキーマデザインで効果的に機能するように設計されています。このセクションで示す原則は、どの設計スキーマにも適用できます。

1. すべてのテーブルのプライマリキーと外部キーを指定します。

   プライマリキーおよび外部キーの制約は、Amazon Redshift によって強制されることはありませんが、クエリプランの生成時にクエリオプティマイザによって使用されます。プライマリキーと外部キーを設定する場合は、アプリケーションがキーの有効性を維持する必要があります。

2. ファクトテーブルとその最大のディメンションテーブルを共通の列に分散します。

テーブルのサイズだけでなく、最も一般的な結合に関与しているデータセットのサイズにも基づいて、最大のディメンションを選択します。WHERE 句を使用したテーブルのフィルタリングが一般的に行われている場合は、そのテーブルの行の一部しか結合に関与しません。このようなテーブルが再分散に及ぼす影響は、より多くのデータを提供するサイズの小さなテーブルよりも小さくなります。ディメンションテーブルのプライマリキーとそれに対応するファクトテーブルの外部キーをいずれも DISTKEY として指定します。複数のテーブルが同じディストリビューションキーを使用している場合、これらのテーブルはファクトテーブルともコロケーションされます。ファクトテーブルに指定できる分散キーは 1 つだけです。別のキーを使用して結合しているテーブルは、ファクトテーブルとコロケーションされません。

3. 他のディメンションテーブルの分散キーを指定します。

   他のテーブルとの結合に最も一般的に使用される方法に応じて、プライマリキーまたは外部キーを使用してテーブルを分散させます。

4. 一部のディメンションテーブルで ALL 分散を使用するよう変更すべきかどうかを評価します。

   ファクトテーブルやその他の重要な結合テーブルとディメンションテーブルをコロケーションできない場合は、テーブル全体を全ノードに分散させることによってパフォーマンスを大幅に向上させることができます。ALL 分散を使用すると、ストレージ領域要件の増大、ロード時間の長期化、メンテナンス操作の増加を招くため、ALL 分散を選択する前にすべての要因を比較検討しておく必要があります。次のセクションでは、EXPLAIN プランを評価することによって ALL 分散の適用候補を特定する方法について説明します。

5. 残りのテーブルに AUTO 分散を使用します。

   テーブルがほとんど非正規化され、結合に関与していない場合や、別のディストリビューションスタイルとして何を選択すべきかが明らかでない場合は、AUTO 分散を使用します。

Amazon Redshift が適切な分散スタイルを選択できるようにするには、分散スタイルを明示的に指定しないでください。

クエリプランの評価

クエリプランを使用すると、分散スタイル最適化の候補を特定できます。

初期設計の決定を行った後で、テーブルを作成し、テーブルにデータをロードして、テーブルをテストします。実際のデータにできるだけ近いテストデータセットを使用します。比較のベースラインとして使用するロード時間を測定します。

実行する予定のクエリのうち、典型的な高コストクエリ、つまり結合と集計を使用するクエリを評価します。さまざまな設計オプションの実行時間を比較します。実行時間を比較する際は、最初のランタイムにはコンパイル時間が含まれるため、最初の時間はカウントしないでください。

DS_DIST_NONE

対応するスライスはコンピューティングノードにコロケーションされているため、再分散は必要となりません。通常は、DS_DIST_NONE ステップ (ファクトテーブルと単一のディメンションテーブル間の結合) が 1 つあるだけです。

DS_DIST_ALL_NONE

内部結合テーブルで DISTSTYLE ALL が使用されているため、再分散は必要となりません。テーブル全体が各ノードに配置されます。

DS_DIST_INNER

内部テーブルが再分散されます。

DS_DIST_OUTER

外部テーブルが再分散されます。

DS_BCAST_INNER

内部テーブル全体のコピーがすべてのコンピューティングノードにブロードキャストされます。

DS_DIST_ALL_INNER

外部テーブルで DISTSTYLE ALL が使用されるため、内部テーブル全体が単一スライスに再分散されます。

DS_DIST_BOTH

両方のテーブルが再分散されます。

DS_DIST_NONE と DS_DIST_ALL_NONE が好適です。この 2 つは、すべての結合がコロケーションされているため、そのステップで分散が必要とならなかったことを示します。

DS_DIST_INNER は、内部テーブルがノードに再分散されているため、ステップのコストが比較的高くなる可能性があることを意味します。DS_DIST_INNER は、外部テーブルが結合キーを使用してすでに正しく分散されていることを示します。これを DS_DIST_NONE に変換するには、内部テーブルの分散キーを結合キーに設定します。場合によっては、外部テーブルが結合キーを使用して配信されていないため、結合キーを使用して内部テーブルを配信できないことがあります。この場合、内部テーブルに ALL ディストリビューションを使用するかどうかを評価します。テーブルの

更新頻度が低いか、更新範囲が広くない場合で、高い再ディストリビューションコストに対応できるだけの十分なテーブルサイズの場合は、ディストリビューションスタイルを ALL に変更してから、再びテストします。ALL 分散はロード時間の長期化を招くため、再テストする場合は、評価要因にロード時間を含めてください。

DS_DIST_ALL_INNER は好適ではありません。これは、外部テーブルで DISTSTYLE ALL が使用されているため、外部テーブル全体のコピーが各ノードに配置されるよう、内部テーブル全体が単一スライスに再分散されることを意味します。その結果、全ノードを使用した並列ランタイムのメリットを生かせず、単一ノードで結合が直列ランタイムになるという非効率が生じます。DISTSTYLE ALL は、内部結合テーブルでのみ使用するよう設計されています。代わりに、外部テーブルの分散キーを指定するか、EVEN 分散を使用してください。

DS_BCAST_INNER と DS_DIST_BOTH は好適ではありません。通常、これらの再分散は、再分散キーを使用してテーブルが結合されないために行われます。ファクトテーブルにまだ分散キーが指定されていない場合は、両方のテーブルの分散キーとして結合列を指定します。ファクトテーブルの別の列にディストリビューションキーがすでに指定されている場合は、この結合をコロケーションするためにディストリビューションキーを変更することが全般的なパフォーマンスの向上につながるかどうかを評価します。外部テーブルのディストリビューションキーを変更することが最適な選択肢でない場合は、内部テーブルに DISTSTYLE ALL を指定することによってコロケーションを実現できます。

次の例は、DS_BCAST_INNER ラベルと DS_DIST_NONE ラベルを持つクエリプランの一部を示しています。

```
->  XN Hash Join DS_BCAST_INNER  (cost=112.50..3272334142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
        ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..3167290276.71 rows=172456
 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

DISTSTYLE ALL を使用するようディメンションテーブルに変更を加えると、同じクエリのクエリプランに DS_BCAST_INNER ではなく DS_DIST_ALL_NONE が表示されるようになります。また、結合ステップの相対的なコストも大幅に変化します。合計コストは前のクエリの `3272334142.59` と比べて `14142.59` です。

```
->  XN Hash Join DS_DIST_ALL_NONE  (cost=112.50..14142.59 rows=170771 width=84)
        Hash Cond: ("outer".venueid = "inner".venueid)
        ->  XN Hash Join DS_DIST_ALL_NONE  (cost=109.98..10276.71 rows=172456 width=47)
              Hash Cond: ("outer".eventid = "inner".eventid)
              ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6286.47 rows=172456 width=30)
                    Merge Cond: ("outer".listid = "inner".listid)
                    ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497
 width=14)
                    ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=24)
```

クエリプランの例

この例では、クエリプランを評価して分散最適化の機会を特定する方法を示します。

EXPLAIN コマンドを使用して次のクエリを実行し、クエリプランを作成します。

```
explain
select lastname, catname, venuename, venuecity, venuestate, eventname,
month, sum(pricepaid) as buyercost, max(totalprice) as maxtotalprice
from category join event on category.catid = event.catid
join venue on venue.venueid = event.venueid
join sales on sales.eventid = event.eventid
join listing on sales.listid = listing.listid
join date on sales.dateid = date.dateid
join users on users.userid = sales.buyerid
group by lastname, catname, venuename, venuecity, venuestate, eventname, month
having sum(pricepaid)>9999
order by catname, buyercost desc;
```

TICKIT データベースでは、SALES がファクトテーブルであり、LISTING がその最大のディメンションです。テーブルをコロケーションするために、SALES は LISTING の外部キーである LISTID を使用して分散され、LISTING はそのプライマリキーである LISTID を使用して分散されます。次の例は、SALES および LISTING の CREATE TABLE コマンドを示しています。

```
create table sales(
 salesid integer not null,
 listid integer not null distkey,
 sellerid integer not null,
 buyerid integer not null,
 eventid integer not null encode mostly16,
```

```
 dateid smallint not null,
 qtysold smallint not null encode mostly8,
 pricepaid decimal(8,2) encode delta32k,
 commission decimal(8,2) encode delta32k,
 saletime timestamp,
 primary key(salesid),
 foreign key(listid) references listing(listid),
 foreign key(sellerid) references users(userid),
 foreign key(buyerid) references users(userid),
 foreign key(dateid) references date(dateid))
        sortkey(listid,sellerid);

create table listing(
 listid integer not null distkey sortkey,
 sellerid integer not null,
 eventid integer not null encode mostly16,
 dateid smallint not null,
 numtickets smallint not null encode mostly8,
 priceperticket decimal(8,2) encode bytedict,
 totalprice decimal(8,2) encode mostly32,
 listtime timestamp,
 primary key(listid),
 foreign key(sellerid) references users(userid),
 foreign key(eventid) references event(eventid),
 foreign key(dateid) references date(dateid));
```

次のクエリプランでは、SALES および LISTING を使用した結合のマージ結合ステップに、そのステップで再分散が必要とならないことを意味する DS_DIST_NONE が示されています。ただし、クエリプランを上に移動するとわかるとおり、他の内部結合には、クエリ実行の一環として内部テーブルがブロードキャストされることを意味する DS_BCAST_INNER が示されています。キー分散を使用してコロケーションできるテーブルは 1 ペアのみであるため、5 つのテーブルを再ブロードキャストする必要があります。

```
QUERY PLAN
XN Merge  (cost=1015345167117.54..1015345167544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  ->  XN Network  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
        Send to leader
        ->  XN Sort  (cost=1015345167117.54..1015345167544.46 rows=170771 width=103)
              Sort Key: category.catname, sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=15345150568.37..15345152276.08 rows=170771
 width=103)
```

```
                   Filter: (sum(pricepaid) > 9999.00)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=742.08..15345146299.10
rows=170771 width=103)
                          Hash Cond: ("outer".catid = "inner".catid)
                          ->  XN Hash Join DS_BCAST_INNER
(cost=741.94..15342942456.61 rows=170771 width=97)
                                Hash Cond: ("outer".dateid = "inner".dateid)
                                ->  XN Hash Join DS_BCAST_INNER
(cost=737.38..15269938609.81 rows=170766 width=90)
                                      Hash Cond: ("outer".buyerid = "inner".userid)
                                      ->  XN Hash Join DS_BCAST_INNER
(cost=112.50..3272334142.59 rows=170771 width=84)
                                            Hash Cond: ("outer".venueid =
"inner".venueid)
                                            ->  XN Hash Join DS_BCAST_INNER
(cost=109.98..3167290276.71 rows=172456 width=47)
                                                  Hash Cond: ("outer".eventid =
"inner".eventid)
                                                  ->  XN Merge Join DS_DIST_NONE
(cost=0.00..6286.47 rows=172456 width=30)
                                                        Merge Cond: ("outer".listid =
"inner".listid)
                                                        ->  XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
                                                        ->  XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
                                                  ->  XN Hash  (cost=87.98..87.98
rows=8798 width=25)
                                                        ->  XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
                                            ->  XN Hash  (cost=2.02..2.02 rows=202
width=41)
                                                  ->  XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
                                      ->  XN Hash  (cost=499.90..499.90 rows=49990
width=14)
                                            ->  XN Seq Scan on users
(cost=0.00..499.90 rows=49990 width=14)
                                ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                      ->  XN Seq Scan on date  (cost=0.00..3.65
rows=365 width=11)
                          ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
```

```
                                -> XN Seq Scan on category  (cost=0.00..0.11 rows=11
 width=10)
```

DISTSTYLE ALL を使用してテーブルを変更することは 1 つの解決策です。

```
ALTER TABLE users ALTER DISTSTYLE ALL;
ALTER TABLE venue ALTER DISTSTYLE ALL;
ALTER TABLE category ALTER DISTSTYLE ALL;
ALTER TABLE date ALTER DISTSTYLE ALL;
ALTER TABLE event ALTER DISTSTYLE ALL;
```

EXPLAIN を使用して同じクエリを再実行し、新しいクエリプランを検証します。DISTSTYLE ALL を使用して全ノードにデータが分散されたため再分散が必要とならないことを意味する DS_DIST_ALL_NONE が、結合に表示されるようになります。

```
QUERY PLAN
XN Merge  (cost=1000000047117.54..1000000047544.46 rows=1000 width=103)
  Merge Key: category.catname, sum(sales.pricepaid)
  ->  XN Network  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
        Send to leader
        ->  XN Sort  (cost=1000000047117.54..1000000047544.46 rows=170771 width=103)
              Sort Key: category.catname, sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=30568.37..32276.08 rows=170771 width=103)
                    Filter: (sum(pricepaid) > 9999.00)
                    ->  XN Hash Join DS_DIST_ALL_NONE  (cost=742.08..26299.10
 rows=170771 width=103)
                          Hash Cond: ("outer".buyerid = "inner".userid)
                          ->  XN Hash Join DS_DIST_ALL_NONE  (cost=117.20..21831.99
 rows=170766 width=97)
                                Hash Cond: ("outer".dateid = "inner".dateid)
                                ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=112.64..17985.08 rows=170771 width=90)
                                      Hash Cond: ("outer".catid = "inner".catid)
                                      ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=112.50..14142.59 rows=170771 width=84)
                                            Hash Cond: ("outer".venueid =
 "inner".venueid)
                                            ->  XN Hash Join DS_DIST_ALL_NONE
 (cost=109.98..10276.71 rows=172456 width=47)
                                                  Hash Cond: ("outer".eventid =
 "inner".eventid)
                                                  ->  XN Merge Join DS_DIST_NONE
 (cost=0.00..6286.47 rows=172456 width=30)
```

```
                                                       Merge Cond: ("outer".listid =
"inner".listid)
                                                       ->  XN Seq Scan on listing
(cost=0.00..1924.97 rows=192497 width=14)
                                                       ->  XN Seq Scan on sales
(cost=0.00..1724.56 rows=172456 width=24)
                                                 ->  XN Hash  (cost=87.98..87.98
rows=8798 width=25)
                                                       ->  XN Seq Scan on event
(cost=0.00..87.98 rows=8798 width=25)
                                           ->  XN Hash  (cost=2.02..2.02 rows=202
width=41)
                                                 ->  XN Seq Scan on venue
(cost=0.00..2.02 rows=202 width=41)
                                     ->  XN Hash  (cost=0.11..0.11 rows=11 width=10)
                                           ->  XN Seq Scan on category
(cost=0.00..0.11 rows=11 width=10)
                               ->  XN Hash  (cost=3.65..3.65 rows=365 width=11)
                                     ->  XN Seq Scan on date  (cost=0.00..3.65
rows=365 width=11)
                         ->  XN Hash  (cost=499.90..499.90 rows=49990 width=14)
                               ->  XN Seq Scan on users  (cost=0.00..499.90 rows=49990
width=14)
```

分散の例

次の例は、CREATE TABLE ステートメントで定義したオプションに応じてデータがどのように分散されるかを示しています。

DISTKEY の例

TICKIT データベースの USERS テーブルのスキーマを確認します。USERID が SORTKEY 列および DISTKEY 列として定義されています。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'users';

    column     |          type          | encoding | distkey | sortkey
---------------+------------------------+----------+---------+---------
 userid        | integer                | none     | t       |       1
 username      | character(8)           | none     | f       |       0
 firstname     | character varying(30)  | text32k  | f       |       0
```

```
...
```

このテーブルの分散列として USERID を選択するのは適切です。SVV_DISKUSAGE システムビューに対してクエリを実行すると、テーブルが非常に均等に分散されていることがわかります。列数はゼロベースであるため、USERID は列 0 です。

```
select slice, col, num_values as rows, minvalue, maxvalue
from svv_diskusage
where name='users' and col=0 and rows>0
order by slice, col;

slice| col | rows  | minvalue | maxvalue
-----+-----+-------+----------+----------
0    | 0   | 12496 | 4        | 49987
1    | 0   | 12498 | 1        | 49988
2    | 0   | 12497 | 2        | 49989
3    | 0   | 12499 | 3        | 49990
(4 rows)
```

テーブルには 49,990 行が含まれます。行 (num_values) 列は、各スライスにほぼ同じ数の行が含まれることを示します。minvalue 列と maxvalue 列は、各スライスの値の範囲を示します。各スライスには、値のほぼ全範囲が含まれるため、各スライスは、ユーザー ID の範囲をフィルターするクエリの実行に参加する可能性が高くなります。

この例は、小規模なテストシステムでの分散を示しています。通常、スライスの合計数はこれよりかなり多くなります。

STATE 列を使用してよく参加または結合する場合は、STATE 列で分散する選択を行うことができます。次の例では、USERS テーブルと同じデータを含む新しいテーブルを作成し、DISTKEY を STATE 列に設定した場合を示しています。この場合、ディストリビューションは均等ではありません。スライス 0 (13,587 行) には、スライス 3 (10,150 行) よりも約 30% 多く行が含まれます。これよりかなり大きなテーブルでは、この量の分散スキューがあるとクエリ処理に悪影響を及ぼす可能性があります。

```
create table userskey distkey(state) as select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userskey' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
```

```
------+-----+-------+----------+----------
    0 |   0 | 13587 |        5 |    49989
    1 |   0 | 11245 |        2 |    49990
    2 |   0 | 15008 |        1 |    49976
    3 |   0 | 10150 |        4 |    49986
(4 rows)
```

DISTSTYLE EVEN の例

USERS テーブルと同じデータを含む新しいテーブルを作成し、DISTSTYLE を EVEN に設定した場合、行はスライス間で常に均等に分散されます。

```
create table userseven diststyle even as
select * from users;

select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'userseven' and col=0 and rows>0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
------+-----+-------+----------+----------
    0 |   0 | 12497 |        4 |    49990
    1 |   0 | 12498 |        8 |    49984
    2 |   0 | 12498 |        2 |    49988
    3 |   0 | 12497 |        1 |    49989
(4 rows)
```

ただし、分散が特定の列に基づいて行われないので、特にテーブルが他のテーブルに結合される場合に、クエリ処理の質が低下する可能性があります。結合列で分散が行われないと、多くの場合、効率的に実行できるタイプの結合操作に影響を及ぼします。結合、集計、およびグループ化操作は、両方のテーブルがそれぞれの結合列で分散およびソートされる場合に最適化されます。

DISTSTYLE ALL の例

USERS テーブルと同じデータを使用して新しいテーブルを作成して、DISTSTYLE を ALL に設定すると、すべての行が各ノードの最初のスライスに分散されます。

```
select slice, col, num_values as rows, minvalue, maxvalue from svv_diskusage
where name = 'usersall' and col=0 and rows > 0
order by slice, col;

slice | col | rows  | minvalue | maxvalue
```

```
------+-----+-------+----------+----------
     0 |    0 | 49990 |         4 |     49990
     2 |    0 | 49990 |         2 |     49990

(4 rows)
```

ソートキー

> **Note**
>
> SORTKEY AUTO を使用してテーブルを作成することをお勧めします。その場合、Amazon Redshift は自動テーブル最適化を使用してソートキーを選択します。詳細については、「自動テーブル最適化」を参照してください。このセクションの残りの部分では、ソート順について詳しく説明します。

テーブルの作成時に、代わりにその 1 つ以上の列をソートキーとして定義することもできます。データが空のテーブルに最初にロードされると、行がディスク上にソート順に格納されます。ソートキー列に関する情報がクエリプランナーに渡され、プランナはこの情報を使用して、データのソート方法を利用するプランを構築します。詳細については、「CREATE TABLE」を参照してください。ソートキーを作成する際のベストプラクティスについては、「最良のソートキーの選択」を参照してください。

ソートは、範囲が制限された述語を効率的に処理することができます。Amazon Redshift は、列データを 1 MB のディスクブロックに保存します。各ブロックの最小値と最大値がメタデータの一部として格納されます。範囲が制限された述語をクエリが使用する際には、クエリプロセッサはこの最小値と最大値を使用します。これにより、テーブルスキャン中に多数のブロックをすばやくスキップすることができます。例えば、日付でソートされた 5 年間のデータがテーブルに保存されており、クエリによって 1 か月間の日付範囲が指定されているとします。この場合、スキャンからディスクブロックの最大 98% を削除できます。データがソートされない場合は、より多くの (場合によっては、すべての) ディスクブロックをスキャンする必要があります。

複合キーまたはインターリーブソートキーを指定できます。複合ソートキーは、クエリ述語が、ソートキー列のサブセットの順序であるプレフィックスを使用する場合に効果的です。インターリーブソートキーは、ソートキーの各列に同じ重み付けをするため、クエリ述語が任意の順序でソートキーを構成する列のサブセットを使用できます。

選択されたソートキーがクエリパフォーマンスに与える影響を把握するには、EXPLAIN コマンドを使用します。詳細については、「クエリプランと実行ワークフロー」を参照してください。

ソートタイプを定義するには、CREATE TABLE または CREATE TABLE AS ステートメントで INTERLEAVED または COMPOUND キーワードを使用します。デフォルトは COMPOUND です。INSERT、UPDATE、または DELETE オペレーションを使用してテーブルを定期的に更新する場合は、COMPOUND を使用することをお勧めします。INTERLEAVED ソートキーは最大 8 列で使用できます。データとクラスターのサイズに応じて、VACUUM REINDEX は、インターリーブソートキーを分析する目的で追加パスを作成するため、VACUUM FULL よりも大幅に実行時間が長くなります。インターリーブテーブルでは、ソート操作およびマージ操作の時間が長くなる場合があります。これは、インターリーブソートでは、複合ソートよりも多くの行の再調整が必要になる可能性があるためです。

テーブルのソートキーを表示するには、SVV_TABLE_INFO システムビューに対してクエリを実行します。

トピック

- 多次元データレイアウトのソート (プレビュー)
- 複合ソートキー
- インターリーブソートキー

多次元データレイアウトのソート (プレビュー)

以下は、多次元データレイアウトソートに関するプレリリースドキュメントで、プレビューリリース版です。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

> **Note**
>
> この機能はプレビュークラスターとプレビューワークグループでのみ使用できます。プレビュークラスターを作成するには、「Amazon Redshift 管理ガイド」の「プレビュークラスターの作成」を参照してください。プレビューワークグループを作成するには、「Amazon Redshift 管理ガイド」の「プレビューワークグループの作成」を参照してください。

多次元データレイアウトソートキーは、ワークロード内の反復述語に基づく AUTO ソートキータイプの一つです。ワークロードに反復述語がある場合、Amazon Redshift は反復述語を満たすデータ行

をコロケーションすることでテーブルスキャンのパフォーマンスを向上させることができます。多次元データレイアウトソートキーでは、テーブルのデータを厳密な列順序で保存する代わりに、ワークロードに現れる反復述語を分析してデータを格納します。1 つのワークロードに複数の反復述語が見られる場合があります。ワークロードによっては、このようなソートキーを使用すると多くの述語のパフォーマンスが向上します。Amazon Redshift は、このソートキーメソッドを AUTO ソートキーで定義されたテーブルに使用すべきかどうかを自動的に判断します。

例えば、データを列の順序でソートしたテーブルがあるとします。多くのデータブロックを調べて、それらがワークロードの述語を満たしているかどうかを判断する必要があるかもしれません。ただし、データが述語順にディスクに保存されている場合は、クエリを満たすためにスキャンする必要があるブロックの数が少なくなります。このような場合は、多次元データレイアウトソートキーを使用すると便利です。

クエリが多次元データレイアウトキーを使用しているかどうかを確認するには、SYS_QUERY_DETAIL ビューの `step_attribute` 列を参照してください。値が `multi-dimensional` の場合、クエリには多次元データレイアウトが使用されています。AUTO ソートキーで定義されたテーブルが多次元データレイアウトを使用しているかどうかを確認するには、SVV_TABLE_INFO ビューの `sortkey1` 列を参照してください。値が `padb_internal_mddl_key_col` である場合は、テーブルのソートキーに多次元データレイアウトが使用されています。

Amazon Redshift が多次元データレイアウトソートキーを使用しないようにするには、`SORTKEY AUTO` 以外の別のテーブルソートキーオプションを選択します。SORTKEY オプションの詳細については、「CREATE TABLE」を参照してください。

複合ソートキー

複合キーは、ソートキー定義内にリストされているすべての列で構成されています。順序はリストされている順です。複合ソートキーは、クエリのフィルターがソートキーのプレフィックスを使用してフィルターや結合などの条件を適用する場合に便利です。複合ソートを実行する利点は、クエリがセカンダリソート列のみに依存しプライマリ列を参照しない場合には薄くなります。COMPOUND はデフォルトのソート形式です。

複合ソートキーで、結合、GROUP BY および ORDER BY 操作、PARTITION BY や ORDER BY を使用したウィンドウ関数の速度が上がる場合があります。例えば、データが結合列で分散および事前にソートされる場合は、ハッシュ結合よりも迅速なことの多いマージ結合が適しています。複合ソートキーは、圧縮の向上にも役立ちます。

すでにデータが含まれているソート済みテーブルに行を追加すると、未ソートのリージョンが増加し、パフォーマンスに大きな影響が生じます。影響は、テーブルがインターリーブソートを使用する場合、特にソート列が日付やタイムスタンプの列など一定間隔で増加するデータを含む場合、いっそう大きくなります。定期的に VACUUM 操作を実行してください。特に大量のデータをロードした後は、データを再ソートし再分析するために必要です。詳細については、「未ソートリージョンのサイズを管理する」を参照してください。バキューム処理を実行してデータを再ソートした後は、ANALYZE コマンドを実行してクエリプランナー用の統計メタデータを更新することをお勧めします。詳細については、「テーブルを分析する」を参照してください。

インターリーブソートキー

インターリーブソートキーは、ソートキー内の各列または列のサブセットに同じ重み付けをします。複数のクエリが 1 つのフィルターで異なる列を使用する場合、インターリーブソート形式を使用することで、多くの場合これらのクエリのパフォーマンスが向上します。クエリがセカンダリソート列で制限述語を使用する場合、インターリーブソートは複合ソートに比べて大幅にクエリのパフォーマンスが向上します。

> **Important**
>
> ID 列、日付、タイムスタンプなど、一定間隔で増加する属性を持つ列で、インターリーブソートキーを使用しないでください。

インターリーブソートキーの実行によって得られるパフォーマンスの改善は、ロードの増分とバキューム処理の回数によって異なります。

インターリーブソートは、例えば `select c_name from customer where c_region = 'ASIA'` のように WHERE 句のソートキー列をフィルタリングする非常に選択的なクエリの場合に最も効果的です。インターリーブソートの効果は、制限されたソート済み列の数で向上します。

インターリーブソートは大きなテーブルに対してより効果的です。ソートは各スライスに適用されます。したがって、スライスごとに 1 MB のブロックを複数必要とする大きなテーブルの場合、インターリーブソートが最も効果的です。ここで、クエリプロセッサは、制限述語を使用してブロックのかなりの割合をスキップできます。テーブルが使用するブロック数を表示するには、STV_BLOCKLIST システムビューに対してクエリを実行します。

単一の列を並べ替える場合、列の値に長い共通プレフィックスがある場合は、インターリーブソートの方が複合ソートよりもパフォーマンスがいい場合があります。例えば、URL は一般的に「http://

www」で始まります。複合ソートキーはプレフィックスから使用する文字数に制限があるため、大量のキーの重複が発生します。インターリーブソートは、ゾーンのマッピング値に内部圧縮方式を使用するため、長い共通プレフィックスがある列の値をよりよく識別できます。

小さな Amazon Redshift プロビジョニングクラスターを Amazon Redshift Serverless に移行すると、Redshift はインターリーブソートキーと DISTSTYLE KEY を含むテーブルを複合ソートキーに変換します。DISTSTYLE は変更されません。分散スタイルの詳細については、「データディストリビューションスタイルの操作」を参照してください。

VACUUM REINDEX

すでにデータが含まれているソート済みテーブルに行を追加すると、パフォーマンスが時間とともに悪化することがあります。この悪化は複合ソートとインターリーブソートの両方で発生しますが、インターリーブテーブルでより影響が大きくなります。VACUUM はソート順序を復元しますが、インターリーブテーブルの場合は操作に時間がかかります。これは、新規のインターリーブデータのマージを行う際に各データブロックの変更が必要になる場合があるためです。

テーブルが最初にロードされると、Amazon Redshift がソートキー列の値の分散を分析し、その情報を基に適切なインターリービングをソートキー列に実行します。テーブルが大きくなるにつれて、ソートキー列の値の分散は、特に日付またはタイムスタンプ列で、変化したり不均等になったりします。不均等が大きくなりすぎると、パフォーマンスに影響を与えます。ソートキーを再分析してパフォーマンスを復元するには、REINDEX キーワードとともに VACUUM コマンドを実行します。インターリーブテーブルの場合、データに対して追加の分析パスを取る必要があるため、VACUUM REINDEX は標準の VACUUM よりも時間がかかります。キーの分散スキューおよび直近のインデックス再作成時間を表示するには、SVV_INTERLEAVED_COLUMNS システムビューにクエリを実行します。

VACUUM の実行頻度および VACUUM REINDEX の実行時期についての詳細は、「インデックスを再生成するかどうかの決定」を参照してください。

テーブルの制約

一意性、プライマリキー、および外部キーの制約は情報提供のみを目的としており、テーブルに値を入れるときに Amazon Redshift によって強要されるわけではありません。例えば、依存関係のあるテーブルにデータを挿入する場合、制約に違反していても挿入は成功します。ただし、プライマリキーと外部キーはプランニング時のヒントとして使用されます。アプリケーションの ETL プロセスまたは他の何らかのプロセスによってこれらのキーの整合性が強要される場合は、これらのキーを宣言する必要があります。

例えば、クエリプランナーは、特定の統計計算でプライマリキーと外部キーを使用します。これは、サブクエリの非相関化手法に影響を与える一意性と参照関係を推測するために行われます。これにより、多数の結合を配列し、冗長な結合を排除できます。

プランナはこれらのキーの関係を活用しますが、Amazon Redshift テーブルのすべてのキーがロード時に有効であることが前提となります。アプリケーションが無効な外部キーまたはプライマリキーを許可する場合、いくつかのクエリが不正な結果を返す可能性があります。例えば、プライマリキーが一意でない場合、SELECT DISTINCT クエリが重複した行を返すことがあります。有効かどうかわからない場合は、テーブルに対してキーの制約を定義しないでください。ただし、有効だとわかっている場合は、プライマリキー、外部キー、および一意性の制約を必ず宣言してください。

Amazon Redshift は、NOT NULL 列の制約を適用します。

テーブルの制約の詳細については、「CREATE TABLE」を参照してください。依存関係のあるテーブルを削除する方法については、「DROP TABLE」を参照してください。

Amazon Redshift でのデータのロード

Amazon Redshift データベースにデータをロードするには、いくつかの方法があります。データのロード元として一般的なソースの 1 つは、Amazon S3 ファイルです。次の表は、Amazon S3 ソースから開始する場合に使用するいくつかの方法を示しています。

使用する方法	説明	方法が必要になる場合
COPY コマンド	バッチファイルの取り込みを実行して、Amazon S3 ファイルからデータをロードします。この方法では、Amazon Redshift の並列処理機能を活用します。詳細については、「COPY コマンドを使ってテーブルをロードする」を参照してください。	バッチファイルの取り込みを手動で開始するための基本的なデータロード要件が必要な場合に使用します。この方法は、主にカスタムおよびサードパーティのファイル取り込みパイプラインで使用するか、1 回限りまたはアドホックのファイル取り込みワークロードで使用します。
COPY... CREATE JOB コマンド (自動コピー)	追跡中の Amazon S3 パスで新しいファイルを作成すると、COPY コマンドが自動的に実行されます。詳細については、「Amazon S3 からの継続的なファイル取り込みによるテーブルのロード (プレビュー)」を参照してください。	Amazon S3 で新しいファイルを作成するときに、ファイル取り込みパイプラインでデータを自動的に取り込む必要がある場合に使用します。Amazon Redshift は、データの重複を防ぐために、取り込んだファイルを追跡し続けます。この方法では、Amazon S3 バケット所有者による設定が必要です。
データレイククエリからのロード	外部テーブルを作成して Amazon S3 ファイルに対してデータレイククエリを実行し、INSERT INTO コマンドを実行してデータレイククエリ	次のいずれかのシナリオで使用します。 • AWS Glue およびオープンテーブル形式 (Apache Iceberg、Apache

使用する方法	説明	方法が必要になる場合
	の結果をローカルテーブル内にロードします。詳細については、「Redshift Spectrum 用の外部テーブル」を参照してください。	Hudi、Delta Lake など) からロードする場合。 • ソースファイルを部分的に取り込む必要がある場合 (WHERE 句を実行して特定の行を取り込む場合など)。 • 特定の列の取り込み (SELECT コマンドの実行など) や、外出先での基本的なデータ変換 (ソースファイルの値に対する基本的なオペレーションの適用、UDF の呼び出しなど) を行うために、より柔軟性が必要な場合。
その他の検討可能な方法		

使用する方法	説明	方法が必要になる場合
ストリーミング取り込み	ストリーミング取り込みでは、Amazon Kinesis Data Streams や Amazon Managed Streaming for Apache Kafka から、Amazon Redshift でプロビジョニングされたビューや Redshift Serverless マテリアライズドビューに、ストリームデータを低レイテンシーかつ高速で取り込むことができます。詳細については、Amazon Kinesis Data Streams からストリーミング取り込みを開始する方法およびAmazon Managed Streaming for Apache Kafka (Amazon MSK) からのストリーミング取り込みを開始するを参照してください。	データを最初に Amazon S3 のファイルにストリーミングし、次に Amazon S3 からロードする場合のユースケースで検討します。Amazon S3 にデータを保持する必要がない場合は、通常、データを Amazon Redshift 内に直接ストリーミングすることを検討できます。
データレイククエリの実行	テーブルの内容をローカルテーブルに取り込む代わりに、データレイクテーブルから直接クエリを実行します。詳細については、「Amazon Redshift Spectrum」を参照してください。	Amazon Redshift でのローカルテーブルクエリのパフォーマンスを必要としないユースケースで使用します。

使用する方法	説明	方法が必要になる場合
Amazon Redshift クエリエディタ v2 を使用したバッチロード	Amazon Redshift クエリエディタ v2 で、バッチファイル取り込みワークロードを視覚的に準備して実行できます。詳細については、「Amazon Redshift 管理ガイド」の「S3 からデータをロードする」を参照してください。	クエリエディタ v2 で COPY ステートメントを準備する場合、COPY ステートメントの準備プロセスを簡素化するビジュアルツールが必要なときに使用します。
Amazon Redshift クエリエディタ v2 を使用したローカルファイルからのデータのロード	デスクトップから Amazon Redshift テーブルにファイルを直接アップロードできます。Amazon S3 にファイルを手動でアップロードする必要はありません。詳細については、「Amazon Redshift 管理ガイド」の「ローカルファイル設定とワークフローからのデータのロード」を参照してください。	1 回限りのクエリのためにローカルコンピュータからファイルをすばやくロードする必要がある場合に使用します。この方法の場合、Amazon Redshift クエリエディタ v2 は、お客様所有の Amazon S3 バケットにファイルを一時的に保存し、この Amazon S3 パスを使用してコピーコマンドを実行します。

COPY コマンドは、テーブルをロードする最も効率的な方法です。INSERT コマンドを使って データをテーブルに追加することもできます。ただし、COPY コマンドを使ったほうが効率的で す。COPY コマンドは、複数のデータファイルまたは複数のデータストリームから同時に読み込む ことができます。Amazon Redshift はワークロードを Amazon Redshift ノードに割り当て、ノード スライス間での行のソートやデータの分散など、ロードオペレーションを並列で実行します。

> **Note**
>
> Amazon Redshift Spectrum の外部テーブルは読み込み専用です。外部テーブルには COPY または INSERT できません。

Amazon Redshift が他の AWS リソースのデータにアクセスするには、これらのリソースにアク セスして必要なアクションを実行するためのアクセス許可が必要です。AWS Identity and Access

Management (IAM) を使用すると、Amazon Redshift のリソースとデータに対するユーザーのアクセスを制限できます。

初回のデータロードの後、大量のデータを追加、変更、削除した場合、VACUUM コマンドを実行してデータを再編成し、削除後の領域を利用可能な状態に戻してください。また、ANALYZE コマンドを実行し、テーブル統計を更新します。

トピック

COPY コマンドを使ってテーブルをロードする

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを活用し、Amazon S3 のファイル、DynamoDB テーブル、リモートホストから出力されたテキストのいずれかから並列でデータをロードします。

COPY コマンドのすべてのオプションを学習する前に、Amazon S3 データをロードするための基本的なオプションを学習することをお勧めします。「Amazon Redshift 入門ガイド」は、デフォルトの IAM ロールを使用して Amazon S3 データをロードするための COPY コマンドの簡単な使用方法を示しています。詳細については、「ステップ 4: Amazon S3 から Amazon Redshift にデータをロードする」を参照してください。

> **Note**
>
> 大量のデータをロードする場合、COPY コマンドを使うことをお勧めします。個々に INSERT ステートメントを使ってテーブルにデータを入力すると著しく時間がかかる場合があります。または、他の Amazon Redshift データベーステーブルにデータが既に存在する場合、パフォーマンスを向上させるには INSERT INTO ... SELECT または CREATE TABLE

AS を使用します。詳細については、「INSERT」または「CREATE TABLE AS」を参照してください。

Amazon Redshift で別の AWS リソースからデータをロードするには、リソースにアクセスして必要なアクションを実行するためのアクセス許可が必要です。

COPY コマンドを使ってデータをテーブルにロードするための権限を与えるか、取り消すには、INSERT 権限を与えるか、取り消します。

データを Amazon Redshift テーブルにロードするために、適切な形式にする必要があります。このセクションでは、ロードする前のデータを準備し確認するための指針、ならびに実行する前の COPY ステートメントを検証するための指針を紹介します。

Amazon S3 バケットにアップロードするデータファイルの情報を保護するには、事前に暗号化しておきます。COPY を実行すると、ロード時にデータが復号されます。一時的セキュリティ認証情報をユーザーに与えることで、データロードのアクセスを制限することもできます。一時的セキュリティ認証情報はセキュリティを強化します。使用期限が短く、期限が切れた後は再利用できないためです。

Amazon Redshift には、区切られた非圧縮データをすばやくロードするための COPY 機能が組み込まれています。加えて、gzip、lzop、bzip2 のいずれかによりファイルを圧縮することで、そのファイルのアップロードにかかる時間を短縮できます。

COPY クエリに ESCAPE、REMOVEQUOTES、および FIXEDWIDTH のキーワードが含まれる場合、非圧縮データの自動分割はサポートされませんが、CSV キーワードの場合はサポートされます。

Amazon Redshift では、AWS クラウド内で転送されるデータのセキュリティを確保するために、COPY、UNLOAD、バックアップ、復旧オペレーションの実行時の Amazon S3 または Amazon DynamoDB との通信に、ハードウェアでアクセラレーションされた SSL を使用します。

Amazon DynamoDB テーブルからデータを直接ロードするとき、Amazon DynamoDB がプロビジョニングするスループットの消費量を制御するオプションがあります。

任意で COPY を使い、入力データを分析したり、ロードプロセスの一部として最適な圧縮エンコーディングをテーブルに自動的に適用したりできます。

トピック

- 認証情報とアクセス許可
- 入力データを準備する
- Amazon S3 からデータをロードする
- Amazon EMR からのデータのロード
- リモートホストからデータをロードする
- Amazon DynamoDB テーブルからのデータのロード
- データが正しくロードされたことを確認する
- 入力データを検証する
- 自動圧縮ありでテーブルをロードする
- 細長いテーブルのストレージの最適化
- デフォルトの列値をロードする
- データロードのトラブルシューティング

認証情報とアクセス許可

Amazon S3、Amazon DynamoDB、Amazon EMR、Amazon EC2 など、別の AWS リソースを Amazon Redshift で使用してデータをロードまたはアンロードするには、リソースにアクセスし、必要なアクションを実行してデータを利用するためのアクセス許可が必要です。例えば、Amazon S3 からデータをロードする場合、COPY はバケットへの LIST アクセスとバケットオブジェクトへの GET アクセスが必要です。

リソースへのアクセス認可を取得するには、Amazon Redshift の認証が必要です。ロールベースのアクセスコントロールまたはキーに基づくアクセスコントロールのいずれかを選択できます。このセクションは 2 つの方法の概要を示します。詳細と例については、他の AWS リソースにアクセスするアクセス許可 を参照してください。

ロールベースアクセスコントロール

ロールベースのアクセスコントロールを使用する場合、Amazon Redshift はユーザーに代わって AWS Identity and Access Management (IAM) ロールを一時的に引き受けます。次に、ロールに付与された許可に応じて、Amazon Redshift は必要な AWS リソースにアクセスできます。

ロールベースのアクセスコントロールの利用をお勧めします。これにより、AWS 認証情報が保護されるだけでなく、AWS のリソースと機密性のあるユーザーデータに対するアクセスがより安全になり、きめ細やかなコントロールが行われます。

ロールベースのアクセスコントロールを使用するには、まず Amazon Redshift サービスロールタイプを使用して IAM ロールを作成し、次にロールをデータウェアハウスにアタッチする必要があります。ロールは、少なくとも COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可 に示されたアクセス権限が必要です。IAM ロールを作成してクラスターにアタッチする手順については、「Amazon Redshift 管理ガイド」の「Amazon Redshift クラスターによって AWS のサービスへのアクセスを許可する IAM ロールを作成する」を参照してください。

クラスターにロールを追加するか、Amazon Redshift マネジメントコンソール、CLI、または API を使用してクラスターに関連付けられるロールを表示できます。詳細については、「Amazon Redshift 管理ガイド」の「IAM ロールを使用した COPY および UNLOAD オペレーションの認可」を参照してください。

IAM ロールを作成する場合、IAM はロールの Amazon リソースネーム (ARN) を返します。IAM ロールを使用して COPY コマンドを実行するには、IAM_ROLE パラメータまたは CREDENTIALS パラメータを介してロールの ARN を指定します。

次の COPY コマンドの例では、ロール `MyRedshiftRole` を指定した IAM_ROLE パラメータを認証に使用します。

```
COPY customer FROM 's3://amzn-s3-demo-bucket/mydata'
IAM_ROLE 'arn:aws:iam::12345678901:role/MyRedshiftRole';
```

AWS ユーザーは、少なくとも COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可 に示されたアクセス許可が必要です。

キーベースのアクセスコントロール

キーベースのアクセスコントロールでは、データがある AWS リソースへのアクセスを許可されたユーザーのアクセスキー ID とシークレットアクセスキーを指定します。

> **Note**
>
> プレーンテキストのアクセスキー ID とシークレットアクセスキーを指定する代わりに、認証のために IAM ロールを使用することを強くお勧めします。キーベースのアクセスコントロールを選択する場合は、AWSアカウント (ルート) 認証情報を使用しないでください。常に IAM ユーザーを作成し、そのユーザーのアクセスキー ID とシークレットアクセスキーを指定します。IAM ユーザーを作成する手順については、「AWS アカウントでの IAM ユーザーの作成」を参照してください。

入力データを準備する

入力データとそれを受け取るテーブル列の間に互換性がない場合、COPY コマンドは失敗します。

次の指針を利用し、入力データの妥当性を確認してください。

- データには最大 4 バイトの UTF-8 文字のみを含めることができます。
- CHAR 文字列と VARCHAR 文字列がそれに対応する列の長さを超えないことを確認してください。VARCHAR 文字列は文字数ではなく、バイト数でカウントされます。そのため、例えば、4 バイトを使う漢字が 4 文字集まった文字列は VARCHAR(16) 列を必要とします。
- マルチバイト文字は VARCHAR 列との連動でのみ使用できます。マルチバイト文字が 4 バイトを超えないことを確認します。
- CHAR 列のデータにシングルバイトの文字のみが含まれることを確認します。
- レコードの最後のフィールドを示すために特殊な文字や構文を含めないでください。このフィールドは区切り文字にすることができます。
- NUL (UTF-8 0000) またはバイナリゼロ (0x000) とも呼ばれる null ターミネーターをデータに含める場合、これらの文字は NULLS として CHAR 列または VARCHAR 列にロードすることができます。その際、COPY コマンドでは NULL AS オプションを使用します (`null as '\0'` または `null as '\000'`)。NULL AS を使用しない場合、null ターミネーターが原因となり、COPY が失敗します。
- 文字列に、区切り文字や埋め込み改行など、特殊文字が含まれる場合、COPY コマンドとともに ESCAPE オプションを使用します。
- 一重引用符と二重引用符がすべて適切に一致していることを確認してください。
- 浮動小数点の文字列が 12.123 のような標準浮動小数点形式と 1.0E4 のような指数形式のいずれかになっていることを確認してください。
- すべてのタイムスタンプおよび日付文字列が DATEFORMAT と TIMEFORMAT の文字列 の仕様に従っていることを確認してください。デフォルトのタイムスタンプ形式は YYYY-MM-DD hh:mm:ss であり、デフォルトの日付形式は YYYY-MM-DD です。
- 個々のデータ型の制約に関する詳細は、「データ型」を参照してください。マルチバイト文字のエラーに関する詳細は、「マルチバイト文字のロードエラー」を参照してください。

Amazon S3 からデータをロードする

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを利用し、Amazon S3 バケット内の単一もしくは複数のファイルとの間で、データの読み取りやロードを並列的に実行しま

す。ファイルが圧縮されている場合は、データを複数のファイルに分割することで、並列処理の長所を最大限に活用できます。(このルールには例外があります。詳細については、「データファイルのロード」を参照してください。) また、テーブルで分散キーを設定することによっても、並列処理の長所を最大化できます。分散キーの詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

データは、ターゲットテーブルの各行に 1 行が対応するようにロードされます。データファイルのフィールドは左から右の順でテーブル列に一致します。データファイルのフィールドは固定幅か文字区切りになります。デフォルトの区切り文字はパイプ (|) です。デフォルトでは、すべてのテーブル列がロードされますが、任意の列のリストをカンマ区切りで指定することもできます。COPY コマンドに指定された列リストに含まれていない列については、デフォルト値がロードされます。詳細については、「デフォルトの列値をロードする」を参照してください。

トピック

- 圧縮および非圧縮のファイルからのデータのロード
- Amazon S3 にファイルをアップロードして COPY で使用する
- COPY コマンドを使用し、Amazon S3 からロードする

圧縮および非圧縮のファイルからのデータのロード

圧縮データをロードする際には、データを、各テーブルに対応させ複数のファイルに分割することをお勧めします。Amazon S3 バケット内の大きなファイルから区切りのある非圧縮データをロードする場合、COPY コマンドは、そのデータのロードに超並列処理 (MPP) とスキャン範囲を使用します。

複数の圧縮されたファイルからのデータのロード

データが圧縮されている場合は、各テーブル用に複数のファイルとして、データを分割することをお勧めします。COPY コマンドを実行すると、複数のファイルから並列でデータをロードすることができます。セットに共通プレフィックスまたはプレフィックスキーを指定するか、マニフェストファイルにファイルのリストを明示的に指定することで、複数のファイルをロードできます。

ファイルの数がクラスターのスライスの数の倍数になるようにデータをファイルに分割します。これにより、Amazon Redshift は各スライス間でデータを均等に分割できます。ノードあたりのスライスの数は、クラスターのノードサイズによって決まります。例えば、各 dc2.large コンピューティングノードには 2 個のスライスがあり、各 dc2.8xlarge コンピューティングノードには 16 個のスライスがあります。各ノードサイズに含まれるスライス数の詳細については、「Amazon Redshift 管理ガイド」の「クラスターおよびノードについて」を参照してください。

ノードはいずれも並列クエリの実行に関与し、スライス全体でできるだけ均等に分散されたデータを処理します。クラスターに dc2.large ノードが 2 つある場合は、データを 4 つのファイルまたは 4 の倍数のファイルに分割できます。Amazon Redshift はワークロードを分割するときにファイルサイズを考慮しません。したがって、圧縮後の各ファイルは、ほぼ同じ (1 MB～1 GB の間の) サイズにする必要があります。

オブジェクトプレフィックスを使ってロード対象のファイルを識別する場合、各ファイルの名前に共通のプレフィックスを付けます。例えば、venue.txt ファイルを下記のように 4 つのファイルに分割します。

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

複数のファイルをバケットのフォルダーに置いている場合、プレフィックスとしてフォルダー名を指定すると、COPY によりフォルダー内のすべてのファイルがロードされます。ロードするファイルが複数のバケットまたはフォルダーに分散している場合は、それらのファイルのリストをマニフェストファイルに明示的に指定します。

マニフェストファイルについて詳しくは、「Example: COPY from Amazon S3 using a manifest」を参照してください。

非圧縮の区切りファイルからのデータの読み込み

区切りのある非圧縮データをロードする際、COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを使用します。Amazon Redshift は、各スライスを自動的に並列処理し、それぞれに応じた範囲のデータを、Amazon S3 バケット内の大きなファイルからロードします。並列ロードを実行するには、ファイルを区切る必要があります。例えば、パイプ区切りなどを使用します。CSV ファイルでは、COPY コマンドでの自動的な並列データロードも利用できます。テーブルに分散キーを設定することによって並列処理を活用することも可能です。分散キーの詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

COPY クエリに ESCAPE、REMOVEQUOTES、および FIXEDWIDTH キーワードのどれかが含まれている場合、自動的な並列データロードはサポートされません。

単一もしくは複数のファイルからのデータは、ターゲットテーブルの各行ごとに 1 行が対応してロードされます。データファイルのフィールドは左から右の順でテーブル列に一致します。データファイルのフィールドは固定幅か文字区切りになります。デフォルトの区切り文字はパイプ (|) で

す。デフォルトでは、すべてのテーブル列がロードされますが、任意の列のリストをカンマ区切りで指定することもできます。テーブルの列が COPY コマンドで指定された列リストに含まれていない場合は、デフォルト値を使用してロードされます。詳細については、「デフォルトの列値をロードする」を参照してください。

Amazon S3 から区切りのある非圧縮なデータをロードする場合は、次の一般的なプロセスに従います。

1. ファイルを Amazon S3 にアップロードします。
2. COPY コマンドを実行し、テーブルをロードします。
3. データが正しくロードされたことを確認します。

COPY コマンドの例については、「COPY の例」を参照してください。Amazon Redshift にロードされたデータに関する情報は、システムテーブル STL_LOAD_COMMITS および STL_LOAD_ERRORS でご確認ください。

これらに含まれるノードとスライスの詳細については、「Amazon Redshift 管理ガイド」の「クラスターおよびノードについて」を参照してください。

Amazon S3 にファイルをアップロードして COPY で使用する

Amazon S3 にテキストファイルをアップロードする場合には、以下のように、取るべきいくつかのアプローチがあります。

- ファイルが大規模で圧縮されている場合は、Amazon Redshift の並列処理を活用するために、そのファイルを分割することをお勧めします。
- また COPY では、大きな圧縮されていないテキスト区切りファイルの場合にデータは自動で分割されるので、これにより並列処理が容易になり、大きなファイルからのデータ配布が効果的に行われます。

データファイルを入れる Amazon S3 バケットを作成し、データファイルをバケットにアップロードします。バケットの作成およびファイルのアップロードの詳細については、Amazon Simple Storage Service ユーザーガイドのAmazon S3 バケットの操作を参照してください。

> **Important**
>
> REGION オプションを使用して Amazon S3 バケットがあるリージョンを指定しない限り、データファイルを保持する Amazon S3 バケットは、クラスターと同じ AWS リージョンに作成する必要があります。

S3 IP 範囲が許可リストに追加されていることを確認します。必要な S3 IP 範囲の詳細については、「ネットワークの隔離」を参照してください。

Amazon S3 コンソールを利用してバケットを作成するときにリージョンを選択するか、Amazon S3 API または CLI を利用してバケットを作成するときにエンドポイントを指定することで、特定のリージョンに Amazon S3 バケットを作成できます。

データのロード後、Amazon S3 に正しいファイルが存在することを確認します。

トピック

- データの整合性の管理
- Amazon S3 への暗号化されたデータのアップロード
- 必要なファイルがバケットにあることを確認する

データの整合性の管理

Amazon S3 では、すべての AWS リージョンにある Amazon S3 バケットに対する COPY、UNLOAD、INSERT (外部テーブル)、CREATE EXTERL TABLE AS、および Amazon Redshift Spectrum オペレーションについて、書き込みとそれに続く読み込みの間での、強力な一貫性が提供されます。さらに、Amazon S3 Select、Amazon S3 アクセスコントロールリスト、Amazon S3 オブジェクトタグ、オブジェクトメタデータ (HEAD オブジェクトなど) での読み込みオペレーションには、強力な整合性があります。データ整合性の詳細については、Amazon Simple Storage Service デベロッパーガイドの Amazon S3 のデータ整合性モデルを参照してください。

Amazon S3 への暗号化されたデータのアップロード

Amazon S3 は、サーバー側の暗号化とクライアント側の暗号化のどちらもサポートします。このトピックでは、サーバー側暗号化とクライアント側暗号化の違いを説明し、Amazon Redshift でクライアント側暗号化を使用するステップを紹介します。サーバー側暗号化は Amazon Redshift に対して透過的です。

サーバー側の暗号化

サーバー側の暗号化は、保管時のデータ暗号化です。つまり、Amazon S3 がデータをアップロードするときにデータを暗号化し、アクセス時にデータを復号化します。COPY コマンドを使用してテーブルをロードする場合には、Amazon S3 にあるサーバー側で暗号化したオブジェクトや暗号化していないオブジェクトからロードする場合との間に差はありません。サーバー側の暗号化の詳細については、Amazon Simple Storage Service ユーザーガイドの[サーバー側の暗号化の使用](#)を参照してください。

クライアント側の暗号化

クライアント側の暗号化の場合、データの暗号化、暗号化キー、および関連ツールを管理するのはクライアントアプリケーションです。クライアント側の暗号化を使って Amazon S3 バケットにデータをアップロードし、その後、COPY コマンド (ENCRYPTED オプション) とプライベート暗号化キーを使ってデータをより安全にロードすることができます。

エンベロープ暗号化を使い、データを暗号化します。エンベロープ暗号化を使うと、アプリケーションですべての暗号化が排他的に処理されます。プライベート暗号化キーと暗号化されていないデータが AWS に送信されることはありません。したがって、暗号化キーを安全に管理することが重要となります。暗号化キーをなくした場合はデータを復号できなくなります。また、AWS から暗号化キーを復元することはできません。エンベロープ暗号化では、非同期キーによるキー管理によりセキュリティを高めながら、高速の同期暗号化のパフォーマンスを実現します。Amazon S3 暗号化クライアントは、データを暗号化するためにワンタイム使用の対称キー (エンベロープ対称キー) を生成します。その後、そのキーはルートキーにより暗号化され、データとともに Amazon S3 に保存されます。Amazon Redshift がロード時にデータにアクセスするとき、暗号化された同期キーが取得され、本物のキーで復号され、その後、データが復号されます。

Amazon Redshift で Amazon S3 クライアント側の暗号化データを操作するには、Amazon Simple Storage Service ユーザーガイドの[クライアント側の暗号化を使用したデータの保護](#)にて説明されているステップに従います。使用する追加の要件はこちらです:

- 対称暗号化 – AWS SDK for Java `AmazonS3EncryptionClient` クラスは、前述の、対称キー暗号化に基づくエンベロープ暗号化を使用します。このクラスを使用して、Amazon S3 クライアントを作成し、クライアント側暗号化データをアップロードします。
- 256 ビット AES ルート対称キー – ルートキーはエンベロープキーを暗号化します。ルートキーは、`AmazonS3EncryptionClient` クラスのインスタンスに渡す必要があります。このキーは、Amazon Redshift にデータをコピーするときに必要になるので、キーを保存する必要があります。

- 暗号化されたエンベロープキーを保存するオブジェクトメタデータ — Amazon S3 のデフォルトでは、エンベロープキーは `AmazonS3EncryptionClient` クラスのオブジェクトメタデータとして保存されます。オブジェクトメタデータとして保存された暗号化されたエンベロープキーは、復号プロセスで使用されます。

> **Note**
>
> 初めて暗号化 API を使用するときに暗号の暗号化エラーメッセージが表示される場合、使用する JDK のバージョンに、暗号化/復号変換に使用する最大キー長を 128 ビットに制限する Java Cryptography Extension (JCE) 管轄ポリシーファイルが含まれている可能性があります。この問題に対処する方法については、Amazon Simple Storage Service ユーザーガイドの「AWS SDK for Java を使用したクライアント側の暗号化の指定」を参照してください。

クライアント側で暗号化したファイルを COPY コマンドを使用して Amazon Redshift テーブルにロードする方法については、暗号化されたデータファイルを Amazon S3 からロードする を参照してください。

例: クライアント側暗号化データのアップロード

AWS SDK for Java を使用してクライアント側の暗号化データをアップロードする例については、Amazon Simple Storage Service ユーザーガイドの「クライアント側の暗号化を使用したデータの保護」を参照してください。

2 つ目のオプションで、データを Amazon Redshift にロードできるように、クライアント側で暗号化を行う際に選択すべき点を示しています。具体的には、オブジェクトメタデータを使用して暗号化されたエンベロープキーを保存する方法と、256 ビット AES マスター対称キーを使用する方法についてです。

ここでは、AWS SDK for Java を使用して 256 ビットの AES 対称ルートキーを作成し、それをファイルに保存するためのコード例を示します。次に、この例ではサンプルデータを S3 暗号化クライアントを使用してクライアント側で最初に暗号化し、オブジェクトを Amazon S3 にアップロードします。この例ではオブジェクトもダウンロードして、データが同じであることを確認します。

必要なファイルがバケットにあることを確認する

Amazon S3 バケットにファイルをアップロードした後に、バケットの内容を一覧表示し、必要なすべてのファイルが揃っており、不必要なファイルがないことを確認します。例えば、バケット

amzn-s3-demo-bucket に「venue.txt.back」という名前のファイルがある場合、そのファイルは、おそらくは意図しなくても、次のコマンドによりロードされます。

```
COPY venue FROM 's3://amzn-s3-demo-bucket/venue' … ;
```

ロードするファイルを厳密に制御したいのであれば、マニフェストファイルを使用し、データファイルをリストに明示的に指定します。マニフェストファイルの使用に関する詳細は、COPY コマンドの copy_from_s3_manifest_file オプションと COPY 例の Example: COPY from Amazon S3 using a manifest を参照してください。

バケット内容のリスト化に関する詳細については、Amazon S3 デベロッパーガイドの「オブジェクトキーのリスト」を参照してください。

COPY コマンドを使用し、Amazon S3 からロードする

COPY コマンドを使い、Amazon S3 のデータファイルからテーブルを並列でロードします。Amazon S3 オブジェクトプレフィックスまたはマニフェストファイルを利用し、ロードするファイルを指定できます。

プレフィックスを使ってロードするファイルを指定するための構文は次のようになります。

```
COPY <table_name> FROM 's3://<bucket_name>/<object_prefix>'
authorization;
```

マニフェストファイルは、ロードするデータファイルのリストを指定する JSON 形式のファイルです。マニフェストファイルを使ってロードするファイルを指定するための構文は次のようになります。

```
COPY <table_name> FROM 's3://<bucket_name>/<manifest_file>'
authorization
MANIFEST;
```

ロードするテーブルはデータベースに存在している必要があります。テーブルの作成に関する詳細は、SQL リファレンスの「CREATE TABLE」を参照してください。

許可の値により、Amazon Redshift が Amazon S3 オブジェクトにアクセスするために必要な AWS の許可が提供されます。必要なアクセス権限については、「COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可」を参照してください。推奨の認証方法は、IAM_ROLE パラメータを指定して、必要なアクセス権限がある IAM ロールの Amazon リソースネーム (ARN) を提供することです。詳細については、「ロールベースアクセスコントロール」を参照してください。

IAM_ROLE パラメータを使用して認証するには、次の構文で示すように、*<aws-account-id>* および *<role-name>* を置き換えます。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

次の例で、IAM ロールを使用した認証を示します。

```
COPY customer
FROM 's3://amzn-s3-demo-bucket/mydata'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

他の認証オプションの詳細については、「認可パラメータ」を参照してください。

テーブルを実際にロードせずにデータを検証する場合、COPY コマンドに NOLOAD オプションを指定します。

次の例では、「venue.txt」という名前のファイルのパイプ区切りデータの最初の数行を示しています。

```
1|Toyota Park|Bridgeview|IL|0
2|Columbus Crew Stadium|Columbus|OH|0
3|RFK Stadium|Washington|DC|0
```

Amazon S3 にファイルをアップロードする前に、ファイルを複数のファイルに分割します。分割されたファイルは COPY コマンドで並列処理を使ってロードされます。ファイルの数はクラスター内のスライスの数の倍数である必要があります。ロードデータファイルを分割して大体同じサイズにし、圧縮後に 1 MB ~ 1 GB になるようにします。詳細については、「圧縮および非圧縮のファイルからのデータのロード」を参照してください。

例えば、venue.txt ファイルを次のように 4 つのファイルに分割します。

```
venue.txt.1
venue.txt.2
venue.txt.3
venue.txt.4
```

次の COPY コマンドは Amazon S3 バケット amzn-s3-demo-bucket でプレフィックスが「venue」になっているデータファイルのパイプ区切りデータを使って VENUE テーブルをロードします。

> **Note**
>
> 次の例の Amazon S3 バケット `amzn-s3-demo-bucket` は存在しません。既存の Amazon S3 バケットの実データを使用した、COPY コマンドのサンプルについては、「サンプルデータをロードする」を参照してください。

```
COPY venue FROM 's3://amzn-s3-demo-bucket/venue'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|';
```

「venue」というキープレフィックスのある Amazon S3 オブジェクトが存在しない場合、ロードは失敗します。

トピック

- マニフェストを使用し、データファイルを指定する
- 圧縮されたデータファイルを Amazon S3 からロードする
- Amazon S3 から固定幅データをロードする
- Amazon S3 からマルチバイトのデータをロードする
- 暗号化されたデータファイルを Amazon S3 からロードする

マニフェストを使用し、データファイルを指定する

COPY コマンドにより必要なすべてのファイルのみがロードされるように、データロードにマニフェストを使用できます。マニフェストを使用し、異なるバケットからファイルをロードしたり、同じプレフィックスを共有しないファイルをロードしたりできます。この場合、COPY コマンドにオブジェクトパスを指定する代わりに、ロードするファイルのリストを JSON 形式のテキストファイルに明示的に指定します。マニフェスト内の URL ではプレフィックスだけでなく、バケットの名前とファイルの完全なオブジェクトパスを指定しておく必要があります。

マニフェストファイルの詳細については、「マニフェストを使用し、データファイルを指定する」を参照してください。

次の例では、バケットが異なり、ファイル名が日付スタンプで始まるファイルをロードする JSON を示しています。

```
{
```

```
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket1/2013-10-05-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/2013-10-05-custdata", "mandatory":true}
  ]
}
```

ファイルが見つからない場合に、オプションの `mandatory` フラグによって COPY コマンドがエラーを返すかどうかを指定します。`mandatory` のデフォルトは `false` です。mandatory 設定と関係なく、どのファイルも見つからない場合、COPY は終了します。

次の例では、前の例にあった「`cust.manifest`」という名前のマニフェストで COPY コマンドを実行しています。

```
COPY customer
FROM 's3://amzn-s3-demo-bucket/cust.manifest'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MANIFEST;
```

UNLOAD で作成されたマニフェストの使用

MANIFEST パラメータを使用して UNLOAD オペレーションで作成したマニフェストには、COPY オペレーションに必要ないキーが含まれている場合があります。例えば、次の `UNLOAD` マニフェストには、Amazon Redshift Spectrum の外部テーブルと `ORC` または `Parquet` ファイル形式でデータファイルをロードするのに必要な `meta` キーが含まれています。`meta` キーには、ファイルの実際のサイズの値 (バイト) の `content_length` キーが含まれます。COPY オペレーションに必要なのは `url` キーとオプションの `mandatory` キーのみです。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/unload/manifest_0000_part_00", "meta":
 { "content_length": 5956875 }},
    {"url":"s3://amzn-s3-demo-bucket/unload/unload/manifest_0001_part_00", "meta":
 { "content_length": 5997091 }}
 ]
}
```

マニフェストファイルについて詳しくは、「Example: COPY from Amazon S3 using a manifest」を参照してください。

圧縮されたデータファイルを Amazon S3 からロードする

gzip、lzop、または bzip2 で圧縮されたデータファイルをロードするには、対応する GZIP、LZOP、または BZIP2 オプションを含めます。

例えば、次のコマンドは lzop で圧縮されたファイルをロードします。

```
COPY customer FROM 's3://amzn-s3-demo-bucket/customer.lzo'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|' LZOP;
```

> **Note**
>
> データファイルを lzop で圧縮して --filter オプションを使用する場合、COPY コマンドはこのオプションをサポートしません。

Amazon S3 から固定幅データをロードする

固定幅データファイルでは、データの各列の長さが統一されています。固定幅データファイルの各フィールドでは、長さと位置がまったく同じです。固定幅データファイルの文字データ (CHAR と VARCHAR) については、幅を統一するために、プレースホルダーとして先行または後続スペースを含める必要があります。整数については、プレースホルダーとして先行ゼロを含める必要があります。固定幅データファイルには列を分割する区切り文字がありません。

固定幅データファイルを既存のテーブルにロードするには、COPY コマンドで FIXEDWIDTH パラメータを使用します。データを正しくロードするには、テーブル仕様が fixedwidth_spec の値に一致する必要があります。

ファイルからテーブルに固定幅データをロードするには、次のコマンドを発行します。

```
COPY table_name FROM 's3://amzn-s3-demo-bucket/prefix'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
FIXEDWIDTH 'fixedwidth_spec';
```

fixedwidth_spec パラメータは、各列の ID と各列の幅がコロンで区切られて指定されている文字列です。**column:width** ペアはカンマで区切られています。ID には数字、文字、またはその 2 つの組み合わせを自由に選択できます。ID とテーブル自体の間には何の関係もありません。そのため、仕様にテーブルと同じ順序で列を含める必要があります。

次の 2 つの例では同じ仕様を示していますが、最初の例では数字の ID を使用し、2 つ目の例では文字列の ID を使用しています。

```
'0:3,1:25,2:12,3:2,4:6'
```

```
'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6'
```

次の例では、前述の仕様を使用して VENUE テーブルにロードするサンプルの固定幅データを示しています。

```

1   Toyota Park              Bridgeview  IL0
2   Columbus Crew Stadium    Columbus    OH0
3   RFK Stadium              Washington  DC0
4   CommunityAmerica Ballpark Kansas City KS0
5   Gillette Stadium         Foxborough  MA68756
```

次の COPY コマンドではこのデータセットが VENUE テーブルにロードされます。

```
COPY venue
FROM 's3://amzn-s3-demo-bucket/data/venue_fw.txt'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
FIXEDWIDTH 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

Amazon S3 からマルチバイトのデータをロードする

データに ASCII 以外のマルチバイト文字 (漢字やキリル文字) が含まれる場合、データを VARCHAR 列にロードする必要があります。VARCHAR データ型は 4 バイトの UTF-8 文字をサポートしますが、CHAR データ型はシングルバイトの ASCII 文字のみを受け取ります。5 バイト以上の文字を Amazon Redshift テーブルにロードすることはできません。CHAR と VARCHAR に関する詳細は、「データ型」を参照してください。

入力ファイルで使用されるエンコーディングを確認するには、Linux *file* コマンドを使用します:

```
$ file ordersdata.txt
ordersdata.txt: ASCII English text
$ file uni_ordersdata.dat
uni_ordersdata.dat: UTF-8 Unicode text
```

暗号化されたデータファイルを Amazon S3 からロードする

COPY コマンドを使い、サーバー側の暗号化、クライアント側の暗号化、またはその両方を使用して Amazon S3 にアップロードされたデータファイルをロードすることができます。

COPY コマンドは、次のタイプの Amazon S3 の暗号化をサポートします。

- Amazon S3 で管理されたキーを使用したサーバー側の暗号化 (SSE-S3)
- AWS KMS keys (SSE-KMS) によるサーバー側の暗号化
- クライアント側対称ルートキーを使用したクライアント側の暗号化

COPY コマンドは、次のタイプの Amazon S3 の暗号化をサポートしません。

- お客様が用意したキーを使用したサーバー側の暗号化 (SSE-C)
- AWS KMS key を使用したクライアント側の暗号化
- お客様が提供する非対称ルートキーを使用したクライアント側の暗号化

Amazon S3 暗号化の詳細については、Amazon Simple Storage Service ユーザーガイドのサーバー側の暗号化を使用したデータの保護およびクライアント側の暗号化を使用したデータの保護を参照してください。

UNLOAD コマンドは SSE-S3 を使用して自動的にファイルを暗号化します。また、カスタマー管理型の対象キーによる SSE-KMS またはクライアント側の暗号化を使用してアンロードすることもできます。詳細については、「暗号化されたデータファイルをアンロードする」を参照してください。

COPY コマンドは SSE-S3 と SSE-KMS を使用して暗号化されたファイルを自動的に認識してロードします。ENCRYPTED オプションを使用し、キーの値を指定することで、クライアント側対称ルートキーで暗号化されたファイルをロードできます。詳細については、「Amazon S3 への暗号化されたデータのアップロード」を参照してください。

クライアント側で暗号化されたデータファイルをロードするには、MASTER_SYMMETRIC_KEY パラメータを使用しながらルートキーの値を指定し、そこに ENCRYPTED オプションを含めます。

```
COPY customer FROM 's3://amzn-s3-demo-bucket/encrypted/customer'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MASTER_SYMMETRIC_KEY '<root_key>'
ENCRYPTED
DELIMITER '|';
```

gzip、lzop、または bzip2 で圧縮された暗号化データファイルをロードするには、ルートキーの値とともに GZIP、LZOP、または BZIP2 オプションを含め、さらに ENCRYPTED オプションを含めます。

```
COPY customer FROM 's3://amzn-s3-demo-bucket/encrypted/customer'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
MASTER_SYMMETRIC_KEY '<root_key>'
ENCRYPTED
DELIMITER '|'
GZIP;
```

Amazon EMR からのデータのロード

COPY コマンドを使用することで、クラスターの Hadoop Distributed File System (HDFS) に、固定幅ファイル、文字区切りファイル、CSV ファイル、または JSON 形式ファイルでテキストファイルを書き込むように設定された Amazon EMR クラスターから、データを並列にロードできます。

Amazon EMR からデータをロードするプロセス

このセクションでは、Amazon EMR クラスターからデータをロードする手順について説明します。以下のセクションでは、各ステップで必要な操作の詳細を説明します。

- ステップ 1: IAM のアクセス許可を設定する

  Amazon EMR クラスターを作成して Amazon Redshift の COPY コマンドを実行するユーザーには、そのための許可が必要です。
- ステップ 2: Amazon EMR クラスターを作成する

  テキストファイルを Hadoop Distributed File System（HDFS）に出力するようにクラスターの設定を変更します。Amazon EMR クラスター ID およびそのクラスターのメインの公開 DNS (クラスターをホストする Amazon EC2 インスタンスのエンドポイント) が必要になります。
- ステップ 3: Amazon Redshift クラスターの公開キーおよびクラスターノード IP アドレスを取得する

  公開キーは、Amazon Redshift クラスターノードがホストへの SSH 接続を確立するために使用されます。ホストのセキュリティグループに各クラスターノードの IP アドレスを設定し、その IP アドレスで Amazon Redshift クラスターからアクセスできるようにします。
- ステップ 4: 各 Amazon EC2 ホストの承認されたキーファイルに Amazon Redshift クラスターの公開キーを追加する

ホストが Amazon Redshift クラスターを認識し、SSH 接続を許可するように、ホストの認可されたキーファイルに Amazon Redshift クラスターの公開キーを追加します。

- [ステップ 5: Amazon Redshift クラスターの IP アドレスすべてを許可するようにホストを設定する]

  Amazon EMR インスタンスのセキュリティグループを変更して、Amazon Redshift の IP アドレスを許可する入力ルールを追加します。

- [ステップ 6: COPY コマンドを実行してデータをロードする]

  Amazon Redshift データベースから COPY コマンドを実行して、Amazon Redshift テーブルにデータをロードします。

ステップ 1: IAM のアクセス許可を設定する

Amazon EMR クラスターを作成して Amazon Redshift の COPY コマンドを実行するユーザーには、そのための許可が必要です。

IAM のアクセス許可を設定するには

1. Amazon EMR クラスターを作成するユーザーに以下のアクセス許可を追加します。

```
ec2:DescribeSecurityGroups
ec2:RevokeSecurityGroupIngress
ec2:AuthorizeSecurityGroupIngress
redshift:DescribeClusters
```

2. COPY コマンドを実行する IAM ロールまたはユーザーに以下のアクセス許可を追加します。

```
elasticmapreduce:ListInstances
```

3. Amazon EMR クラスターの IAM ロールに次のアクセス許可を追加します。

```
redshift:DescribeClusters
```

ステップ 2: Amazon EMR クラスターを作成する

COPY コマンドでは、Amazon EMR の Hadoop Distributed File System (HDFS) のファイルからデータをロードします。Amazon EMR クラスターを作成する場合には、クラスターの HDFS にデータファイルを出力するようにクラスターを設定する必要があります。

Amazon EMR クラスターを作成するには

1. Amazon Redshift クラスターと同じ AWS リージョンに Amazon EMR クラスターを作成します。

   Amazon Redshift クラスターが VPC にある場合、Amazon EMR クラスターも同じ VPC グループにある必要があります。Amazon Redshift クラスターで EC2-Classic モードを使用する (つまり、そのクラスターが VPC にない) 場合は、Amazon EMR クラスターでも EC2 Classic モードを使用する必要があります。詳細については、「Amazon Redshift 管理ガイド」の「仮想プライベートクラウド (VPC) でクラスターを管理する」を参照してください。
2. クラスターの HDFS にデータファイルを出力するようにクラスターを設定します。HDFS ファイル名にアスタリスク (*) と疑問符 (?) は使用できません。

   > **Important**
   >
   > ファイル名にアスタリスク (*) と疑問符 (?) は使用できません。
3. COPY コマンドの実行中もクラスターを継続して使用できるように、Amazon EMR クラスター設定の [Auto-terminate] (自動終了) オプションで [No] (いいえ) を指定します。

   > **Important**
   >
   > COPY が完了する前にデータ ファイルのいずれかが変更または削除されると、予期しない結果を招いたり、COPY 操作が失敗したりする可能性があります。
4. クラスター ID およびメインの公開 DNS (クラスターをホストする Amazon EC2 インスタンスのエンドポイント) を書き留めておいてください。この情報は、後のステップで使用します。

ステップ 3: Amazon Redshift クラスターの公開キーおよびクラスターノード IP アドレスを取得する

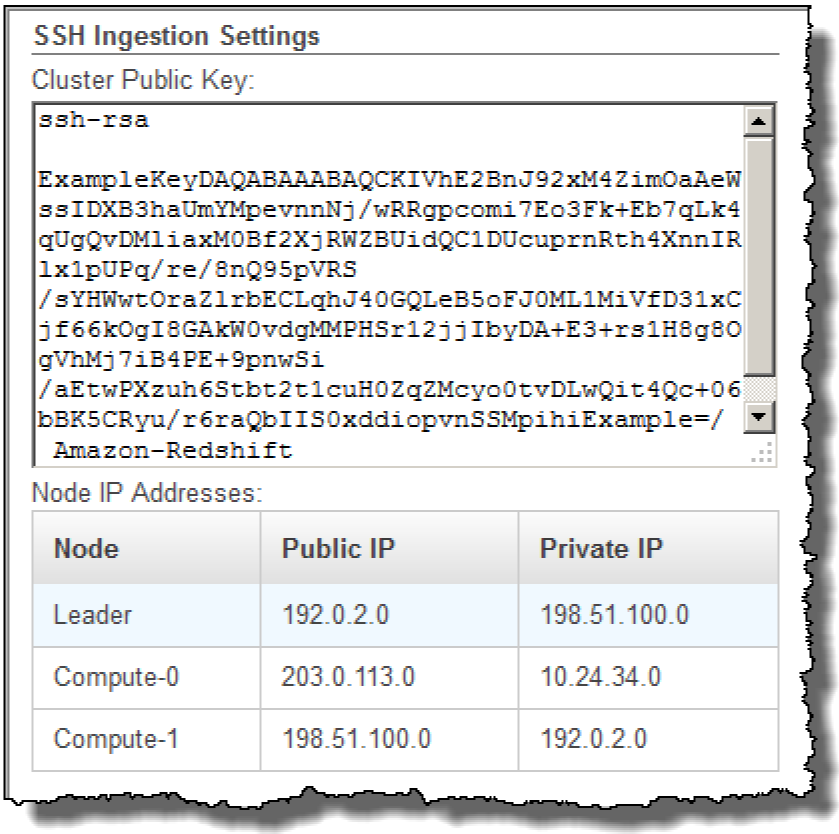
ホストのセキュリティグループに各クラスターノードの IP アドレスを設定し、その IP アドレスで Amazon Redshift クラスターからアクセスできるようにします。

コンソールを使用して Amazon Redshift クラスター公開キーとクラスターのクラスターノード IP アドレスを取得する方法は、以下のとおりです。

1. Amazon Redshift マネジメントコンソールにアクセスします。

2. ナビゲーションペインで [Clusters] (クラスター) リンクを選択します。
3. リストからクラスターを選択します。
4. [SSH 取り込み設定] グループを探します。

   [クラスターパブリックキー] と [ノード IP アドレス] の内容を書き留めておきます。この 2 つは、後のステップで使用します。



   このプライベート IP アドレスは、ステップ 3 で Amazon Redshift からの接続を許可するように Amazon EC2 ホストを設定するために使用します。

Amazon Redshift CLI を使用してクラスター公開キーとクラスターノード の IP アドレスを取得するには、describe-clusters コマンドを実行します。例:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

応答には、以下のような ClusterPublicKey 値とプライベートおよびパブリック IP アドレスのリストが含まれます。

```
{
```

```
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
            "ClusterNodes": [
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

Amazon Redshift API を使用してクラスターの公開キーとクラスターノード IP アドレスを取得するには、`DescribeClusters` アクションを使用します。詳細については、Amazon Redshift CLI ガイドの describe-clusters または Amazon Redshift API ガイドの DescribeClusters を参照してください。

ステップ 4: 各 Amazon EC2 ホストの承認されたキーファイルに Amazon Redshift クラスターの公開キーを追加する

Amazon EMR クラスターノードすべてについて、各ホストの承認されたキーファイルにクラスターの公開キーを追加し、ホストが Amazon Redshift を認識して SSH 接続を許可できるようにします。

Amazon Redshift クラスターの公開キーをホストの認可されたキーファイルに追加するには

1. SSH 接続を使用してホストにアクセスします。

   SSH を使用したインスタンスへの接続については、Amazon EC2 ユーザーガイドの インスタンスへの接続を参照してください。
2. コンソールまたは CLI 応答のテキストから Amazon Redshift の公開キーをコピーします。
3. パブリックキーの内容をコピーして、ホストの `/home/<ssh_username>/.ssh/authorized_keys` ファイルに貼り付けます。プレフィックス "`ssh-rsa`" やサフィックス "`Amazon-Redshift`" も含めた完全な文字列を入力してください。次に例を示します。

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-Redshift
```

ステップ 5: Amazon Redshift クラスターの IP アドレスすべてを許可するようにホストを設定する

ホストインスタンスへのインバウンドトラフィックを許可するには、セキュリティグループを編集して、Amazon Redshift クラスターノードごとに 1 つのインバウンドルールを追加します。[タイプ] として、ポート 22 での TCP プロトコルを使用した SSH を選択します。[Source] (ソース) としては、ステップ 3: Amazon Redshift クラスターの公開キーおよびクラスターノード IP アドレスを取得する で取得した Amazon Redshift クラスターノードのプライベート IP アドレスを入力します。Amazon EC2 セキュリティグループへのルール追加の詳細については、Amazon EC2 ユーザーガイドからインスタンスのインバウンドトラフィックの認可を参照してください。

ステップ 6: COPY コマンドを実行してデータをロードする

COPY コマンドを実行して Amazon EMR クラスターに接続し、Amazon Redshift テーブルにデータをロードします。Amazon EMR クラスターは、COPY コマンドが完了するまで稼動している必要があります。例えば、クラスターに対して自動終了は設定しないようにしてください。

> **Important**
>
> COPY が完了する前にデータ ファイルのいずれかが変更または削除されると、予期しない結果を招いたり、COPY 操作が失敗したりする可能性があります。

COPY コマンドでは、Amazon EMR クラスター ID と、HDFS のファイルパスおよびファイル名を指定します。

```
COPY sales
FROM 'emr://myemrclusterid/myoutput/part*' CREDENTIALS
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

ファイル名の引数にはワイルドカード文字としてアスタリスク (*) および疑問符 (?) を使用できます。たとえば、`part*` であれば、`part-0000`、`part-0001` などのファイルがロードされます。COPY コマンドでフォルダー名のみを指定した場合には、フォルダー内のすべてのファイルがロードされます。

> **Important**
>
> ワイルドカード文字を使用する場合や、フォルダー名のみを指定する場合には、フォルダーを確認して不要なファイルがロードされることのないようにしてください。不要なファイルがロードされると、COPY コマンドが失敗します。例えば、一部のプロセスでは出力フォルダにログファイルが書き込まれることがあります。

リモートホストからデータをロードする

COPY コマンドでは、Amazon EC2 インスタンスもしくは他のコンピュータなど、1 つ以上のリモートホストから並列的にデータをロードすることができます。COPY では SSH を使用してリモートホストに接続し、そのホスト上でコマンドを実行しテキスト出力を生成します。

リモートホストになることができるのは、Amazon EC2 の Linux インスタンスか、SSH 接続を許可するように設定されている Unix コンピュータまたは Linux コンピュータです。このガイドでは、リモートホストが Amazon EC2 インスタンスであることを前提としています。これ以外のコンピュータの場合に手順が異なるときは、その都度説明します。

Amazon Redshift は複数のホストに接続でき、各ホストに対して複数の SSH 接続を開くことができます。Amazon Redshift は、各接続を介して一意のコマンドを送信し、ホストの標準出力にテキスト出力を生成します。その後、Amazon Redshift はテキストファイルと同じように読み込みます。

開始する前に

開始する前に、以下があることを確認してください。

- 1 台以上のホストマシン (SSH を使用して接続できる Amazon EC2 インスタンスなど)。

- ホストのデータソース。

  テキスト出力を生成するために Amazon Redshift クラスターがホストで実行するコマンドを指定します。COPY コマンドは、クラスターがホストに接続するとコマンドを実行し、ホストの標準的な出力先からテキストを読み込み、Amazon Redshift テーブルにデータを並列的にロードします。テキストの出力は、COPY コマンドが取り込むことができる形式であることが必要です。詳細については、「入力データを準備する」を参照してください。
- コンピュータからホストへのアクセス権。

  Amazon EC2 インスタンスでは、ホストにアクセスする際に SSH 接続を使用します。このため、ホストにアクセスし、ホストの承認されたキーファイルに Amazon Redshift クラスターの公開キーを追加する必要があります。
- 実行中の Amazon Redshift クラスター。

  クラスターを起動する方法については、Amazon Redshift 入門ガイドを参照してください。

データをロードする手順

このセクションでは、リモートホストからデータをロードする手順を説明します。以下のセクションでは、各ステップで必要な操作の詳細を説明します。

- ステップ 1: クラスターパブリックキーおよびクラスターノード IP アドレスを取得する

  公開キーは、Amazon Redshift クラスターノードがリモートホストへの SSH 接続を確立するために使用されます。各クラスターノードの IP アドレスを使用して、その IP アドレスによる Amazon Redshift クラスターからのアクセスを許可するように、ホストのセキュリティグループまたはファイアウォールを設定します。
- ステップ 2: ホストの承認されたキーファイルに Amazon Redshift クラスターの公開キーを追加する

  ホストが Amazon Redshift クラスターを認識し、SSH 接続を許可するように、ホストの認可されたキーファイルに Amazon Redshift クラスターの公開キーを追加します。
- ステップ 3: Amazon Redshift クラスターの IP アドレスすべてを許可するようにホストを設定する

  Amazon EC2 の場合、インスタンスのセキュリティグループを変更して、Amazon Redshift の IP アドレスを許可する入力ルールを追加します。他のホストの場合には、Amazon Redshift ノードがリモートホストに対して SSH 接続を確立できるようにファイアウォールを設定します。
- ステップ 4: ホストのパブリックキーを取得する

Amazon Redshift がホストの識別に公開キーを使用するように指定することもできます。この場合、パブリックキーを探してマニフェストファイルにテキストをコピーする必要があります。

- ステップ 5: マニフェストファイルを作成する

  マニフェストは、Amazon Redshift がホストに接続し、データを取得する際に必要になる詳細情報が記載された JSON 形式のテキストファイルです。

- ステップ 6: Amazon S3 バケットにマニフェストファイルをアップロードする

  Amazon Redshift はマニフェストを読み込み、その情報を使ってリモートホストに接続します。Amazon S3 バケットが Amazon Redshift クラスターと同じリージョンに存在しない場合は、REGION オプションを使用して、データがあるリージョンを指定する必要があります。

- ステップ 7: COPY コマンドを実行してデータをロードする

  Amazon Redshift データベースから COPY コマンドを実行して、Amazon Redshift テーブルにデータをロードします。

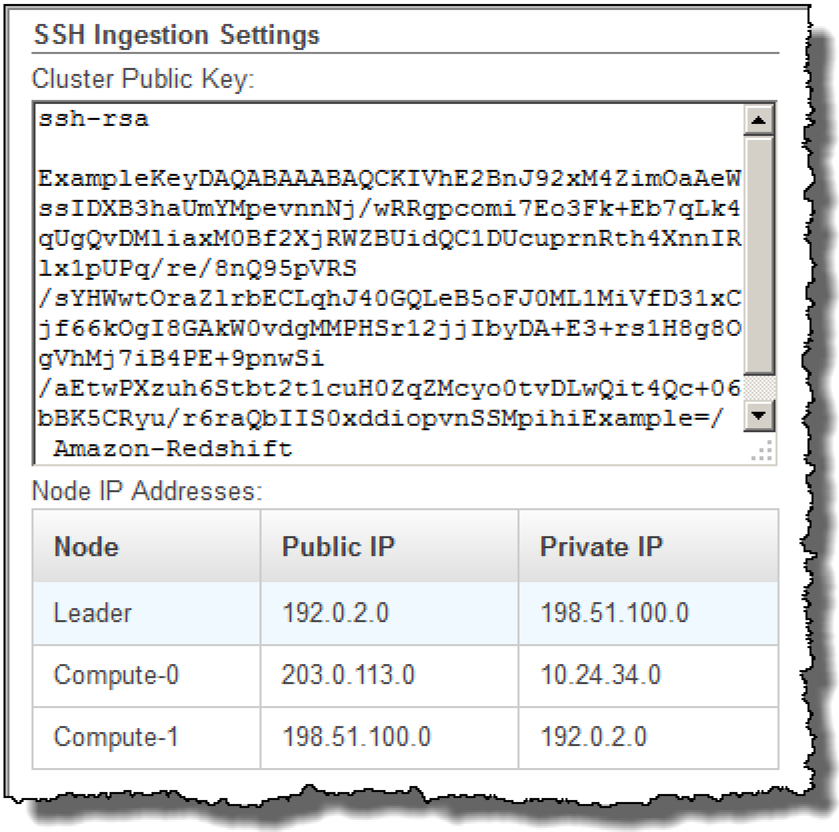
ステップ 1: クラスターパブリックキーおよびクラスターノード IP アドレスを取得する

ホストのセキュリティグループに各クラスターノードの IP アドレスを設定し、その IP アドレスで Amazon Redshift クラスターからアクセスできるようにします。

コンソールを使用してクラスターパブリックキーとクラスターのクラスターノード IP アドレスを取得する方法は、以下のとおりです。

1. Amazon Redshift マネジメントコンソールにアクセスします。
2. ナビゲーションペインで [Clusters] (クラスター) リンクを選択します。
3. リストからクラスターを選択します。
4. [SSH 取り込み設定] グループを探します。

   [クラスターパブリックキー] と [ノード IP アドレス] の内容を書き留めておきます。この 2 つは、後のステップで使用します。



この IP アドレスは、ステップ 3 で Amazon Redshift からの接続を許可するようにホストを設定するために使用します。接続するホストの種類や、ホストが VPC であるかに応じて、パブリック IP アドレスとプライベート IP アドレスのどちらかを使用します。

Amazon Redshift CLI を使用してクラスター公開キーとクラスターノード の IP アドレスを取得するには、describe-clusters コマンドを実行します。

例:

```
aws redshift describe-clusters --cluster-identifier <cluster-identifier>
```

応答には、以下のような ClusterPublicKey 値とプライベートおよびパブリック IP アドレスのリストが含まれます。

```
{
    "Clusters": [
        {
            "VpcSecurityGroups": [],
            "ClusterStatus": "available",
```

```
            "ClusterNodes": [
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "LEADER",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-0",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                },
                {
                    "PrivateIPAddress": "10.nnn.nnn.nnn",
                    "NodeRole": "COMPUTE-1",
                    "PublicIPAddress": "10.nnn.nnn.nnn"
                }
            ],
            "AutomatedSnapshotRetentionPeriod": 1,
            "PreferredMaintenanceWindow": "wed:05:30-wed:06:00",
            "AvailabilityZone": "us-east-1a",
            "NodeType": "dc2.large",
            "ClusterPublicKey": "ssh-rsa AAAABexamplepublickey...Y3TAl Amazon-
Redshift",
             ...
             ...
}
```

Amazon Redshift API を使用してクラスターの公開キーとクラスターノード IP アドレスを取得するには、DescribeClusters アクションを使用します。詳細については、Amazon Redshift CLI ガイドの describe-clusters または Amazon Redshift API ガイドの DescribeClusters を参照してください。

ステップ 2: ホストの承認されたキーファイルに Amazon Redshift クラスターの公開キーを追加する

ホストが Amazon Redshift を認識して SSH 接続を許可するように、各ホストの承認されたキーファイルにクラスターの公開キーを追加します。

Amazon Redshift クラスターの公開キーをホストの認可されたキーファイルに追加するには

1. SSH 接続を使用してホストにアクセスします。

SSH を使用したインスタンスへの接続については、Amazon EC2 ユーザーガイドの インスタンスへの接続を参照してください。

2. コンソールまたは CLI 応答のテキストから Amazon Redshift の公開キーをコピーします。
3. パブリックキーの内容をコピーして、リモートホストの `/home/<ssh_username>/.ssh/authorized_keys` ファイルに貼り付けます。`<ssh_username>` は、マニフェストファイルの [username] フィールドの値と一致している必要があります。プレフィックス "`ssh-rsa`" やサフィックス "`Amazon-Redshift`" も含めた完全な文字列を入力してください。次に例を示します。

```
ssh-rsa AAAACTP3isxgGzVWoIWpbVvRCOzYdVifMrh… uA70BnMHCaMiRdmvsDOedZDOedZ Amazon-
Redshift
```

ステップ 3: Amazon Redshift クラスターの IP アドレスすべてを許可するようにホストを設定する

Amazon EC2 インスタンスまたは Amazon EMR クラスターを使用する場合は、ホストのセキュリティグループにインバウンドルールを追加して、各 Amazon Redshift クラスターノードからのトラフィックを許可します。[タイプ] として、ポート 22 での TCP プロトコルを使用した SSH を選択します。[Source (ソース)] には、ステップ 1: クラスターパブリックキーおよびクラスターノード IP アドレスを取得する で取得した Amazon Redshift クラスターノード IP アドレスを入力します。Amazon EC2 セキュリティグループへのルール追加の詳細については、Amazon EC2 ユーザーガイドからインスタンスのインバウンドトラフィックの認可を参照してください。

プライベート IP アドレスを使用する局面は、以下のとおりです。

- Virtual Private Cloud (VPC) に存在しない Amazon Redshift クラスターと Amazon EC2 - Classic インスタンスがいずれも同じ AWS リージョンにある場合。
- VPC に存在する Amazon Redshift クラスターと Amazon EC2 - VPC インスタンスがいずれも同じ AWS リージョンにあり、かつ同じ VPC に存在する場合。

これ以外の場合には、パブリック IP アドレスを使用します。

VPC での Amazon Redshift の使用についての詳細は、「Amazon Redshift 管理ガイド」の「仮想プライベートクラウド (VPC) でクラスターを管理する」を参照してください。

ステップ 4: ホストのパブリックキーを取得する

Amazon Redshift がホストを識別できるように、マニフェストファイルでホストの公開キーを指定することもできます。COPY コマンドではホストのパブリックキーは必須でないものの、セキュリティの観点から「中間者 (MITM) 」攻撃の予防のため、パブリックキーを使用することを強くお勧めしています。

ホストのパブリックキーは、以下の場所にあります。ここで `<ssh_host_rsa_key_name>` は、ホストのパブリックキーのユニーク名です。

```
:  /etc/ssh/<ssh_host_rsa_key_name>.pub
```

> **Note**
>
> Amazon Redshift は RSA キーのみをサポートしています。DSA キーはサポートしていません。

パブリックキーのテキストは、ステップ 5 でマニフェストファイルを作成するときにマニフェストファイルのエントリにある [Public Key] フィールドに貼り付けることになります。

ステップ 5: マニフェストファイルを作成する

COPY コマンドは、SSH を使用して複数のホストに接続できるだけでなく、各ホストに対して複数の SSH 接続を作成できます。COPY はそれぞれのホスト接続を介してコマンドを実行し、コマンドからの出力を並列的にテーブルにロードします。マニフェストファイルは、Amazon Redshift がホストに接続する際に使用する JSON 形式のテキストファイルです。マニフェストファイルにより、SSH ホストのエンドポイントと、Amazon Redshift にデータを返すためにそのホストで実行されるコマンドを指定します。このほか、ホストのパブリックキー、ログインユーザー名、および各エントリの必須フラグを記載することもできます。

マニフェストファイルをローカルコンピュータで作成します。後の手順で、Amazon S3 にファイルをアップロードします。

マニフェスト ファイルの形式は以下のとおりです。

```
{
   "entries": [
     {"endpoint":"<ssh_endpoint_or_IP>",
```

```
        "command": "<remote_command>",
        "mandatory":true,
        "publickey": "<public_key>",
        "username": "<host_user_name>"},
      {"endpoint":"<ssh_endpoint_or_IP>",
        "command": "<remote_command>",
        "mandatory":true,
        "publickey": "<public_key>",
        "username": "host_user_name"}
    ]
}
```

マニフェストファイルには、SSH 接続ごとに 1 つずつ "entries" 構造が含まれます。各エントリは、1 つの SSH 接続を表します。単一のホストに対して接続を複数作成することも、複数のホストに対して複数の接続を作成することもできます。二重引用符は、フィールド名と値のどちらにも必要です。必須フィールドのなかで二重引用符を必要としない値は、**`true`** または **`false`** のブール値のみです。

以下に、マニフェストファイルのフィールドについて説明します。

endpoint

ホストの URL アドレスまたは IP アドレス。たとえば、"`ec2-111-222-333.compute-1.amazonaws.com`"、"`22.33.44.56`" などです。

コマンド

テキストまたはバイナリ (gzip、lzop、または bzip2) の出力を生成する際にホストが実行するコマンド。コマンドは、ユーザー "host_user_name" が実行権限を持つコマンドであれば、どれでも指定できます。ファイルを印刷するなどのシンプルなコマンドでも、データベースにクエリを実行したり、スクリプトを実行したりするコマンドでもかまいません。出力 (テキストファイル、gzip バイナリファイル、lzop バイナリファイル、または bzip2 バイナリファイル) は、Amazon Redshift の COPY コマンドが取り込める形式にする必要があります。詳細については、「入力データを準備する」を参照してください。

publickey

(オプション) ホストのパブリックキー。公開キーが指定されている場合、Amazon Redshift は公開キーを使用してホストを特定します。公開キーが指定されていなければ、Amazon Redshift がホストの特定を試みることはありません。例えば、リモートホストのパブリックキーが `ssh-rsa AbcCbaxxx…xxxDHKJ root@amazon.com` であれば、publickey フィールドには `AbcCbaxxx…xxxDHKJ` と入力してください。

mandatory

(オプション) 接続ができなかった場合に COPY コマンドを失敗とするかどうかを示すものです。デフォルト: `false`。Amazon Redshift が接続を 1 つも正常に確立できなかった場合に、COPY コマンドが失敗になります。

username

(オプション) ホストシステムにログオンし、リモートコマンドを実行する際に使用するユーザーネーム。ユーザーログイン名は、ステップ 2 でホストの認可されたキーファイルにパブリックキーを追加するときに使用したログイン名と同じものにする必要があります。デフォルトのユーザー名は「redshift」です。

以下の例に、同じホストに対して 4 つの接続を確立し、接続ごとに異なるコマンドを実行するマニフェストの全容を示します。

```
{
  "entries": [
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata1.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata2.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata3.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"},
       {"endpoint":"ec2-184-72-204-112.compute-1.amazonaws.com",
          "command": "cat loaddata4.txt",
          "mandatory":true,
          "publickey": "ec2publickeyportionoftheec2keypair",
          "username": "ec2-user"}
     ]
}
```

ステップ 6: Amazon S3 バケットにマニフェストファイルをアップロードする

Amazon S3 バケットにマニフェストファイルをアップロードします。Amazon S3 バケットが Amazon Redshift クラスターと同じ AWS リージョンに存在しない場合は、REGION オプションを使用して、マニフェストがある AWS リージョンを指定する必要があります。Amazon S3 バケットの作成およびファイルのアップロードの詳細については、Amazon Simple Storage Service のユーザーガイドを参照してください。

ステップ 7: COPY コマンドを実行してデータをロードする

COPY コマンドを実行してホストに接続し、Amazon Redshift テーブルにデータをロードします。COPY コマンドでは、マニフェストファイルに Amazon S3 オブジェクトのパスを明示するとともに、SSH オプションを使用します。例:

```
COPY sales
FROM 's3://amzn-s3-demo-bucket/ssh_manifest'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|'
SSH;
```

> **Note**
>
> 自動圧縮を使用する場合には、COPY コマンドでデータの読み取りが 2 回実行されます。つまりリモートコマンドが 2 回実行されることになります。初回の読み取りは圧縮の分析用サンプルを提供するためのものであり、実際にデータがロードされるのは 2 回目の読み取りです。リモートコマンドを 2 回実行することによる影響が問題になるようであれば、自動圧縮は無効にする必要があります。自動圧縮を無効にするには、COMPUPDATE オプションを OFF に設定して COPY コマンドを実行します。詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

Amazon DynamoDB テーブルからのデータのロード

COPY コマンドで、1 つの Amazon DynamoDB テーブルのデータを使用してテーブルをロードすることができます。

> **Important**
>
> REGION オプションを使用して Amazon DynamoDB テーブルがある AWS リージョンを指定しない限り、データを提供する Amazon DynamoDB テーブルは、クラスターと同じ AWS リージョンに作成される必要があります。

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを使用して、Amazon DynamoDB テーブルからデータを並列でロードします。Amazon Redshift テーブルに分散方式を設定すれば、並列処理を最大限に活用できます。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

> **Important**
>
> COPY コマンドで Amazon DynamoDB テーブルからデータを読み込むとき、結果として起こるデータ転送はそのテーブルにプロビジョンドスループットの一部になります。

プロビジョニングされた読み込みスループットの過度の消費を避けるために、本稼働環境にある Amazon DynamoDB テーブルからはデータをロードしないことをお勧めします。本稼働テーブルからデータをロードする場合、プロビジョニングされ、使用されていないスループットの平均パーセントよりかなり低く READRATIO オプションを設定することをお勧めします。READRATIO を低く設定すると、スロットルの問題が最小限に抑えられます。Amazon DynamoDB テーブルにプロビジョンドスループット全体を使用するには、READRATIO を 100 に設定します。

COPY コマンドは以下のルールを使用し、DynamoDB テーブルから取得された項目の属性名と既存の Amazon Redshift テーブルの列名を照合します。

- Amazon Redshift テーブルの列と Amazon DynamoDB 項目の属性が大文字と小文字を区別せずに照合されます。DynamoDB テーブルの項目に、Price と PRICE のように、大文字/小文字だけが異なる複数の属性が含まれる場合、COPY コマンドは失敗します。
- Amazon Redshift テーブルの属性と一致しない Amazon DynamoDB テーブル列は、COPY コマンドの EMPTYASNULL オプションで指定された値に基づき、NULL または空としてロードされます。
- Amazon Redshift テーブルの列に一致しない Amazon DynamoDB 属性は破棄されます。属性は照合の前に読み込まれます。そのため、破棄された属性もそのテーブルにプロビジョニングされたスループットの一部を消費します。

- データ型がスカラー STRING と NUMBER の Amazon DynamoDB 属性のみがサポートされます。Amazon DynamoDB BINARY および SET データ型はサポートされません。COPY コマンドがサポートされないデータ型を持つ属性をロードしようとすると失敗します。属性が Amazon Redshift テーブル列に一致しない場合、COPY はロードを試行せず、エラーを発行しません。

COPY コマンドは次の構文を使用し、Amazon DynamoDB テーブルからデータをロードします。

```
COPY <redshift_tablename> FROM 'dynamodb://<dynamodb_table_name>'
authorization
readratio '<integer>';
```

authorization の値は、Amazon DynamoDB テーブルにアクセスするために必要な AWS の認証情報です。これらの認証情報がユーザーと対応する場合、このユーザーは、ロードする Amazon DynamoDB テーブルに SCAN と DESCRIBE を実行するアクセス許可が必要です。

authorization の値は、クラスターが Amazon DynamoDB テーブルにアクセスする際に必要な AWS 認証を提供します。許可には、ロードする Amazon DynamoDB テーブルに対する SCAN および DESCRIBE が含まれている必要があります。必要なアクセス許可の詳細については、「COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可」を参照してください。推奨の認証方法は、IAM_ROLE パラメータを指定して、必要なアクセス権限がある IAM ロールの Amazon リソースネーム (ARN) を提供することです。詳細については、「ロールベースアクセスコントロール」を参照してください。

IAM_ROLE パラメータを使用して認証するには、次の構文で示すように、*<aws-account-id>* および *<role-name>* を置き換えます。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

次の例で、IAM ロールを使用した認証を示します。

```
COPY favoritemovies
FROM 'dynamodb://ProductCatalog'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

他の認証オプションの詳細については、「認可パラメータ」を参照してください。

テーブルを実際にロードせずにデータを検証する場合、COPY コマンドに NOLOAD オプションを指定します。

次の例では、「my-favorite-movies-table」という名前の DynamoDB テーブルから FAVORITEMOVIES テーブルにデータをロードします。読み取りアクティビティでは、プロビジョニングされたスループットの最大 50% が消費されます。

```
COPY favoritemovies FROM 'dynamodb://my-favorite-movies-table'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
READRATIO 50;
```

スループットを最大化するために、COPY コマンドはクラスターのコンピューティングノード全体で並列で Amazon DynamoDB テーブルからデータをロードします。

自動圧縮のあるプロビジョニングされたスループット

デフォルトでは、圧縮エンコーディングなしで空のターゲットテーブルを指定したとき、COPY コマンドは自動圧縮を適用します。自動圧縮分析は、最初に Amazon DynamoDB テーブルから大量の行をサンプリングします。サンプルサイズは、COMPROWS パラメータの値に基づきます。デフォルトはスライスごとに 100,000 行です。

サンプリングの後、サンプル行は破棄され、テーブル全体がロードされます。結果として、多くの行が 2 回読み取られます。自動圧縮の仕組みについては、「自動圧縮ありでテーブルをロードする」を参照してください。

> **Important**
>
> COPY コマンドで Amazon DynamoDB テーブルからサンプリングで使われる行を含むデータを読み込むと、結果として起こるデータ転送はそのテーブルにプロビジョンドスループットの一部になります。

Amazon DynamoDB からのマルチバイトデータのロード

データに ASCII 以外のマルチバイト文字 (漢字やキリル文字) が含まれる場合、データを VARCHAR 列にロードする必要があります。VARCHAR データ型は 4 バイトの UTF-8 文字をサポートしますが、CHAR データ型はシングルバイトの ASCII 文字のみを受け取ります。5 バイト以上の文字を Amazon Redshift テーブルにロードすることはできません。CHAR と VARCHAR に関する詳細は、「データ型」を参照してください。

データが正しくロードされたことを確認する

ロード操作が完了したら、STL_LOAD_COMMITS システムテーブルにクエリし、ファイルが予定どおりロードされたことを確認します。ロードに問題が発生した場合にトランザクション全体をロールバックできるように、COPY コマンドとロードの検証は同じトランザクションで実行します。

次のクエリでは、TICKIT データベースのテーブルをロードするためのエントリが返されます。

```
SELECT query, trim(filename) AS filename, curtime, status
FROM stl_load_commits
WHERE filename like '%tickit%' order by query;


 query |           filename         |          curtime           | status
-------+----------------------------+----------------------------+--------
 22475 | tickit/allusers_pipe.txt   | 2013-02-08 20:58:23.274186 |      1
 22478 | tickit/venue_pipe.txt      | 2013-02-08 20:58:25.070604 |      1
 22480 | tickit/category_pipe.txt   | 2013-02-08 20:58:27.333472 |      1
 22482 | tickit/date2008_pipe.txt   | 2013-02-08 20:58:28.608305 |      1
 22485 | tickit/allevents_pipe.txt  | 2013-02-08 20:58:29.99489  |      1
 22487 | tickit/listings_pipe.txt   | 2013-02-08 20:58:37.632939 |      1
 22489 | tickit/sales_tab.txt       | 2013-02-08 20:58:37.632939 |      1
(6 rows)
```

入力データを検証する

実際にデータをロードする前に Amazon S3 入力ファイルまたは Amazon DynamoDB テーブルのデータを検証するには、COPY コマンドとともに NOLOAD オプションを使用します。データをロードする際に使用するものと同じ COPY コマンドおよびオプションとともに NOLOAD を使用します。NOLOAD はデータベースにロードすることなくすべてのデータの完全性をチェックします。NOLOAD オプションは、データのロードを試行した場合に発生する可能性のあるエラーがあればそれらを表示します。

例えば、入力ファイルに間違った Amazon S3 パスを指定した場合、Amazon Redshift は以下のエラーを表示します。

```
ERROR:  No such file or directory
DETAIL:
-----------------------------------------------
Amazon Redshift error:  The specified key does not exist
code:      2
```

```
context:   S3 key being read :
location:  step_scan.cpp:1883
process:   xenmaster [pid=22199]
-----------------------------------------------
```

エラーメッセージをトラブルシューティングする方法については、「ロードエラー参照」を参照してください。

NOLOAD オプションの使用例については、「NOLOAD オプションを使用する COPY コマンド」を参照してください。

自動圧縮ありでテーブルをロードする

データの独自の評価に基づいて、テーブルの列に圧縮エンコーディングを手動で適用できます。または、COMPUPDATE を ON に設定して COPY コマンドを使用すると、サンプルデータに基づいて自動的に圧縮を分析および適用できます。

新しいテーブルを作成し、ロードするときに自動圧縮を利用できます。COPY コマンドは、圧縮分析を実行します。また、すでに入力されているテーブルに ANALYZE COMPRESSION コマンドを実行すれば、データをロードしたり、テーブルの圧縮を変更したりすることなく圧縮分析を実行できます。例えば、既存のデータ定義言語 (DDL) ステートメントを保持しながら、将来の使用のためにテーブルの圧縮を分析する場合は、ANALYZE COMPRESSION を実行できます。

自動圧縮では、圧縮エンコードを選択する際に全体的なパフォーマンスの負荷を分散させます。ソートキー列が、同じクエリ内の他の列よりもかなり高度に圧縮される場合、範囲が制限されたスキャンはパフォーマンスが低下する可能性があります。その結果、自動圧縮はソートキー列のデータ分析フェーズをスキップし、ユーザー定義のエンコーディングタイプを保持します。

自動圧縮では、エンコードのタイプを明示的に定義していない場合、RAW エンコードが選択されます。ANALYZE COMPRESSION の動作は同じです。最適なクエリパフォーマンスを得るには、ソートキーに RAW を使用することを検討してください。

自動圧縮の仕組み

COMPUPDATE パラメータが ON になっている場合、COPY コマンドは、空のターゲットテーブルで COPY コマンドを実行し、すべてのテーブル列で RAW エンコーディングかエンコーディングなしが設定されている場合に、自動圧縮を適用します。

現在の圧縮エンコーディングに関係なく、空のテーブルに自動圧縮を適用するには、COMPUPDATE オプションを ON に設定して COPY コマンドを実行します。自動圧縮を無効にするには、COMPUPDATE オプションを OFF に設定して COPY コマンドを実行します。

すでにデータが含まれているテーブルには自動圧縮を適用できません。

> **Note**
>
> 自動圧縮分析を実行するには、意味のあるサンプルを生成するために十分な行 (少なくともスライスごとに 100,000 行) がロードデータに含まれている必要があります。

自動圧縮では、ロード処理の一部として次の操作がバックグラウンドで実行されます。

1. 行の初回サンプルが入力ファイルからロードされます。サンプルサイズは COMPROWS パラメータの値に基づきます。デフォルトは 100,000 です。
2. 圧縮オプションは列ごとに選択されます。
3. サンプル行がテーブルから削除されます。
4. テーブルは、選択した圧縮エンコーディングで再作成されます。
5. 入力ファイル全体がロードされ、新しいエンコーディングで圧縮されます。

COPY コマンドを実行すると、テーブルが完全にロードされ、圧縮され、使用する準備ができます。その後、追加でデータをロードする場合、追加された行は既存のエンコーディングに基づいて圧縮されます。

圧縮分析のみを実行する場合、ANALYZE COMPRESSION を実行します。完全な COPY を実行するよりも効率的です。その後、結果を評価し、自動圧縮を使用するのか、またはテーブルを手動で再作成するのかを決定できます。

自動圧縮は COPY コマンドでのみサポートされます。代わりに、テーブルを作成するときに圧縮エンコーディングを手動で適用できます。手動圧縮エンコーディングに関する詳細は、「保存データのサイズを削減するための列圧縮」を参照してください。

自動圧縮の例

この例では、TICKIT データベースに「BIGLIST」という名前の LISTING テーブルのコピーが含まれており、約 300 万行をロードするときにこのテーブルに自動圧縮を適用するものと仮定します。

テーブルをロードし、自動的に圧縮するには、次の操作を実行します。

1. テーブルが空であることを確認します。空のテーブルにのみ自動圧縮を適用できます:

```
TRUNCATE biglist;
```

2. 単一の COPY コマンドでテーブルをロードします。テーブルは空であっても、以前のエンコーディングが指定されている可能性があります。Amazon Redshift が圧縮分析を実行できるようにするために、COMPUPDATE パラメータを ON に設定します。

```
COPY biglist FROM 's3://amzn-s3-demo-bucket/biglist.txt'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
DELIMITER '|' COMPUPDATE ON;
```

   COMPROWS オプションを指定していないため、デフォルトであり、推奨のサンプルサイズである 100,000 行 (スライスごとに) が使用されます。

3. BIGLIST テーブルの新しいスキーマを見て、自動的に選択されたエンコーディングスキームを確認します。

```
SELECT "column", type, encoding
from pg_table_def where tablename = 'biglist';


     Column     |            Type             | Encoding
----------------+-----------------------------+----------
 listid         | integer                     | az64
 sellerid       | integer                     | az64
 eventid        | integer                     | az64
 dateid         | smallint                    | none
 numtickets     | smallint                    | az64
 priceperticket | numeric(8,2)                | az64
 totalprice     | numeric(8,2)                | az64
 listtime       | timestamp without time zone | az64
```

4. 予想どおりの数の行がロードされたことを確認します。

```
select count(*) from biglist;

count
---------
3079952
(1 row)
```

後に COPY または INSERT ステートメントを使用してこのテーブルに行が追加されるとき、同じ圧縮エンコーディングが適用されます。

細長いテーブルのストレージの最適化

列の数が非常に少なく、行の数が非常に多いテーブルがある場合、3 つの非表示メタデータ ID 列 (INSERT_XID、DELETE_XID、ROW_ID) がテーブルのディスク領域を過度に消費します。

非表示列の圧縮を最適化するには、可能な限り、1 回の COPY 処理でテーブルをロードします。複数の別個の COPY コマンドでテーブルをロードする場合、INSERT_XID 列は十分に圧縮されません。複数の COPY コマンドを使用する場合、バキューム操作を実行する必要があります。ただし、INSERT_XID の圧縮は改善されません。

デフォルトの列値をロードする

任意で COPY コマンドに列のリストを定義できます。このリストに含まれていないテーブルの列については、COPY を実行すると、CREATE TABLE コマンドで指定された DEFAULT オプションにより提供される値か、DEFAULT オプションが指定されていない場合は NULL がロードされます。

COPY を実行し、NOT NULL として定義されている列に NULL を割り当てようとすると、COPY コマンドは失敗します。DEFAULT オプションの割り当てに関する詳細は、「CREATE TABLE」を参照してください。

Amazon S3 のデータファイルからロードするとき、リストの列がデータファイルのフィールドと同じ順序になっている必要があります。リストに指定された列に該当するフィールドがデータファイルにない場合、COPY コマンドは失敗します。

Amazon DynamoDB テーブルからロードする場合、順序は関係ありません。Amazon Redshift テーブルの列に一致しない Amazon DynamoDB 属性のフィールドは破棄されます。

COPY コマンドを使って DEFAULT 値をテーブルにロードするとき、次の制限が適用されます。

- IDENTITY 列がリストに含まれる場合、EXPLICIT_IDS オプションも COPY コマンドに指定する必要があります。指定しない場合、COPY コマンドは失敗します。同様に、IDENTITY 列をリストから除外し、EXPLICIT_IDS オプションを指定すると、COPY 操作は失敗します。
- 指定の列に対して評価される DEFAULT 式はロードされるすべての行で同じであるため、RANDOM() 関数を使用する DEFAULT 式はすべての行に同じ値を割り当てます。
- CURRENT_DATE または SYSDATE を含む DEFAULT 式は現在の処理のタイムスタンプに設定されます。

例えば、「COPY の例」の「デフォルト値を使用してファイルのデータをロードする」を参照してください。

データロードのトラブルシューティング

Amazon Redshift のテーブルにデータをロードすると、Amazon S3 からのエラー、無効な入力データ、COPY コマンドエラーが発生する可能性があります。以下のセクションでは、データロードエラーの特定と解決に関する情報を提供します。

トピック

- S3ServiceException エラー
- データロードをトラブルシューティングするためのシステムテーブル
- マルチバイト文字のロードエラー
- ロードエラー参照

S3ServiceException エラー

最も一般的な s3ServiceException エラーは、異なる AWS リージョンに存在するクラスターやバケットを指定していたり、不十分な Amazon S3 のアクセス許可が記述されていたりする、形式が不適切か不正確である認証情報文字列により引き起こされます。

このセクションでは、各種エラーのトラブルシューティング情報を提供します。

無効な認証情報文字列

認証情報文字列が不適切にフォーマットされた場合、次のエラーメッセージを受け取ります。

```
ERROR: Invalid credentials. Must be of the format: credentials
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>
[;token=<temporary-session-token>]'
```

認証情報文字列にスペースまたは改行が含まれておらず、一重引用符で囲まれていることを確認してください。

無効なアクセスキー ID

アクセスキー ID が存在しない場合、次のエラーメッセージが表示されます。

```
[Amazon](500310) Invalid operation: S3ServiceException:The AWS Access Key Id you
 provided does not exist in our records.
```

多くの場合、これはコピーと貼り付けのエラーです。アクセスキー ID を正しく入力したことを確認してください。また、一時的なセッションキーを使用している場合は、token の値が設定されていることを確認します。

無効なシークレットアクセスキー

シークレットアクセスキーが正しくない場合、次のようなエラーメッセージを受け取ります。

```
[Amazon](500310) Invalid operation: S3ServiceException:The request signature we
 calculated does not match the signature you provided.
Check your key and signing method.,Status 403,Error SignatureDoesNotMatch
```

多くの場合、これはコピーと貼り付けのエラーです。シークレットアクセスキーを正しく入力したこと、およびそれがアクセスキー ID 用の正しいキーであることを確認します。

バケットが異なるリージョンにある

COPY コマンドに指定された Amazon S3 バケットはクラスターと同じ AWS リージョンに置かれている必要があります。Amazon S3 バケットとクラスターが異なるリージョンにある場合、次のようなエラーを受け取ります。

```
ERROR: S3ServiceException:The bucket you are attempting to access must be addressed
 using the specified endpoint.
```

Amazon S3 マネジメントコンソールを利用してバケットを作成するときにリージョンを選択するか、Amazon S3 API または CLI を利用してバケットを作成するときにエンドポイントを指定することで、特定のリージョンに Amazon S3 バケットを作成できます。詳細については、「Amazon S3 にファイルをアップロードして COPY で使用する」を参照してください。

Amazon S3 リージョンの詳細については、Amazon Simple Storage Service ユーザーガイドのバケットへのアクセスを参照してください。

代わりに、COPY コマンドで REGION オプションを使用してリージョンを指定できます。

アクセスが拒否される

ユーザーに十分なアクセス許可がない場合、次のエラーメッセージを受け取ります。

```
ERROR: S3ServiceException:Access Denied,Status 403,Error AccessDenied
```

考えられる原因の 1 つは、認証情報によって特定されたユーザーが、Amazon S3 バケットへの LIST および GET アクセス権を持っていないことです。その他の原因については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 でのアクセス拒否 (403 Forbidden) エラーのトラブルシューティング」を参照してください。

バケットへのユーザーアクセスの管理については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 での Identity and Access Management」を参照してください。

データロードをトラブルシューティングするためのシステムテーブル

以下の Amazon Redshift システムテーブルは、データのロードに関する問題のトラブルシューティングに役立ちます。

- 特定のロード中に発生したエラーを見つけるには、STL_LOAD_ERRORS にクエリします。
- 特定のファイルのロード時間を参照したり、特定のファイルが読み取られたかどうかを確認するには、STL_FILE_SCAN にクエリします。
- STL_S3CLIENT_ERROR にクエリを実行し、Amazon S3 からのデータ転送時に発生したエラーの詳細を調べます。

ロードエラーを検出し、診断するには、次の操作を実行します。

1. ロードエラーに関する詳細を返すビューを作成するか、クエリを定義します。次の例では、STL_LOAD_ERRORS テーブルが STV_TBL_PERM テーブルと結合し、テーブル ID と実際のテーブル名が照合されます。

   ```
   create view loadview as
   (select distinct tbl, trim(name) as table_name, query, starttime,
   trim(filename) as input, line_number, colname, err_code,
   trim(err_reason) as reason
   from stl_load_errors sl, stv_tbl_perm sp
   where sl.tbl = sp.id);
   ```

2. COPY コマンドの MAXERRORS オプションを十分な大きさの値に設定し、データに関する役に立つ情報を COPY で返せるようにします。COPY でエラーが発生した場合、エラーメッセージにより STL_LOAD_ERRORS テーブルで詳細を確認するように指示されます。
3. LOADVIEW ビューにクエリし、エラー詳細を確認します。次に例を示します。

```
select * from loadview where table_name='venue';
```

```
  tbl    | table_name | query |         starttime
--------+------------+-------+----------------------------
 100551 | venue      | 20974 | 2013-01-29 19:05:58.365391

|     input      | line_number | colname | err_code |       reason
+----------------+-------------+---------+----------+---------------------
| venue_pipe.txt |           1 |       0 |     1214 | Delimiter not found
```

4. ビューが返す情報に基づいて、入力ファイルまたはロードスクリプトの問題を修正します。注意するべき一般的なロードエラーには次のものがあります。

   - テーブルのデータ型と入力データフィールドの値の不一致。
   - テーブルの列数と入力データのフィールド数の不一致。
   - 引用符が一致しません。Amazon Redshift は、一重引用符と二重引用符の両方をサポートしています。ただし、これらの引用符は適切にバランスを取る必要があります。
   - 正しくない入力ファイルの日付/時刻形式。
   - 入力ファイルの値が範囲外 (数値列に関して)。
   - 列の値の数がその圧縮エンコーディングの制限を超えている。

マルチバイト文字のロードエラー

CHAR データ型の列はシングルバイト UTF-8 文字のみを最大 127 バイトまで、または 16 進数の 7F まで受け取ります。これは ASCII 文字セットでもあります。VARCHAR 列はマルチバイト UTF-8 文字を最大 4 バイトまで受け取ります。詳細については、「文字型」を参照してください。

ロードデータの行に列のデータ型として有効でない文字が含まれる場合、COPY はエラーを返し、STL_LOAD_ERRORS システムログテーブルの行にエラー番号 1220 がログ記録されます。ERR_REASON フィールドには無効な文字のバイトシーケンスが 16 進数で含まれます。

ロードデータの無効な文字を修正する代替策は、ロードプロセス中に有効でない文字を置換することです。有効でない UTF-8 文字を置換するには、COPY コマンドに ACCEPTINVCHARS オプション付けて実行します。ACCEPTINVCHARS オプションが設定されている場合は、指定した文字がコードポイントに置き換わります。ACCEPTINVCHARS オプションが設定されていない場合、Amazon

Redshift は有効な UTF-8 として文字を受け入れます。詳細については、「ACCEPTINVCHARS」を参照してください。

次に、有効な UTF-8 で記述したコードポイントのリストを示します。ACCEPTINVCHARS オプションが設定されていない場合、COPY オペレーションはエラーを返しません。また、ここでのコードポイントは有効でない文字で記述されています。ACCEPTINVCHARS オプションを使用すると、このコードポイントを指定したい文字に置き換えられます。これらのコードポイントには、`0xFDD0` から `0xFDEF` の範囲および 最大で `0x10FFFF` までの値が含まれ、末尾には `FFFE` または `FFFF` の値が付加されます。

- `0xFFFE`, `0x1FFFE`, `0x2FFFE`, …, `0xFFFFE`, `0x10FFFE`
- `0xFFFF`, `0x1FFFF`, `0x2FFFF`, …, `0xFFFFF`, `0x10FFFF`

次の例は、COPY により UTF-8 文字の `e0 a1 c7a4` を CHAR 列にロードしようとして発生する、エラーの理由を示しています。

```
Multibyte character not supported for CHAR
(Hint: Try using  VARCHAR). Invalid char: e0 a1 c7a4
```

エラーが VARCHAR データ型に関連する場合、エラーの理由にはエラーコードと、有効でない UTF-8 16 進数シーケンスが含まれます。次の例では、COPY で UTF-8 a4 を VARCHAR フィールドにロードしようとした際に発生する、エラーの理由を示しています。

```
String contains invalid or unsupported UTF-8 codepoints.
Bad UTF-8 hex sequence: a4 (error 3)
```

次のテーブルには、VARCHAR ロードエラーの説明と回避策提案を記載しています。これらのエラーの 1 つが発生した場合、文字を有効な UTF-8 コードシーケンスで置換するか、文字を削除します。

エラーコード	Description
1	UTF-8 バイトシーケンスが VARCHAR でサポートされる 4 バイトの上限を超えています。
2	UTF-8 バイトシーケンスが不完全です。COPY を実行したが、文字列の終わりの前にあるマルチバイト文字の連続バイトが予想された数ではありませんでした。

エラーコード	Description
3	UTF-8 シングルバイト文字が範囲外です。開始バイトを 254、255、または 128 から 191 までの間の任意の文字 (128 と 191 を含む) にすることはできません。
4	バイトシーケンスの後続バイトの値が範囲外です。連続バイトは 128 から 191 まで (128 と 191 を含む) にする必要があります。
5	UTF-8 文字が代理として予約されています。代理コードポイント (U+D800～U+DFFF) は有効ではありません。
8	バイトシーケンスが最大 UTF-8 コードポイントを超えています。
9	UTF-8 バイトシーケンスに一致するコードポイントがありません。

ロードエラー参照

ファイルからデータをロードしている間にエラーが発生した場合、STL_LOAD_ERRORS テーブルにクエリしてエラーを特定し、考えられる説明を確認します。次のテーブルは、データロード中に発生する可能性があるすべてのエラーコードの一覧です。

ロードエラーコード

エラーコード	説明
1200	未知の解析エラー。サポートにお問い合わせください。
1201	入力ファイルでフィールド区切り文字が見つかりませんでした。
1202	DDL に定義されているよりも多い列が入力データに含まれていました。
1203	DDL に定義されているよりも少ない列が入力データに含まれていました。
1204	入力データがデータ型で受入可能な範囲を超過していました。
1205	日付形式が有効ではありません。有効な形式については、「DATEFORMAT と TIMEFORMAT の文字列」を参照してください。

エラーコード	説明
1206	タイムスタンプ形式が有効ではありません。有効な形式については、「DATEFORMAT と TIMEFORMAT の文字列」を参照してください。
1207	期待される 0～9 の範囲外の値がデータに含まれています。
1208	FLOAT データ型の形式エラー。
1209	DECIMAL データ型の形式エラー。
1210	BOOLEAN データ型の形式エラー。
1211	入力行にデータがありません。
1212	ロードファイルが見つかりませんでした。
1213	NOT NULL として指定されたフィールドにデータが含まれていませんでした。
1214	区切り記号が見つかりません。
1215	CHAR フィールドエラー。
1216	入力行が有効ではありません。
1217	ID 列の値が有効ではありません。
1218	NULL AS '\0' を使用している場合に、null ターミネーター (NUL または UTF-8 0000) が含まれるフィールドが 1 バイトを超えています。
1219	UTF-8 の 16 進数に無効な桁が含まれています。
1220	文字列には無効であるかサポートされていない UTF-8 コードポイントが含まれています。
1221	ファイルのエンコードが、COPY コマンドで指定したものと同じではありません。
1222	整数値のオーバーフローエラー。

エラーコード	説明
1223	データタイプが無効です。
1224	入力データが不正な JSON 形式であるか、またはスーパーデータ型ではありません。
8001	MANIFEST パラメータを使用した COPY には、Amazon S3 オブジェクトのフルパスが必要です。
9005	指定された終了キーが無効です。

Amazon S3 からの継続的なファイル取り込みによるテーブルのロード (プレビュー)

これは、プレビューで使用できる自動コピー (SQL COPY JOB) に関する、プレリリースドキュメントです。 ドキュメントと機能はどちらも変更されることがあります。この機能については、テスト環境のみで使用し、本番環境では使用しないことをお勧めします。パブリックプレビューは 2024 年 10 月 31 日に終了します。プレビュークラスターは、プレビュー終了 2 週間後に自動的に削除されます。ベータ版の利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

> **Note**
>
> Amazon Redshift クラスターを [Preview] (プレビュー) で作成して、Amazon Redshift の新機能をテストできます。これらの機能を本番稼働で使用したり、[プレビュー] クラスターを本稼働クラスターや別のトラックのクラスターに移動したりすることはできません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。
>
> [Preview] (プレビュー) でクラスターを作成するには
>
> 1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。

2. ナビゲーションメニューで [Provisioned clusters dashboard] (プロビジョニングされたクラスターダッシュボード) を選択し、[Clusters] (クラスター) を選択します。現在の AWS リージョン にあるアカウントのクラスターがリストされています。各クラスターのプロパティのサブセットが、リストの列に表示されます。
3. [Clusters] (クラスター) リストページに、プレビューを紹介するバナーが表示されます。[Create preview cluster] (プレビュークラスターの作成) ボタンを選択して、クラスターの作成ページを開きます。
4. クラスターのプロパティを入力します。テストしたい機能を含む [プレビュートラック] を選択します。プレビュートラックにあることを示すクラスターの名前を入力することをお勧めします。テストする機能について、-preview というラベルの付いたオプションを含む、クラスターのオプションを選択します。クラスター作成の詳細については、「Amazon Redshift 管理ガイド」の「クラスターの作成」を参照してください。
5. [クラスターを作成] を選択して、プレビューのクラスターを作成します。
6. プレビュークラスターが使用可能になったら、SQL クライアントを使用してデータをロードし、クエリを実行します。

クラスターは `preview_2023` という名前のプレビュートラックで作成する必要があります。テストには新しいクラスターを使用してください。このトラックへのクラスターの復元はサポートされていません。自動コピー機能は Amazon Redshift Serverless ワークグループでは使用できません。
このプレビューは以下の AWS リージョン で使用できます。

- 米国東部 (オハイオ) リージョン (us-east-2)
- 米国東部 (バージニア北部) リージョン (us-east-1)
- 米国西部 (オレゴン) リージョン (us-west-2)
- アジアパシフィック (東京) リージョン (ap-northeast-1)
- 欧州 (ストックホルム) リージョン (eu-north-1)
- 欧州 (アイルランド) リージョン (eu-west-1)

COPY JOB を使用して、Amazon S3 に保存されているファイルから Amazon Redshift テーブルにデータをロードすることができます。Amazon Redshift は、COPY コマンドで指定したパスに新しい Amazon S3 ファイルが追加されると、これを検出します。これに伴い、外部データインジェストパイプラインを作成しなくても、COPY コマンドが自動的に実行されます。Amazon Redshift は、ロー

ドされたファイルを追跡します。Amazon Redshift は、COPY コマンドごとにまとめてバッチ処理されるファイルの数を決定します。結果の COPY コマンドはシステムビューで表示できます。

COPY JOB は 1 回定義します。今後の実行でも同じパラメータが使用されます。

CREATE、LIST、SHOW、DROP、ALTER、RUN の各ジョブのオプションを使用して、ロード操作を管理します。詳細については、「COPY JOB (プレビュー)」を参照してください。

システムビューにクエリを実行して、COPY JOB のステータスと進行状況を確認できます。ビューは次のように表示されます。

- SYS_COPY_JOB (プレビュー) — 現在定義されている各 COPY JOB の行が含まれます。
- STL_LOAD_ERRORS — COPY コマンドのエラーが含まれます。
- STL_LOAD_COMMITS — COPY コマンドのデータロードのトラブルシューティングに使用される情報が含まれます。
- SYS_LOAD_HISTORY — COPY コマンドの詳細が含まれます。
- SYS_LOAD_ERROR_DETAIL — COPY コマンドエラーの詳細が含まれます。

COPY JOB によってロードされたファイルのリストを取得するには、以下の例を *<job_id>* に置き換えて実行します。

```
SELECT job_id, job_name, data_source, copy_query,filename,status, curtime
FROM sys_copy_job copyjob
JOIN stl_load_commits loadcommit
ON copyjob.job_id = loadcommit.copy_job_id
WHERE job_id = <job_id>;
```

DML コマンドによるテーブルのロード

Amazon Redshift は、テーブルの行の変更に利用できる標準のデータ操作言語 (DML) コマンド (INSERT、UPDATE、DELETE) をサポートします。TRUNCATE コマンドを使用し、高速一括削除を実行することもできます。

> **Note**
>
> 大量のデータをロードする場合は COPY コマンドを使用することを強くお勧めします。個々に INSERT ステートメントを使ってテーブルにデータを入力すると著しく時間がかかる場合があります。または、他の Amazon Redshift データベーステーブルにデータが既に存在する

場合、パフォーマンスを向上させるには INSERT INTO ... SELECT FROM または CREATE TABLE AS を使用します。詳細については、「INSERT」または「CREATE TABLE AS」を参照してください。

テーブルで行を挿入、更新、削除するとき、それが変更前と比較して相当な数になる場合、完了時にテーブルに ANALYZE および VACUUM コマンドを実行します。アプリケーションの小さな変更の数が時間とともに累積される場合、定期的に ANALYZE および VACUUM コマンドを実行するように日程を計画することもできます。詳細については、テーブルを分析するおよびテーブルのバキューム処理を参照してください。

トピック

- 新しいデータの更新と挿入

新しいデータの更新と挿入

MERGE コマンドを使用すると、既存のテーブルに新しいデータを効率的に追加できます。マージ操作を実行するには、ステージングテーブルを作成し、このセクションで説明している方法のいずれかを使用して、ステージングテーブルからターゲットテーブルを更新します。MERGE コマンドの詳細については、「MERGE」を参照してください。

マージの例 は、Amazon Redshift 用のサンプルデータセット (TICKIT データセット) を使用します。前提条件として、「一般的なデータベースタスクの開始方法」に記載されている手順に従って、TICKIT テーブルとデータを設定できます。サンプルデータセットの詳細については、「サンプルデータベース」を参照してください。

マージ方法 1: 既存の行を置き換える

ターゲットテーブルのすべての列を上書きする場合、マージを実行する最速の方法は、既存の行を置き換えることです。これは、内部結合を使用して更新される行を削除するため、ターゲットテーブルを 1 回だけスキャンします。行は削除されると、1 回の挿入オペレーションによってステージングテーブルの新しい行と置き換えられます。

この方法は、以下の条件のすべてに該当する場合に使用してください。

- ターゲットテーブルとステージングテーブルに同じ列が含まれる。
- ターゲットテーブルの列のデータをすべて、ステージングテーブルの列で置き換えることを予定している。

- マージでステージングテーブルの行をすべて使用する。

ここに挙げた条件のなかに当てはまらないものがあった場合は、次のセクションで説明する「マージ方法 2: MERGE を使用せずに列リストを指定する」を使用します。

ステージングテーブルの行すべてを使用するわけではない場合には、DELETE ステートメントおよび INSERT ステートメントに WHERE 句を使用して、変更がない行を除外します。ただし、マージでステージングテーブルの行の大部分を使用しない場合には、このセクションで後ほど説明するように UPDATE と INSERT を別個に実行する方法を推奨しています。

マージ方法 2: MERGE を使用せずに列リストを指定する

行全体を上書きするのではなく、テーブルの特定の列を更新する場合には、この方法を使用します。この方法では、更新のステップを追加する必要があり、MERGE コマンドを使用しないため、前の方法よりも時間がかかります。このため、以下の条件のいずれかに該当する場合にこの方法を使用してください。

- テーブルの列すべてを更新するわけではない。
- 更新でステージングテーブルの行のほとんどを使用しない。

トピック

- 一時的に使用するステージングテーブルを作成する
- 既存の行を置き換えてマージ操作を実現する
- MERGE コマンドを使用せずに列リストを指定してマージ操作を実行する
- マージの例

一時的に使用するステージングテーブルを作成する

ステージングテーブルは、ターゲットテーブルを変更 (更新、挿入含む) する際に使用するデータすべてを一時的に保持するためのテーブルです。

マージ操作では、ステージングテーブルとターゲットテーブルを結合する必要があります。結合する列をコロケーションするには、ステージングテーブルの分散キーを、ターゲットテーブルの分散キーと同じ列に設定する必要があります。例えば、ターゲットテーブルが分散キー列に外部キーを使用している場合には、ステージングテーブルの分散キーと同じ列を使用する必要があります。CREATE TABLE LIKE ステートメントを使用してステージングテーブルを作成すると、ステージングテーブル

が親テーブルから分散キーを継承します。CREATE TABLE AS ステートメントを使用する場合、新しいテーブルは分散キーを継承しません。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

分散キーがプライマリキーと異なり、マージ操作の一環として分散キーが更新されない場合には、分散キー列の冗長結合述語を追加し、コロケーテッド結合を実現します。次に例を示します。

```
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
```

クエリでコロケーテッド結合を使用するかどうかを確認するには、EXPLAIN を使用してクエリを実行し、結合すべてに DS_DIST_NONE があるかどうかを確認します。詳細については、「クエリプランの評価」を参照してください。

既存の行を置き換えてマージ操作を実現する

手順で説明しているマージオペレーションを実行するときは、一時的なステージングテーブルの作成と削除のステップを除き、すべてのステップを 1 つのトランザクションにまとめます。いずれかのステップが失敗した場合でも、トランザクションはロールバックされます。トランザクションを 1 つにすると、他にもコミットの回数が減るため、時間とリソースの節約になります。

既存の行を置き換えてマージ操作を実現するには、以下の手順を実行します。

1. 次の疑似コードに示すように、ステージングテーブルを作成し、マージの対象となるデータを移します。

   ```
   CREATE temp table stage (like target);

   INSERT INTO stage
   SELECT * FROM source
   WHERE source.filter = 'filter_expression';
   ```

2. MERGE を使用してステージングテーブルとの内部結合を実行し、ステージングテーブルと一致するターゲットテーブルの行を更新します。次に、ステージングテーブルと一致しない残りのすべての行をターゲットテーブルに挿入します。

   更新と挿入の操作は 1 つの MERGE コマンドで実行することをお勧めします。

   ```
   MERGE INTO target
   USING stage [optional alias] on (target.primary_key = stage.primary_key)
   ```

```
WHEN MATCHED THEN
UPDATE SET col_name1 = stage.col_name1 , col_name2= stage.col_name2, col_name3 =
 {expr}
WHEN NOT MATCHED THEN
INSERT (col_name1 , col_name2, col_name3) VALUES (stage.col_name1, stage.col_name2,
 {expr});
```

3. ステージングテーブルを削除 (Drop) します。

```
DROP TABLE stage;
```

MERGE コマンドを使用せずに列リストを指定してマージ操作を実行する

手順で説明しているマージオペレーションを実行するときは、すべてのステップを 1 つのトランザクションにまとめます。いずれかのステップが失敗した場合でも、トランザクションはロールバックされます。トランザクションを 1 つにすると、他にもコミットの回数が減るため、時間とリソースの節約になります。

列リストを指定してマージ操作を実行するには、以下の手順を実行します。

1. 操作全体を 1 つのトランザクションブロックにまとめます。

```
BEGIN transaction;
…
END transaction;
```

2. 次の疑似コードに示すように、ステージングテーブルを作成し、マージの対象となるデータを移します。

```
create temp table stage (like target);
insert into stage
select * from source
where source.filter = 'filter_expression';
```

3. ステージングテーブルで内部結合を使用して、ターゲットテーブルを更新します。

   - UPDATE 句で、更新の対象となる列を明示的にリストします。
   - ステージングテーブルで内部結合を実行します。
   - 分散キーがプライマリキーと異なり、分散キーが更新の対象でない場合は、冗長結合を分散キーに追加します。クエリでコロケーテッド結合を使用するかどうかを確認するに

は、EXPLAIN を使用してクエリを実行し、結合すべてに DS_DIST_NONE があるかどうかを確認します。詳細については、「クエリプランの評価」を参照してください。

- ターゲットテーブルがタイムスタンプでソートされる場合、述語を追加し、ターゲットテーブルの範囲限定スキャンを活用します。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。
- マージですべての行を使用しない場合は、句を追加して、変更する行をフィルタリングします。例えば、1 つ以上の列に不等フィルタを追加して、変更されていない列を除外します。
- 問題が発生した場合に全体をロールバックできるよう、更新操作、削除操作、挿入操作を単一のトランザクションにまとめます。

次に例を示します。

```
begin transaction;

update target
set col1 = stage.col1,
col2 = stage.col2,
col3 = 'expression'
from stage
where target.primarykey = stage.primarykey
and target.distkey = stage.distkey
and target.col3 > 'last_update_time'
and (target.col1 != stage.col1
or target.col2 != stage.col2
or target.col3 = 'filter_expression');
```

4. ターゲットテーブルで内部結合を使用して、ステージングテーブルから不要になった行を削除します。ターゲットテーブルの一部の行はすでにステージングテーブルの対応する行に一致し、その他は前のステップで更新されました。どちらの場合も、挿入のために必要ありません。

```
delete from stage
using target
where stage.primarykey = target.primarykey;
```

5. ステージングテーブルから残りの行を挿入します。ステップ 2 の UPDATE ステートメントで使用した列リストと同じものを VALUES 句内で使用します。

```
insert into target
(select col1, col2, 'expression'
```

```
from stage);

end transaction;
```

6. ステージングテーブルを削除 (Drop) します。

```
drop table stage;
```

マージの例

以下の例では、マージを実行して SALES テーブルを更新します。最初の例では、ターゲットテーブルから削除した後でステージングテーブルからすべての行を挿入する、よりシンプルな方法を使用しています。2 番目の例では、ターゲットテーブル内の選択した列の更新が必要であるため、追加の更新ステップが含まれています。

マージの例 は、Amazon Redshift 用のサンプルデータセット (TICKIT データセット) を使用します。前提条件として、「一般的なデータベースタスクの開始方法」ガイドの手順に従って、TICKIT テーブルとデータを設定できます。サンプルデータセットの詳細については、「サンプルデータベース」を参照してください。

マージデータソースのサンプル

このセクションの例では、更新と挿入を両方とも含んでいるサンプル データソースが必要です。これらの例では、SALES テーブルからのデータを使用する SALES_UPDATE というサンプルテーブルを作成します。12 月の新しい販売アクティビティを表すランダムデータを新しいテーブルに入力します。SALES_UPDATE サンプルテーブルを使用して、ステージングテーブルを以下の例で作成します。

```
-- Create a sample table as a copy of the SALES table.

create table tickit.sales_update as
select * from tickit.sales;

-- Change every fifth row to have updates.

update tickit.sales_update
set qtysold = qtysold*2,
pricepaid = pricepaid*0.8,
commission = commission*1.1
where saletime > '2008-11-30'
```

```
and mod(sellerid, 5) = 0;

-- Add some new rows to have inserts.
-- This example creates a duplicate of every fourth row.

insert into tickit.sales_update
select (salesid + 172456) as salesid, listid, sellerid, buyerid, eventid, dateid,
 qtysold, pricepaid, commission, getdate() as saletime
from tickit.sales_update
where saletime > '2008-11-30'
and mod(sellerid, 4) = 0;
```

マッチングキーに基づいて既存の行を置き換えるマージの例

以下のスクリプトでは、SALES_UPDATE テーブルを使用して、12 月の販売アクティビティの新しいデータによる SALES テーブルでのマージ操作を実行します。この例では、SALES テーブルの更新がある行を置き換えます。この例では、qtysold 列と pricepaid 列を更新しますが、commission と saletime は変更しません。

```
MERGE into tickit.sales
USING tickit.sales_update sales_update
on ( sales.salesid = sales_update.salesid
and sales.listid = sales_update.listid
and sales_update.saletime > '2008-11-30'
and (sales.qtysold != sales_update.qtysold
or sales.pricepaid != sales_update.pricepaid))
WHEN MATCHED THEN
update SET qtysold = sales_update.qtysold,
pricepaid = sales_update.pricepaid
WHEN NOT MATCHED THEN
INSERT (salesid, listid, sellerid, buyerid, eventid, dateid, qtysold , pricepaid,
 commission, saletime)
values (sales_update.salesid, sales_update.listid, sales_update.sellerid,
 sales_update.buyerid, sales_update.eventid,
sales_update.dateid, sales_update.qtysold , sales_update.pricepaid,
 sales_update.commission, sales_update.saletime);

-- Drop the staging table.
drop table tickit.sales_update;

-- Test to see that commission and salestime were not impacted.
SELECT sales.salesid, sales.commission, sales.salestime, sales_update.commission,
 sales_update.salestime
```

```
FROM tickit.sales
INNER JOIN tickit.sales_update sales_update
ON
sales.salesid = sales_update.salesid
AND sales.listid = sales_update.listid
AND sales_update.saletime > '2008-11-30'
AND (sales.commission != sales_update.commission
OR sales.salestime != sales_update.salestime);
```

MERGE を使用せずに列リストを指定するマージの例

以下の例では、12 月の販売アクティビティの新しいデータで SALES を更新するマージ操作を実行します。更新と挿入の両方を含み、変更されていない行を含む、サンプルデータが必要です。この例では、QTYSOLD および PRICEPAID 列を更新し、COMMISSION および SALETIME は変更しません。以下のスクリプトでは、SALES_UPDATE テーブルを使用して、SALES テーブルでのマージ操作を実行しています。

```
-- Create a staging table and populate it with rows from SALES_UPDATE for Dec
create temp table stagesales as select * from sales_update
where saletime > '2008-11-30';

-- Start a new transaction
begin transaction;

-- Update the target table using an inner join with the staging table
-- The join includes a redundant predicate to collocate on the distribution key -- A
 filter on saletime enables a range-restricted scan on SALES

update sales
set qtysold = stagesales.qtysold,
pricepaid = stagesales.pricepaid
from stagesales
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid
and stagesales.saletime > '2008-11-30'
and (sales.qtysold != stagesales.qtysold
or sales.pricepaid != stagesales.pricepaid);

-- Delete matching rows from the staging table
-- using an inner join with the target table

delete from stagesales
using sales
```

```
where sales.salesid = stagesales.salesid
and sales.listid = stagesales.listid;

-- Insert the remaining rows from the staging table into the target table
insert into sales
select * from stagesales;

-- End transaction and commit
end transaction;

-- Drop the staging table
drop table stagesales;
```

ディープコピーを実行する

ディープコピーでは、テーブルを自動的にソートする一括挿入を利用してテーブルを再作成し、データを入力します。テーブルにソートされていない大規模なリージョンがある場合、ディープコピーの方がバキューム処理より高速です。ディープコピーオペレーション中の同時更新は、追跡可能な場合にのみ行うことをお勧めします。プロセスが完了したら、差分更新を新しいテーブルに移動します。VACUUM オペレーションは同時更新を自動でサポートします。

4 つの方法の 1 つを選択し、元のテーブルのコピーを作成できます:

- 元のテーブル DDL を使用します。

  CREATE TABLE DDL が利用できる場合、これが最も速く推奨の方法になります。新しいテーブルを作成する場合は、プライマリキーと外部キーを含むすべてのテーブルと列の属性を指定できます。SHOW TABLE 関数を使用すると、元の DDL を検索できます。
- CREATE TABLE LIKE を使用します。

  元の DDL が利用できない場合、CREATE TABLE LIKE を使用して元のテーブルを再作成できます。新しいテーブルは親テーブルのエンコーディング、分散キー、ソートキー、非 null の属性を継承します。新しいテーブルは親テーブルのプライマリキーと外部キーの属性を継承しませんが、ALTER TABLE を使ってそれらを追加できます。
- 一時テーブルを作成し、元のテーブルの全データを削除します。

  親テーブルのプライマリキーおよび外部キーの属性を保持する必要がある場合。親テーブルに依存関係がある場合は、CREATE TABLE... を使用できます。AS (CTAS) を使って一時テーブルを作成します。次に、元のテーブルの全データを削除してから、一時テーブルのデータを入力します。

一時テーブルを使用すると、永続テーブルを使用する場合と比較して著しくパフォーマンスが向上しますが、データを失うリスクがあります。一時テーブルは、作成したセッションの終了時に自動的に削除されます。TRUNCATE を使うと、トランザクションブロック内にある場合でも、即座にコミットされます。TRUNCATE が成功しても後続の INSERT が完了する前にシャットダウンした場合、データが失われます。データ損失を許容できない場合は、永続テーブルを使用します。

テーブルのコピーを作成した後、新しいテーブルへのアクセスを許可する必要がある場合があります。GRANT を使用してアクセス権限を定義できます。テーブルのすべてのアクセス権限を表示および付与するには、次のいずれかに該当する必要があります。

- スーパーユーザー。
- コピーするテーブルの所有者。
- テーブルの権限を確認するための ACCESS SYSTEM TABLE 権限と、関連するすべての権限の付与権限を持つユーザー。

さらに、ディープコピーが格納されているスキーマの使用権限を付与する必要がある場合があります。ディープコピーのスキーマが元のテーブルのスキーマと異なり、また `public` スキーマでもない場合は、使用権限を付与する必要があります。使用権限を表示および付与するには、次のいずれかに該当する必要があります。

- スーパーユーザー。
- ディープコピーのスキーマに対する USAGE 権限を付与できるユーザー。

元のテーブル DDL を使用してディープコピーを実行するには、次のように実行します。

1. (オプション) `v_generate_tbl_ddl` というスクリプトを実行してテーブル DDL を再作成します。
2. 元の CREATE TABLE DDL を使い、テーブルのコピーを作成します。
3. INSERT INTO … SELECT ステートメントを使い、元のテーブルのデータをコピーに入力します。
4. 古いテーブルに付与されている権限を確認してください。これらの権限は、SVV_RELATION_PRIVILEGES システムビューで確認できます。
5. 必要に応じて、古いテーブルの権限を新しいテーブルに付与します。

6. 元のテーブルで権限を持つすべてのグループとユーザーに使用権限を付与します。ディープコピーテーブルが public スキーマ内にある場合、または元のテーブルと同じスキーマにある場合は、この手順は必要ありません。
7. 元のテーブルを削除 (Drop) します。
8. ALTER TABLE ステートメントを使用し、コピーの名前を元のテーブル名に変更します。

次の例では、sample_copy という名前の SAMPLE の複製を利用し、SAMPLE テーブルにディープコピーを実行します。

```
--Create a copy of the original table in the sample_namespace namespace using the
 original CREATE TABLE DDL.
create table sample_namespace.sample_copy ( … );

--Populate the copy with data from the original table in the public namespace.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
alter table sample_namespace.sample_copy rename to sample;
```

CREATE TABLE LIKE を使用してディープコピーを実行するには、次のように実行します。

1. CREATE TABLE LIKE を使用して新しいテーブルを作成します。

2. INSERT INTO ... SELECT ステートメントを使用し、現在のテーブルから新しいテーブルに行をコピーします。
3. 古いテーブルに付与されている権限を確認してください。これらの権限は、SVV_RELATION_PRIVILEGES システムビューで確認できます。
4. 必要に応じて、古いテーブルの権限を新しいテーブルに付与します。
5. 元のテーブルで権限を持つすべてのグループとユーザーに使用権限を付与します。ディープコピーテーブルが public スキーマ内にある場合、または元のテーブルと同じスキーマにある場合は、この手順は必要ありません。
6. 現在のテーブルを削除 (Drop) します。
7. ALTER TABLE ステートメントを使用し、新しいテーブルの名前を元のテーブル名に変更します。

次の例では、CREATE TABLE LIKE を使用して SAMPLE テーブルにディープコピーを実行します。

```
--Create a copy of the original table in the sample_namespace namespace using CREATE
 TABLE LIKE.
create table sameple_namespace.sample_copy (like public.sample);

--Populate the copy with data from the original table.
insert into sample_namespace.sample_copy (select * from public.sample);

--Check SVV_RELATION_PRIVILEGES for the original table's privileges.
select * from svv_relation_privileges where namespace_name = 'public' and relation_name
 = 'sample' order by identity_type, identity_id, privilege_type;

--Grant the original table's privileges to the copy table.
grant DELETE on table sample_namespace.sample_copy to group group1;
grant INSERT, UPDATE on table sample_namespace.sample_copy to group group2;
grant SELECT on table sample_namespace.sample_copy to user1;
grant INSERT, SELECT, UPDATE on table sample_namespace.sample_copy to user2;

--Grant usage permission to every group and user that has privileges in the original
 table.
grant USAGE on schema sample_namespace to group group1, group group2, user1, user2;

--Drop the original table.
drop table public.sample;

--Rename the copy table to match the original table's name.
```

```
alter table sample_namespace.sample_copy rename to sample;
```

一時テーブルを作成し、元のテーブルの全データを削除する方法でディープコピーを実行するには、次のように実行します。

1. CREATE TABLE AS を使用し、元のテーブルの行を使用して一時テーブルを作成します。
2. 現在のテーブルの全データを削除します。
3. INSERT INTO … SELECT ステートメントを使用し、一時テーブルから元のテーブルに行をコピーします。
4. 一時テーブルを削除 (Drop) します。

次の例では、一時テーブルを作成し、元のテーブルの全データを削除することによって、SALES テーブルにディープコピーが実行されます。元のテーブルはそのまま残っているため、コピーテーブルにアクセス権限を付与する必要はありません。

```
--Create a temp table copy using CREATE TABLE AS.
create temp table salestemp as select * from sales;

--Truncate the original table.
truncate sales;

--Copy the rows from the temporary table to the original table.
insert into sales (select * from salestemp);

--Drop the temporary table.
drop table salestemp;
```

テーブルを分析する

ANALYZE オペレーションは、クエリプランナーで最適な計画の選択に使用される統計メタデータを更新します。

多くの場合、ANALYZE コマンドを明示的に実行する必要はありません。Amazon Redshift は、ワークロードの変更をモニタリングし、統計をバックグラウンドで自動的に更新します。さらに、COPY コマンドは空のテーブルにデータをロードした際に分析を自動で実行します。

テーブルまたはデータベース全体を明示的に分析するには、ANALYZE コマンドを実行します。

自動分析

Amazon Redshift はデータベースを継続的にモニタリングし、バックグラウンドで自動的に分析オペレーションを実行します。システムパフォーマンスへの影響を最小限にするために、自動分析はワークロードが軽い期間に実行されます。

自動分析はデフォルトで有効になっています。自動分析を無効にするには、クラスターのパラメータグループを変更して `auto_analyze` パラメータを **`false`** に設定します。

処理時間を短縮し、システムの全体的なパフォーマンスを向上させるために、Amazon Redshift は、変更の程度が低いテーブルの自動分析を省略します。

分析オペレーションは、最新の統計を含むテーブルを省略します。抽出およびロード (ETL) ワークフローの一環として ANALYZE を実行する場合、自動分析は最新の統計を含むテーブルを省略します。同様に、自動分析によりテーブルの統計が更新された場合、明示的な ANALYZE はテーブルを省略します。

新しいテーブルデータの分析

デフォルトでは、COPY コマンドは空のテーブルにデータをロードした後に ANALYZE を実行します。STATUPDATE を ON に設定すれば、テーブルが空であるかどうかに関係なく、ANALYZE を強制できます。STATUPDATE に OFF を指定した場合、ANALYZE は実行されません。テーブルの所有者とスーパーユーザーのみが ANALYZE コマンドまたは STATUPDATE を ON に設定した COPY コマンドを実行できます。

Amazon Redshift では、以下のコマンドを使用して作成したテーブルが分析されます。

- CREATE TABLE AS (CTAS)
- CREATE TEMP TABLE AS
- SELECT INTO

データの初回ロード後に分析されなかった新しいテーブルにクエリを実行する場合、Amazon Redshift は警告メッセージを返します。その後の更新またはロードの後にテーブルにクエリを実行しても警告は発生しません。分析されていないテーブルを参照するクエリに EXPLAIN コマンドを実行すると、同じ警告メッセージが返されます。

データを空ではないテーブルに追加するとテーブルのサイズが大きく変化する場合はいつでも、統計を明示的に更新することができます。これは、ANALYZE コマンドを実行するか、STATUPDATE オプションを ON にした COPY コマンドを使用することで実行できます。最後の ANALYZE 以降に

挿入または削除された行数の詳細を表示するには、PG_STATISTIC_INDICATOR システムカタログテーブルに対してクエリを実行します。

ANALYZE コマンドの範囲を次のいずれかに指定できます。

- 現在のデータベース全体
- 1 つのテーブル
- 1 つのテーブルの 1 つまたは複数の特定の列
- クエリの述語として使用される可能性が高い列

ANALYZE コマンドを実行すると、テーブルからサンプルの行が取得され、いくつかの計算が行われ、結果的に生成される列の統計が保存されます。デフォルトでは、Amazon Redshift は DISTKEY 列にサンプルパスを実行し、テーブルのその他すべての列に別のサンプルパスを実行します。列のサブセットの統計を生成するには、カンマ区切りの列リストを指定します。PREDICATE COLUMNS 句を使用して ANALYZE を実行して、述語として使用されていない列を省略できます。

ANALYZE 操作はリソースを集中的に使います。そのため、実際に統計更新を必要とするテーブルと列にのみ実行します。定期的に、または同じスケジュールですべてのテーブルのすべての行を分析する必要はありません。データが大幅に変更される場合、次で頻繁に使用される列を分析します。

- ソートおよびグループ化の操作
- 結合
- クエリ述語

処理時間を短縮し、システム全体のパフォーマンスを向上させるために、Amazon Redshift は、変更された行の割合が低いテーブルの ANALYZE をスキップします。この動作は analyze_threshold_percent パラメータで決定されます。デフォルトでは、分析のしきい値は 10 パーセントに設定されます。SET コマンドを実行して、現在のセッションの分析しきい値を変更できます。

頻繁に分析する必要のない列は、大きな VARCHAR 列など、実際に問い合わされることがない事実、単位、関連属性を表す列です。例えば、TICKIT データベースの LISTING テーブルについて考えてみます。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'listing';
```

```
column         |        type        | encoding | distkey | sortkey
---------------+--------------------+----------+---------+---------
listid         | integer            | none     | t       | 1
sellerid       | integer            | none     | f       | 0
eventid        | integer            | mostly16 | f       | 0
dateid         | smallint           | none     | f       | 0
numtickets     | smallint           | mostly8  | f       | 0
priceperticket | numeric(8,2)       | bytedict | f       | 0
totalprice     | numeric(8,2)       | mostly32 | f       | 0
listtime       | timestamp with...  | none     | f       | 0
```

このテーブルに大量の新しいレコードが毎日ロードされる場合、結合キーとしてクエリで頻繁に使用される LISTID 列を定期的に分析する必要があります。TOTALPRICE と LISTTIME が頻繁に使用されるクエリの制約である場合、平日は毎日、それらの列と分散キーを分析できます。

```
analyze listing(listid, totalprice, listtime);
```

アプリケーションの販売者とイベントが非常に静的であり、日付 ID がわずか 2 年または 3 年をカバーする固定日数セットを参照するとします。この場合、これらの列の一意の値は大きく変更されません。ただし、一意の各値のインスタンス数は着実に増加します。

さらに、NUMTICKETS および PRICEPERTICKET メジャーが TOTALPRICE 列とまれに比較され、クエリされるという場合を考えてみてください。この場合、毎週末に 1 回、テーブル全体で ANALYZE コマンドを実行して、毎日分析されていない 5 つの列の統計を更新することができます。

述語列

列リストを指定する、便利な代替方法として、述語として使用される可能性が高い列のみを分析するように選択できます。クエリを実行すると、結合、フィルタ条件、または GROUP BY 句で使用される列は、システムカタログの述語列としてマークされます。PREDICATE COLUMNS 句を指定して ANALYZE を実行すると、ANALYZE 操作には次の基準を満たす列のみが含まれます。

- 列は述語列としてマークされます。
- 列が分散キーです。
- 列はソートキーの一部です。

表の列のいずれも述部としてマークされていない場合、PREDICATE COLUMNS が指定されていても、ANALYZE にはすべての列が含まれます。述語の列としてマークされている列がない場合は、表がまだクエリされていない可能性があります。

ワークロードのクエリパターンが比較的安定している場合は、PREDICATE COLUMNS の使用を選択できます。クエリパターンが可変で、さまざまな列が頻繁に述語として使用される場合、PREDICATE COLUMNS を使用すると一時的に古い統計が返される場合があります。古い統計により、最適でないクエリランタイムプランや長いランタイムにつながる可能性があります。ただし、次に PREDICATE COLUMNS を使用して ANALYZE を実行すると、新しい述語の列が含まれます。

述語の列の詳細を表示するには、次の SQL を使用して PREDICATE_COLUMNS という名前のビューを作成します。

```
CREATE VIEW predicate_columns AS
WITH predicate_column_info as (
SELECT ns.nspname AS schema_name, c.relname AS table_name, a.attnum as col_num,
 a.attname as col_name,
        CASE
            WHEN 10002 = s.stakind1 THEN array_to_string(stavalues1, '||')
            WHEN 10002 = s.stakind2 THEN array_to_string(stavalues2, '||')
            WHEN 10002 = s.stakind3 THEN array_to_string(stavalues3, '||')
            WHEN 10002 = s.stakind4 THEN array_to_string(stavalues4, '||')
            ELSE NULL::varchar
        END AS pred_ts
   FROM pg_statistic s
   JOIN pg_class c ON c.oid = s.starelid
   JOIN pg_namespace ns ON c.relnamespace = ns.oid
   JOIN pg_attribute a ON c.oid = a.attrelid AND a.attnum = s.staattnum)
SELECT schema_name, table_name, col_num, col_name,
       pred_ts NOT LIKE '2000-01-01%' AS is_predicate,
       CASE WHEN pred_ts NOT LIKE '2000-01-01%' THEN (split_part(pred_ts,
 '||',1))::timestamp ELSE NULL::timestamp END as first_predicate_use,
       CASE WHEN pred_ts NOT LIKE '%||2000-01-01%' THEN (split_part(pred_ts,
 '||',2))::timestamp ELSE NULL::timestamp END as last_analyze
FROM predicate_column_info;
```

LISTING テーブルに対して次のクエリを実行するとします。LISTID、LISTTIME、および EVENTID は、結合、フィルタ、および GROUP BY 句で使用されることに注意してください。

```
select s.buyerid,l.eventid, sum(l.totalprice)
from listing l
join sales s on l.listid = s.listid
where l.listtime > '2008-12-01'
group by l.eventid, s.buyerid;
```

次の例に示すように、PREDICATE_COLUMNS ビューをクエリすると、LISTID、EVENTID、および LISTTIME が述語の列としてマークされていることが分かります。

```
select * from predicate_columns
where table_name = 'listing';
```

```
schema_name | table_name | col_num | col_name       | is_predicate |
 first_predicate_use | last_analyze
------------+-----------+---------+----------------+--------------
+---------------------+--------------------
public      | listing    |       1 | listid         | true         | 2017-05-05
 19:27:59 | 2017-05-03 18:27:41
public      | listing    |       2 | sellerid       | false        |
  | 2017-05-03 18:27:41
public      | listing    |       3 | eventid        | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
public      | listing    |       4 | dateid         | false        |
  | 2017-05-03 18:27:41
public      | listing    |       5 | numtickets     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       6 | priceperticket | false        |
  | 2017-05-03 18:27:41
public      | listing    |       7 | totalprice     | false        |
  | 2017-05-03 18:27:41
public      | listing    |       8 | listtime       | true         | 2017-05-16
 20:54:32 | 2017-05-03 18:27:41
```

統計を最新状態に保つことで、クエリプランナーが最適なプランを選択できるようになるため、クエリのパフォーマンスが向上します。Amazon Redshift は、統計をバックグラウンドで自動的に更新します。また、明示的に ANALYZE コマンドを実行することもできます。ANALYZE を明示的に実行することを選択した場合は、以下を実行する必要があります。

- クエリを実行する前に ANALYZE コマンドを実行します。
- 定期的なロードまたは更新サイクルが終わるたびに、データベースで ANALYZE コマンドを定期的に実行します。
- 作成した新しいテーブルと大幅に変更された既存のテーブルまたは列で ANALYZE コマンドを実行します。
- クエリでの使用と変更傾向に基づき、異なるタイプのテーブルおよび列に対し、異なるスケジュールで ANALYZE 操作を実行することを考慮します。

- 時間とクラスターリソースを節約するには、ANALYZE を実行するときに PREDICATE COLUMNS 句を使用します。

スナップショットをプロビジョニング済みクラスターまたはサーバーレス名前空間に復元した後や、一時停止中のプロビジョニング済みクラスターを再開した後で、ANALYZE コマンドを明示的に実行する必要はありません。Amazon Redshift は、このような場合でもシステムテーブル情報を保持するため、手動の ANALYZE コマンドは不要です。Amazon Redshift は、必要に応じて引き続き自動分析オペレーションを実行します。

分析オペレーションは、最新の統計を含むテーブルを省略します。抽出およびロード (ETL) ワークフローの一環として ANALYZE を実行する場合、自動分析は最新の統計を含むテーブルを省略します。同様に、自動分析によりテーブルの統計が更新された場合、明示的な ANALYZE はテーブルを省略します。

ANALYZE コマンド履歴

最後の ANALYZE コマンドがテーブルまたはデータベースで実行された日時を知っておくと役立ちます。ANALYZE コマンドが実行されると、Amazon Redshift は以下のような複数のクエリを実行します。

```
padb_fetch_sample: select * from table_name
```

STL_ANALYZE をクエリして、分析操作の履歴を表示します。Amazon Redshift が、自動分析でテーブルを分析する場合、`is_background` 列は `t` (true) に設定されます。それ以外の場合は、`f` (false) に設定されます。次の例では、STV_TBL_PERM を結合して、テーブル名とランタイムの詳細を表示します。

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;


xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+--------------------
```

```
  1582 | users | Full             | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
244287 | users | Full             | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full             | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped          | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped          | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

または、ANALYZE コマンドが含まれたすべての完了トランザクションで実行されたすべてのステートメントを返す、より複雑なクエリを実行できます:

```
select xid, to_char(starttime, 'HH24:MM:SS.MS') as starttime,
datediff(sec,starttime,endtime ) as secs, substring(text, 1, 40)
from svl_statementtext
where sequence = 0
and xid in (select xid from svl_statementtext s where s.text like 'padb_fetch_sample
%' )
order by xid desc, starttime;

xid  |  starttime   | secs |                  substring
-----+--------------+------+------------------------------------------
1338 | 12:04:28.511 |    4 | Analyze date
1338 | 12:04:28.511 |    1 | padb_fetch_sample: select count(*) from
1338 | 12:04:29.443 |    2 | padb_fetch_sample: select * from date
1338 | 12:04:31.456 |    1 | padb_fetch_sample: select * from date
1337 | 12:04:24.388 |    1 | padb_fetch_sample: select count(*) from
1337 | 12:04:24.388 |    4 | Analyze sales
1337 | 12:04:25.322 |    2 | padb_fetch_sample: select * from sales
1337 | 12:04:27.363 |    1 | padb_fetch_sample: select * from sales
...
```

テーブルのバキューム処理

Amazon Redshift は、バックグラウンドでテーブルを自動でソートし、VACUUM DELETE オペレーションを実行できます。ロードまたは一連の増分更新の後にテーブルをクリーンアップするには、データベース全体または個々のテーブルに対して VACUUM コマンドを実行することもできます。

> **Note**
>
> 必要なテーブルのアクセス許可を持つユーザーのみが、テーブルのバキューム処理を効果的に行うことができます。必要なテーブルアクセス許可なしで VACUUM が実行された場合、オペレーションは完了しますが、効果はありません。バキューム処理を効果的に実行するための有効なテーブルのアクセス許可のリストについては、「VACUUM」を参照してください。
> このため、必要に応じて、テーブルを個別にバキューム処理することをお勧めします。この方法をお勧めするもう 1 つの理由は、データベース全体をバキューム処理すると、コストのかかるオペレーションになる場合があるからです。

自動テーブルソート

Amazon Redshift は、データをバックグラウンドで自動的にソートし、テーブルデータをソートキー順に維持します。Amazon Redshift は、スキャンクエリを追跡し、テーブルのどのセクションがソートする利点あるかを判断します。

システムの負荷に応じて、Amazon Redshift は自動でソートを開始します。この自動ソートにより、データをソートキー順に維持するために VACUUM コマンドを実行する必要が少なくなります。例えば、大きなデータをロードした後に、ソートキー順でデータを完全にソートする必要がある場合でも、VACUUM コマンドを手動で実行することができます。VACUUM SORT コマンドの実行が有効的かどうかを判断するには、SVV_TABLE_INFOのvacuum_sort_benefitをモニタします。

Amazon Redshift は、各テーブルのソートキーを使用するスキャンクエリを追跡します。Amazon Redshift は、テーブルが完全にソートされている場合、各テーブルのデータのスキャンやフィルターの最大改善率を見積ります。この見積もりは、SVV_TABLE_INFOのvacuum_sort_benefit列で表示されます。この列は、unsorted列と一緒に使用することができ、クエリの際、いつテーブルで VACUUM SORT を手動で実行したら効率的かを判断します。unsorted列は、テーブルの物理的なソート順を反映しています。vacuum_sort_benefit 列は、VACUUM SORT を手動で実行することでソートによるテーブルへの影響を特定します。

例えば、次のクエリを考えてみます。

```
select "table", unsorted,vacuum_sort_benefit from svv_table_info order by 1;
```

```
 table | unsorted | vacuum_sort_benefit
```

```
-------+----------+---------------------
 sales |    85.71 |                 5.00
 event |    45.24 |                67.00
```

「sales」というテーブルの場合、86% が物理的にソートされていなくても、86% ソートされていないテーブルからのクエリ実行の影響は、5% だけとなります。理由としては、クエリがテーブルの一部にしかアクセスしていないか、ごく少数のクエリがテーブルにアクセスしたかのどちらかとなります。「event」というテーブルの場合、テーブルは、45% が物理的にソートされていません。ですが、67% のクエリ実行の影響は、クエリがテーブルの大部分にアクセスしたか、テーブルにアクセスしたクエリが多かったことを表しています。「event」というテーブルには、VACUUM SORT の実行が潜在的に有効となります。

自動バキューム削除

削除を実行すると、行は削除対象としてマークされますが、削除されません。Amazon Redshift は、データベーステーブルの削除された行数に基づいて、バックグラウンドで VACUUM DELETE オペレーションを自動的に実行します。Amazon Redshift では、負荷が軽減されているときに VACUUM DELETE を実行するようにスケジュールし、負荷が高いときにはオペレーションを一時停止します。

トピック

VACUUM の頻度

一貫性のあるクエリパフォーマンスを維持するために、必要な頻度でバキューム処理を実行する必要があります。VACUUM コマンドの実行頻度を決めるときは、これらの要因を考慮します。

- VACUUM は、夜間や指定されたデータベース管理期間など、クラスターのアクティビティが最小限になると予想される期間に実行します。
- メンテナンスウィンドウ以外で VACUUM コマンドを実行します。詳細については、「保守時間の時間枠を回避したスケジュール計画」を参照してください。

- 大きなリージョンが未ソートの状態であれば、バキューム処理の時間が長くなります。バキューム処理を遅らせる場合、再整理しなければならないデータが増えるのでバキューム処理にかかる時間が長くなります。
- VACUUM は I/O 集約型な操作です。そのため、バキューム処理の完了にかかる時間が長ければ、同時クエリとクラスターで実行されている他のデータベース操作に与える影響が大きくなります。
- VACUUM は、インターリーブソートを使用するテーブルに対しては時間がかかります。インターリーブテーブルを再ソートする必要があるかどうかを評価するには、SVV_INTERLEAVED_COLUMNS ビューのクエリを実行します。

ソートステージとマージステージ

Amazon Redshift では、2 つのステージでバキュームオペレーションを実行します。最初に未ソートのリージョン内の行をソートし、続いて必要に応じて、テーブルの最後の新しくソートされた行に既存の行をマージします。大きなテーブルでバキュームを実行すると、バキューム操作は一連の差分ソートから成るステップを実行した後で、マージを実行します。オペレーションが失敗した場合、または Amazon Redshift がバキュームの間にオフラインになった場合は、部分的にバキュームが実行されたテーブルまたはデータベースは一貫性のある状態が保たれますが、バキュームオペレーションを手動で再開する必要があります。差分ソートは失われますが、操作が失敗する前にコミットされたマージ済みの行に再度バキューム処理を実行する必要はありません。未ソートのリージョンが大きい場合は、無駄になる時間が大きくなる可能性があります。ソートとマージのステージの詳細については、「マージ済みの行のボリューム管理」を参照してください。

ユーザーは、バキューム処理中のテーブルにアクセスできます。バキューム処理中のテーブルにクエリおよび書き込み操作を実行できますが、DML およびバキュームを同時に実行すると両方の処理時間が長くなる可能性があります。バキューム処理中に UPDATE および DELETE ステートメントを実行する場合は、システムのパフォーマンスが低減する場合があります。差分マージにより UPDATE と DELETE の同時操作が一時的にブロックされ、UPDATE と DELETE により操作で影響のあるテーブルでのマージのステップが一時的にブロックされます。ALTER TABLE などの DDL 操作は、テーブルでバキューム操作が終了するまでブロックされます。

> **Note**
>
> VACUUM の様々な修飾子により、その動作方法が制御されます。それらを使用して、現在のニーズに合わせてバキュームオペレーションを調整することができます。例えば、VACUUM RECLUSTER を使用すると、完全なマージ操作が実行されないので、バキューム操作が短縮されます。詳細については、「VACUUM」を参照してください。

バキュームのしきい値

デフォルトではVACUUM コマンドで、テーブルの行の 95 パーセント以上がすでにソートされているテーブルのソートフェーズをスキップします。ソートフェーズをスキップすることにより、VACUUM のパフォーマンスが大幅に向上します。VACUUM コマンドを実行するとき、テーブル名および TO threshold PERCENT パラメータを含む 1 つのテーブルの、デフォルトのソートしきい値を変更する。

バキュームの種類

バキュームのタイプ別の詳細については、「VACUUM」を参照してください。

バキューム処理時間の最小化

Amazon Redshift は、背景で自動的にデータをソートし、VACUUM DELETE を実行します。これにより、VACUUM コマンドを実行する必要が少なくなります。バキューム処理は時間がかかる可能性があります。データの特性に応じて、バキューム処理時間を最小化するために、以下のプラクティスをお勧めします。

トピック

- インデックスを再生成するかどうかの決定
- 未ソートリージョンのサイズを管理する
- マージ済みの行のボリューム管理
- ソートキー順序でデータをロードする
- 時系列テーブルを使用して保存データを削減する

インデックスを再生成するかどうかの決定

多くの場合、インターリーブソート方式を使用することで、クエリのパフォーマンスを大幅に向上させることができますが、時間の経過とともに、ソートキー列の値の分散が変わった場合、パフォーマンス低下につながることがあります。

最初に COPY または CREATE TABLE AS を使用して空のインターリーブテーブルをロードすると、Amazon Redshift は自動的にインターリーブインデックスを構築します。最初に INSERT を使用してインターリーブテーブルをロードする場合は、その後に VACUUM REINDEX を実行して、インターリーブインデックスを初期化する必要があります。

時間の経過と共に、新しいソートキー値を持つ行を追加するにつれて、ソートキー列の値の分布が変更されると、パフォーマンスが低下する可能性があります。新しい行が既存のソートキー値の範囲内に主に存在する場合、インデックスを再作成する必要はありません。VACUUM SORT ONLY あるいは VACUUM FULL を実行して、ソート順序を復元します。

クエリエンジンはソート順を使用して、クエリの処理用にスキャンする必要のあるデータブロックを効率的に選択できます。インターリーブソートの場合、Amazon Redshift はソートキー列の値を分析して、最適なソート順を決定します。行が追加されて、キー値の分散が変わった、つまりスキューが発生した場合、ソート方法は最適でなくなり、ソートのパフォーマンス低下につながります。ソートキーの分散を再分析するには、VACUUM REINDEX を実行できます。REINDEX オペレーションには時間がかかるため、テーブルに対してインデックスの再生成が有効かどうかを決定するには、SVV_INTERLEAVED_COLUMNS ビューのクエリを実行します。

例えば、以下のクエリを実行すると、インターリーブソートキーを使用するテーブルについて詳細が表示されます。

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;


 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+--------------------
 100048 | customer   |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | part       |   0 |             1.65 | 2015-04-22 22:05:45
 100077 | supplier   |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

`interleaved_skew` の値はスキューの量を示す比率です。値が 1 の場合、スキューがないことを意味します。スキューが 1.4 よりも大きい場合、基とするセットからのスキューでなければ、VACUUM REINDEX により通常はパフォーマンスが向上します。

`last_reindex` で日付値を使用して、前回のインデックス再生成から経過した時間を調べることができます。

未ソートリージョンのサイズを管理する

データがすでに含まれているテーブルに大量の新しいデータをロードするか、定期的な保守管理操作の一環としてテーブルのバキューム処理を実行しないとき、未ソートのリージョンが増えます。長時間のバキューム操作を避けるために、以下の手法を利用できます。

- 定期的なスケジュールでバキューム操作を実行します。

  (テーブルの合計行数の少ないパーセンテージを表す毎日の更新など) 少ない増分でテーブルをロードする場合、VACUUM を定期的に実行すると、個別のバキューム操作が短時間で終了します。

- 最も大きなロードを最初に実行します。

  複数の COPY 操作で新しいテーブルをロードする必要がある場合、最も大きなロードを最初に実行します。新しいテーブルまたはTRUNCATE されたテーブルに初回ロードを実行すると、すべてのデータがソート済みリージョンに直接ロードされます。そのため、バキュームは必要ありません。

- すべての行を削除するのではなく、テーブルの全データを削除します (Truncate)。

  テーブルから行を削除した場合、その行が占有していた領域はバキューム操作を実行するまで再利用されません。ただし、テーブルの全データを削除した (Truncate) 場合、テーブルが空になり、ディスク領域が再利用されます。そのため、バキュームは必要ありません。または、テーブルを削除し (Drop)、再作成します。

- テストテーブルの全データを削除するか、テーブル自体を削除します。

  テスト目的で少ない数の行をテーブルにロードする場合、完了時に行を削除しないでください。代わりに、テーブルの全データを削除し、後続の本稼働のロード操作の一環としてこれらの行を再ロードします。

- ディープコピーを実行します。

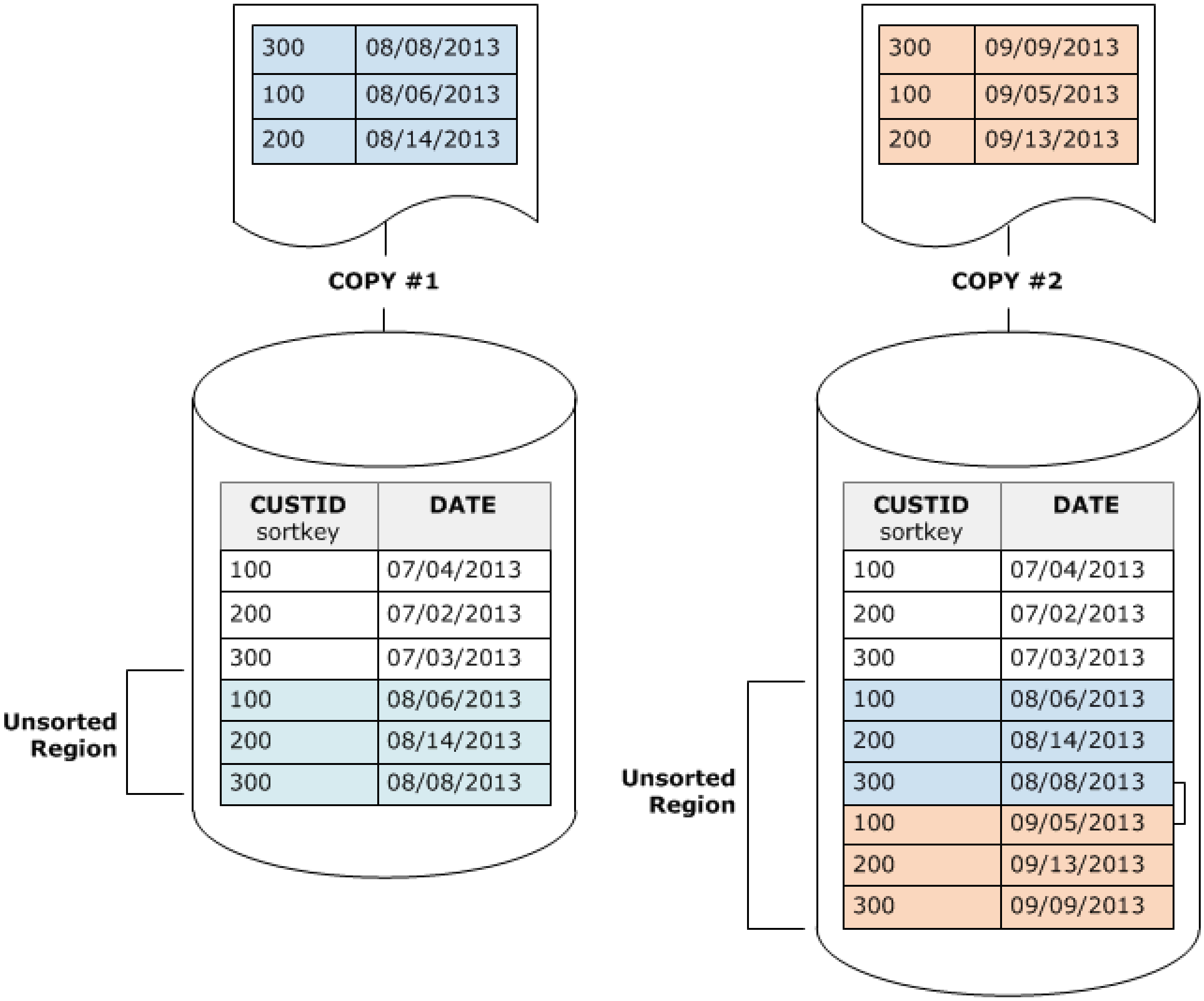
  複合ソートキーテーブルを使用するテーブルにソートされていない大きなリージョンがある場合、ディープコピーがバキュームよりずっと高速です。ディープコピーでは、一括挿入を使用してテーブルが再作成され、再設定されます。これにより、テーブルが自動的に再ソートされます。テーブルにソートされていない大規模なリージョンがある場合、ディープコピーの方がバキューム処理より高速です。欠点としては、ディープコピー処理中は同時更新を実行できません。バキューム処理では実行できます。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。

マージ済みの行のボリューム管理

バキューム操作で新しい行をテーブルのソート済みリージョンにマージする必要がある場合、バキュームに必要な時間はテーブルが大きくなるにつれて長くなります。マージが必要な行の数を少なくすると、バキュームのパフォーマンスが向上します。

バキューム処理前のテーブルは、最初にソート済みリージョン、その後ろに未ソートのリージョンで構成され、未ソート領域は行の追加または更新が行われると大きくなります。COPY 操作で行のセットが追加された場合、新しい行のセットは、テーブルの最後の未ソート領域に追加されたときにソートキーでソートされます。新しい行は、未ソートリージョン内ではなく固有のセット内で順序付けされます。

次の図は、2 つの連続する COPY 操作後の未ソートのリージョンを示しています。ソートキーは CUSTID です。分かりやすいように、この例では複合ソートキーを示していますが、インターリーブテーブルにより未ソートリージョンの影響が大きくならなければ、同じ原則はインターリーブソートキーにも当てはまります。



バキュームにより、2 つのステージでテーブルのソート順が元に戻ります。

1. 未ソートのリージョンを新しくソートされたリージョンにソートします。

   最初のステージでは未ソートのリージョンのみが書き換えられるため、比較的低コストです。新しくソートされたリージョンのソートキーの値の範囲が既存の範囲を超えた場合、新しい行のみを書き換える必要があり、バキュームは完了します。例えば、ソート済みリージョンに 1～500 の ID 値が含まれ、それ以降のコピーオペレーションで追加されたキーの値が 500 を超えた場合は、未ソートのリージョンのみを書き換える必要があります。

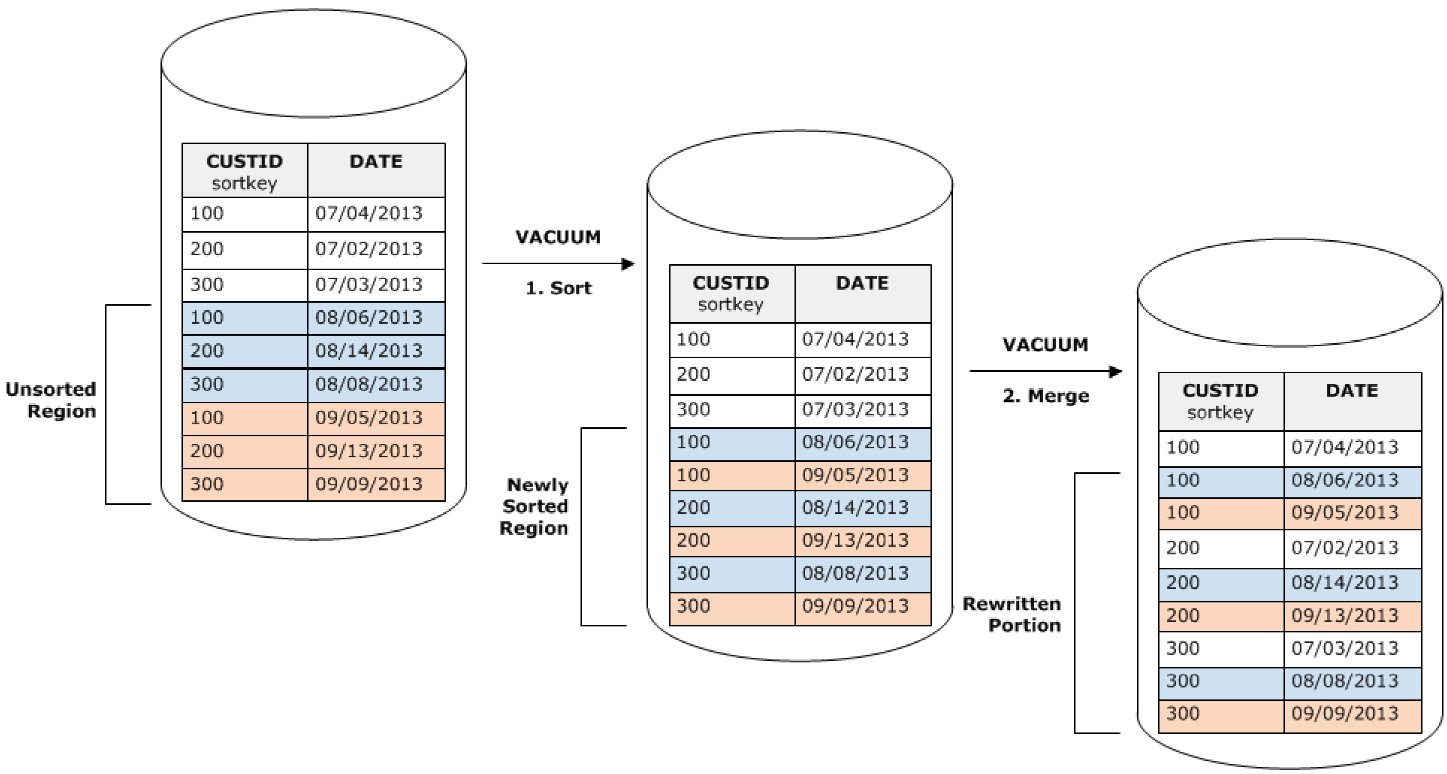
2. 新しくソートされたリージョンと前にソートされたリージョンをマージします。

   新しくソートされたリージョンのキーとソート済みリージョンのキーが重複する場合は、VACUUM で行をマージする必要があります。新しくソートされたリージョンの先頭から開始

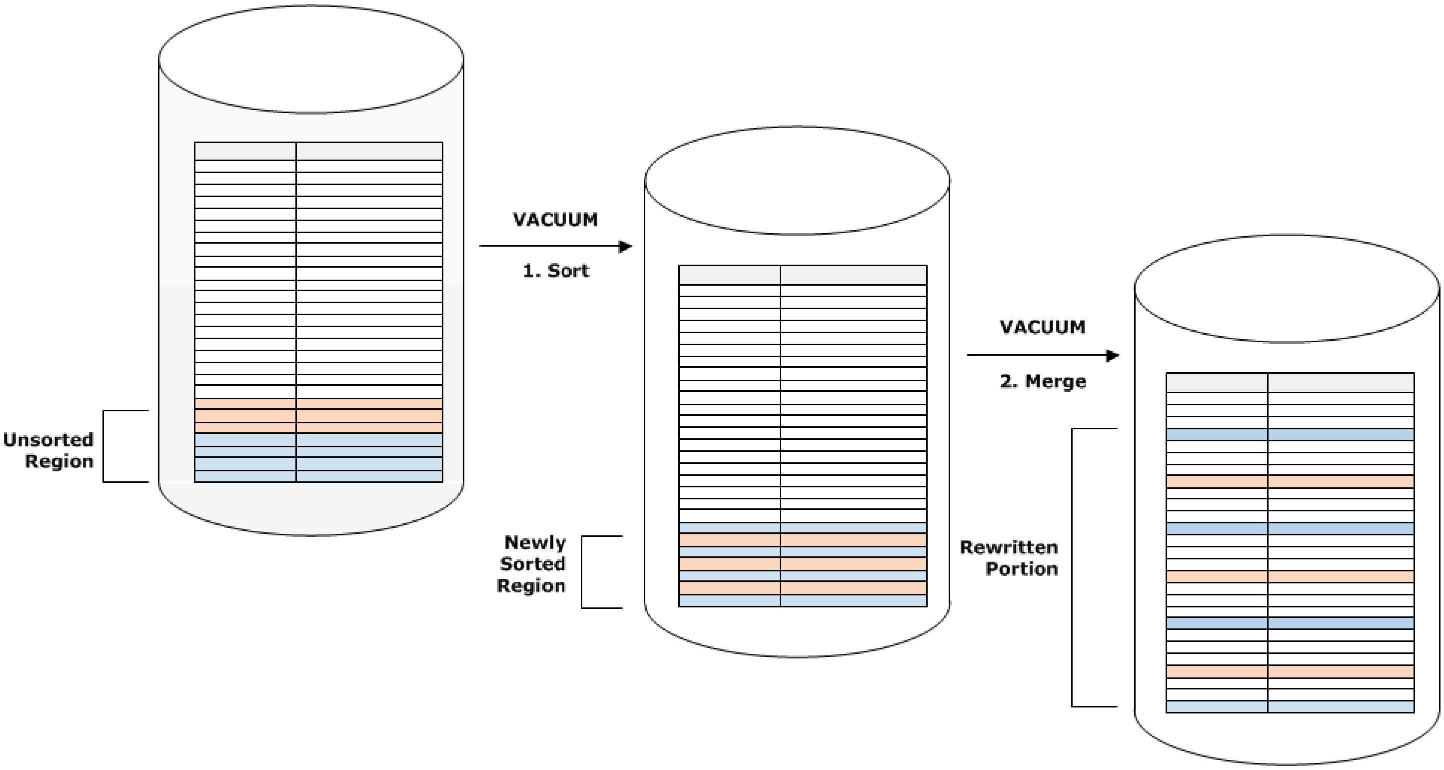
すると (最も低いソートキーで)、バキュームにより、前にソートされたリージョンおよび新しくソートされたリージョンからマージされた行が新しいブロックのセットに書き込まれます。

新しいソートキーの範囲と既存のソートキーが重複する範囲により、再度書き込む必要のあるソート済みリージョンの範囲が決まります。未ソートのキーが既存のソート範囲全体に分散する場合は、バキュームにより、テーブルの既存部分を再度書き込む必要があります。

次の図は、テーブルに追加された行をバキュームによってソートおよびマージする方法を示しています。CUSTID はソートキーです。各コピー操作により、既存のキーと重複するキー値を持つ新しい行セットが追加されるため、テーブルのほぼ全体を再度書き込む必要があります。図に示されているのは単一のソートとマージですが、実際には、大規模なバキュームは一連の差分ソートおよびマージステップから成ります。



新しい行のセットのソートキーの範囲が既存のキーの範囲と重複する場合、マージステージのコストは、テーブルが大きくなると、テーブルサイズに比例して大きくなります。一方、ソートステージのコストは、未ソートリージョンのサイズに比例したままとなります。このような場合、次の図に示すように、マージステージのコストはソートステージのコストを上回ります。



テーブルのどれだけの部分が再マージされたかを特定するには、バキューム操作が完了した後で SVV_VACUUM_SUMMARY のクエリを行います。次のクエリは、CUSTSALES が時間の経過と共に大きくなったときに、6 回連続でバキュームを実行した場合の効果を示します。

```
select * from svv_vacuum_summary
where table_name = 'custsales';


 table_name | xid  | sort_      | merge_     | elapsed_   | row_  | sortedrow_ | block_
  | max_merge_
            |      | partitions | increments | time       | delta | delta      | delta
  | partitions
 -----------+------+------------+------------+------------+-------+------------
+---------+---------------
  custsales | 7072 |          3 |          2 |  143918314 |     0 |   88297472 |   1524
  |       47
  custsales | 7122 |          3 |          3 |  164157882 |     0 |   88297472 |    772
  |       47
  custsales | 7212 |          3 |          4 |  187433171 |     0 |   88297472 |    767
  |       47
  custsales | 7289 |          3 |          4 |  255482945 |     0 |   88297472 |    770
  |       47
```

```
 custsales | 7420 |          3 |          5 |  316583833 |     0 |   88297472 |    769
 |      47
 custsales | 9007 |          3 |          6 |  306685472 |     0 |   88297472 |    772
 |      47
(6 rows)
```

merge_increments 列は、バキューム操作ごとにマージされたデータの量を示します。連続バキュームを超えるマージインクリメントの数がテーブルサイズの増加と比例して増えた場合は、既存のソート済みリージョンと新たにソートされるリージョンが重複するため、バキューム操作ごとにテーブル内の行数が増加して再マージされていることを示します。

ソートキー順序でデータをロードする

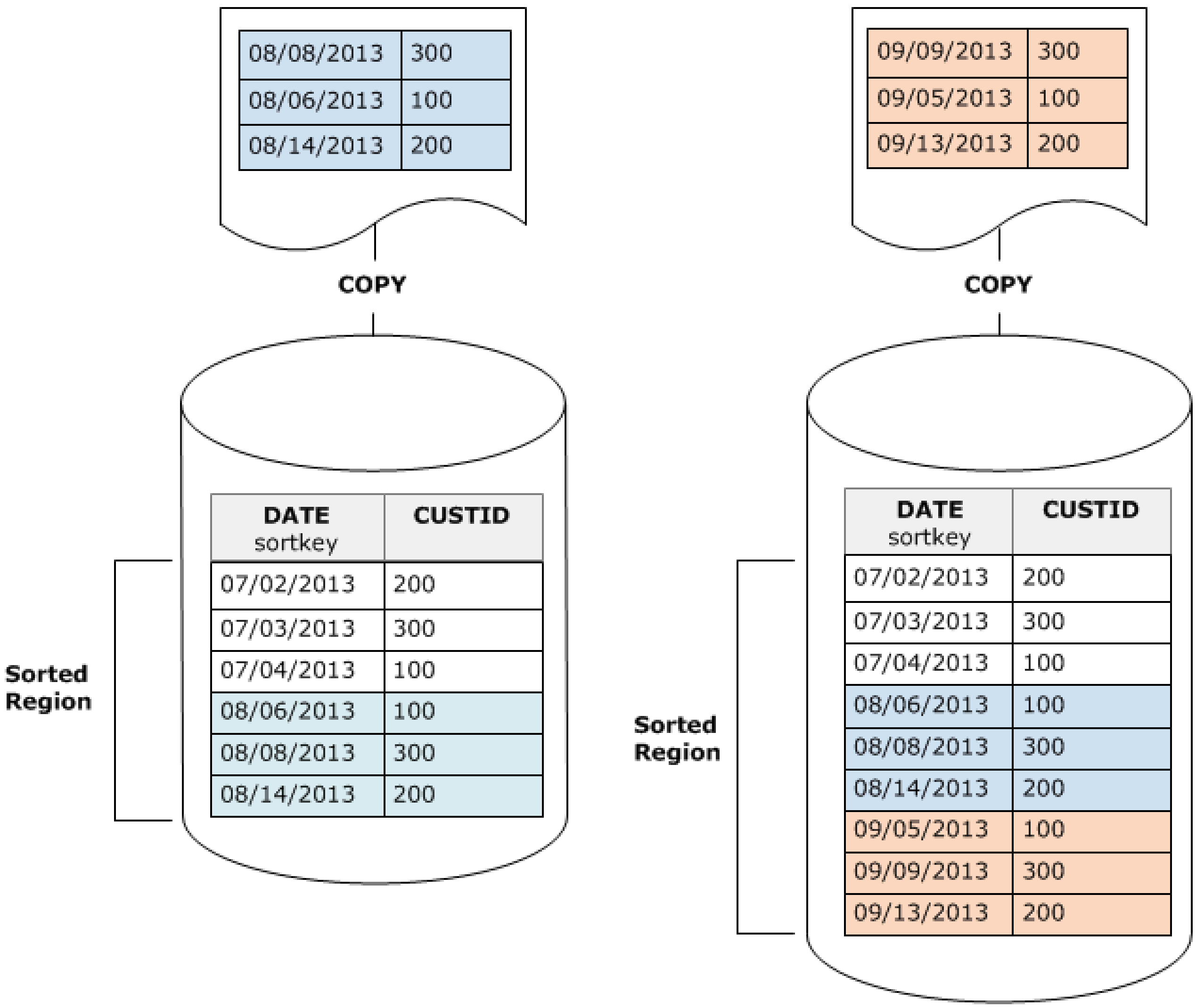
COPY コマンドを使用してソートキー順序でデータをロードする場合、バキューム処理の必要性が減少するか、なくなることもあります。

COPY では、以下のすべてが該当する場合に、テーブルのソート済みリージョンに自動的に新しい行が追加されます。

- テーブルでは、1 つのソート列のみで複合ソートキーが使用されます。
- ソート列は NOT NULL です。
- テーブルは 100% ソート済みであるか空です。
- すべての新しい行は、既存の行 (削除対象としてマークされた行も含む) よりソート順が高くなっています。この場合、Amazon Redshift では、ソートキーの最初の 8 バイトを使用してソート順が決定されます。

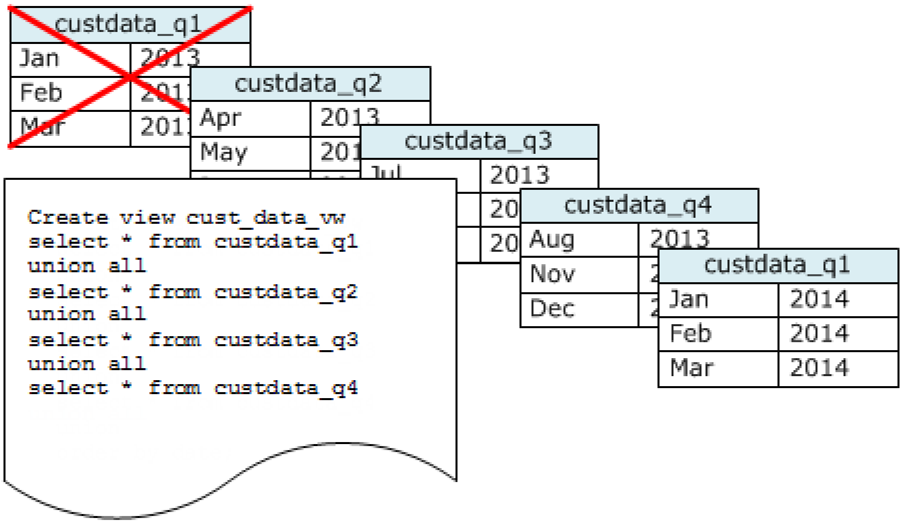
例えば、顧客 ID と時刻を使用して顧客イベントを記録するテーブルがあるとします。顧客 ID でソートする場合は、前の例に示すとおり、差分ロードによって新たに追加された行のソートキー範囲が既存の範囲と重複し、コストの高いバキューム操作につながる可能性があります。

タイムスタンプ列にソートキーを設定する場合、新しい行は、次の図に示すとおり、テーブルの末尾にソート順で追加されるため、バキュームの必要が減少するか、なくなります。



時系列テーブルを使用して保存データを削減する

ローリング期間のデータを保持する場合は、次の図に示すとおり、一連のテーブルを使用します。



データセットを追加するたびに新しいテーブルを作成して、シリーズの最も古いテーブルを削除します。次のような二重の利点があります。

- DROP TABLE 操作は大量の DELETE よりもはるかに効率的であるため、行を削除する余分なコストを避けることができます。
- テーブルがタイムスタンプでソートされる場合は、バキュームは必要ではありません。各テーブルに 1 か月のデータが含まれる場合、テーブルがタイムスタンプでソートされていない場合でも、バキュームで最大でも 1 か月分のデータを再書き込みする必要があります。

レポーティングクエリで使用するために、UNION ALL ビューを作成し、データが複数のテーブルに保存されているという事実を隠すことができます。クエリがソートキーでフィルタリングされる場合は、クエリプランナーは使用されないすべてのテーブルを効率的にスキップできます。その他の種類のクエリでは UNION ALL の効率が下がる可能性があるため、テーブルを使用するすべてのクエリのコンテキストでクエリパフォーマンスを評価してください。

同時書き込み操作を管理する

Amazon Redshift では、増分ロードまたは増分変更の実行中にテーブルを読み込むことができます。

一部の従来のデータウエアハウジングおよびビジネスインテリジェンスアプリケーションでは、毎日の夜間のロードが完了したときにのみデータベースが利用可能になります。そのような場合、分析クエリが実行され、レポートが生成される通常の作業時間の間は更新が許可されません。ただし、増え続けるアプリケーションは日中ずっと、あるいは 1 日中稼動状態を維持し、ロード時間枠という概念は古いものになります。

Amazon Redshift では増分ロードまたは増分変更の実行中にテーブル読み込みができるので、これらのアプリケーションがサポートされます。クエリは、コミットする次のバージョンの待機ではなく、データの最新コミットバージョンまたはスナップショットだけを確認します。特定のクエリに別の書き込み操作からのコミットを待機させる場合、それを適宜スケジュールする必要があります。

次のトピックでは、トランザクション、データベーススナップショット、更新、同時動作など、主要な概念とユースケースの一部を紹介します。

トピック

直列化可能分離

一部のアプリケーションでは、クエリとロードを同時に行うだけでなく、複数のテーブルまたは同じテーブルに同時に書き込む能力を必要とします。この文脈では、同時とは、厳密に同じ時間に実行するようにスケジュールするという意味ではなく、重複するという意味です。最初のトランザクションがコミットする前に 2 つ目のトランザクションが開始する場合、2 つのトランザクションは同時であると考えられます。同時操作は、同じユーザーまたは異なるユーザーが制御する異なるセッションから発生する可能性があります。

> **Note**
>
> Amazon Redshift はデフォルトで自動コミット動作がサポートされています。この機能では、個別に実行された SQL コマンドが個別にコミットします。(BEGIN および END ステートメントにより定義された) トランザクションブロックに一連のコマンドを含めた場合、そのブロックは 1 つのトランザクションとしてコミットされます。そのため、必要に応じてロールバックできます。この動作の例外は、TRUNCATE コマンドと VACUUM コマンドです。このコマンドは、現在のトランザクションで行われた未処理の変更をすべて自動的にコミットします。
> 一部の SQL クライアントでは、BEGIN コマンドと COMMIT コマンドが自動的に発行されるため、クライアントは、ステートメントのグループをトランザクションとして実行するか、個々のステートメントを独自のトランザクションとして実行するかを制御します。使用しているインターフェイスのマニュアルを確認してください。例えば、Amazon Redshift JDBC ドライバーを使用している場合、複数の (セミコロンで区切られた) SQL コマンドを含むクエリ文字列を持つ JDBC `PreparedStatement` は、すべてのステートメントを 1

つのトランザクションとして実行します。対照的に、SQL Workbench/J を使用して AUTO COMMIT ON を設定した後、複数のステートメントを実行すると、各文が独自のトランザクションとして実行されます。

Amazon Redshift では、同時書き込みオペレーションはテーブルの書き込みロックと直列化分離を利用して安全にサポートされます。直列化分離では、テーブルに対して実行されるトランザクションは、そのテーブルに対して実行される唯一のトランザクションであるという錯覚が守られます。例えば、T1 と T2 という 2 つの同時実行トランザクションで次の少なくとも 1 つとして同じ結果が生成されます。

- T1 と T2 がこの順序で連続して実行されます。
- T2 と T1 がこの順序で連続して実行されます。

同時トランザクションは互いに認識されません。互いの変更を検出できません。各同時トランザクションにより、トランザクションの始めにデータベースのスナップショットが作成されます。データベーススナップショットは、ほとんどの SELECT ステートメント、COPY、DELETE、INSERT、UPDATE、TRUNCATE などの DML コマンド、次の DDL コマンドの最初の発生時にトランザクション内で作成されます。

- ALTER TABLE (列を追加または削除する)
- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE

同時トランザクションの任意の直列実行において、同時実行と同じ結果が生成された場合は、そのトランザクションは「直列化可能」とみなされるので、安全に実行できます。そのようなトランザクションの直列実行で、同じ結果が生成されない場合、直列化可能性を阻害する可能性のあるステートメントを実行するトランザクションが中断され、ロールバックされます。

システムカタログテーブル (PG) と他の Amazon Redshift システムテーブル (STL と STV) はトランザクション内にロックされません。そのため、DDL および TRUNCATE オペレーションから発生したデータベースオブジェクトに対する変更は、いずれかの同時トランザクションにコミットすることで表示が可能になります。

例えば、T1 と T2 という 2 つの同時トランザクションが開始するとき、テーブル A がデータベースに存在します。今、T2 が、テーブルのリストを PG_TABLES カタログテーブルから選択して返し

ているとします。この時、T1 はテーブル A をドロップしてコミットし、その後 T2 はテーブルを再度リストします。この後、テーブル A はリストされることはなくなります。T2 が削除されたテーブルのクエリを試行すると、Amazon Redshift は "関係が存在しない" というエラーを返します。T2 にテーブルのリストを返す、またはテーブル A が存在することをチェックするカタログクエリは、ユーザーテーブルに対し実行されるオペレーションと同じ分離ルールの対象にはなりません。

これらのテーブルに対する更新のトランザクションはコミット済み読み取り分離モードで実行されます。PG プレフィックスのカタログテーブルは、スナップショットの分離をサポートしていません。

システムテーブルとカタログテーブルの直列化分離

データベーススナップショットは、ユーザーが作成したテーブルまたは Amazon Redshift システムテーブル (STL または STV) を参照する SELECT クエリのトランザクションでも作成されます。どのテーブルも参照しない SELECT クエリは、新しいトランザクションのデータベーススナップショットを作成しません。システムカタログテーブル (PG) でだけ実行される INSERT、DELETE、UPDATE ステートメントでは、新しいトランザクションのデータベーススナップショットは作成されません。

直列化可能分離エラーを修正する方法

エラー: 1023 詳細: Redshift テーブルで直列化可能分離に関する違反が発生しました

Amazon Redshift で直列化可能な分離エラーが検出されると、次のようなエラーメッセージが表示されます。

```
ERROR:1023 DETAIL: Serializable isolation violation on table in Redshift
```

直列化可能な分離エラーに対処するには、次の方法をお試しください。

- キャンセルされたトランザクションを再試行します。

  Amazon Redshift は、同時ワークロードがシリアル化できないことを検出しました。これは、アプリケーションのロジックにギャップがあることを示唆しています。通常、エラーが発生したトランザクションを再試行することで回避できます。問題が解決しない場合は、他のいずれかの方法をお試しください。

- 同じアトミックトランザクション内にある不要なオペレーションは、トランザクション外に移動します。

この方法は、2 つのトランザクション内の個々のオペレーションが、他のトランザクションの結果に影響を及ぼす可能性のある方法で相互参照する場合に適用されます。例えば、次の 2 つのセッションはそれぞれトランザクションを開始します。

```
Session1_Redshift=# begin;
```

```
Session2_Redshift=# begin;
```

各トランザクションの SELECT ステートメントの結果は、もう一方の INSERT ステートメントの影響を受ける可能性があります。つまり、次のステートメントを任意の順序で連続して実行するとします。いずれの場合でも、結果として、トランザクションが同時に実行された場合よりも、SELECT ステートメントで 1 行多く返ります。同時実行の場合と同じ結果を生成するオペレーションを連続して実行できる順序はありません。そのため、最後に実行されるオペレーションは、直列化可能な分離エラーになります。

```
Session1_Redshift=# select * from tab1;
Session1_Redshift=# insert into tab2 values (1);
```

```
Session2_Redshift=# insert into tab1 values (1);
Session2_Redshift=# select * from tab2;
```

多くの場合、SELECT ステートメントの結果は重要ではありません。つまり、トランザクション内のオペレーションのアトミック性は重要ではありません。これらの場合、次の例に示すように、トランザクションの外で SELECT ステートメントを移動します。

```
Session1_Redshift=# begin;
Session1_Redshift=# insert into tab1 values (1)
Session1_Redshift=# end;
Session1_Redshift=# select * from tab2;
```

```
Session2_Redshift # select * from tab1;
Session2_Redshift=# begin;
Session2_Redshift=# insert into tab2 values (1)
Session2_Redshift=# end;
```

これらの例では、トランザクションに相互参照はありません。2 つの INSERT ステートメントは相互に影響しません。これらの例では、トランザクションを連続して実行し、同時に実行する場合と同じ結果を生成できる順序が少なくとも 1 つあります。つまり、このトランザクションは直列化可能です。

- 直列化を適用するには、各セッションですべてのテーブルをロックします。

  LOCK コマンドでは、直列化可能な分離エラーを引き起こす可能性のあるオペレーションをブロックします。LOCK コマンドを使用するときは、次の点を確認してください。

  - トランザクション内の読み取り専用 SELECT ステートメントの影響を受けるテーブルなど、トランザクションの影響を受けるすべてのテーブルをロックします。
  - オペレーションが実行される順序に関係なく、テーブルを同じ順序でロックします。
  - オペレーションを実行する前に、トランザクションの開始時にすべてのテーブルをロックします。

- 同時実行トランザクションにはスナップショット分離を使用します。

  スナップショット分離で ALTER DATABASE コマンドを使用します。ALTER DATABASE の SNAPSHOT パラメータの詳細については、「パラメータ」を参照してください。

エラー: 1018 詳細: リレーションが存在しません

Amazon Redshift オペレーションの同時実行を異なるセッションで行うと、次のようなエラーメッセージが表示されます。

```
ERROR: 1018 DETAIL: Relation does not exist.
```

Amazon Redshift のトランザクションは、スナップショットの分離に従います。トランザクションが開始されると、Amazon Redshift はデータベースのスナップショットを作成します。トランザクションのライフサイクル全体で、トランザクションはスナップショットに反映されているデータベースの状態で動作します。トランザクションがスナップショットに存在しないテーブルから読み込む場合、前に示した 1018 エラーメッセージをスローします。トランザクションがスナップショットを取得した後に別の並行トランザクションがテーブルを作成した場合でも、トランザクションは新しく作成されたテーブルから読み込むことができません。

このシリアル化分離エラーに対処するには、テーブルが存在することがわかっている時点にトランザクションの開始を移動してみてください。

テーブルが別のトランザクションによって作成された場合、この時点は、少なくともそのトランザクションがコミットされた後です。また、テーブルを削除した可能性のある同時トランザクションがコミットされていないことを確認します。

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session2 = # BEGIN;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # SELECT * FROM A;
```

そのため、session2 によって読み込みオペレーションとして最後に実行されるオペレーションは、直列化可能な分離エラーになります。このエラーは、session2 がスナップショットを取得し、コミットされた session1 によってテーブルがすでに削除されている場合に発生します。別の表現をすると、同時実行の session3 がテーブルを作成しても、それがスナップショット内にないため、session2 はそのテーブルを認識しません。

このエラーを解決するには、次のようにセッションを並べ替えます。

```
session1 = # BEGIN;
session1 = # DROP TABLE A;
session1 = # COMMIT;
```

```
session3 = # BEGIN;
session3 = # CREATE TABLE A (id INT);
session3 = # COMMIT;
```

```
session2 = # BEGIN;
session2 = # SELECT * FROM A;
```

session2 がスナップショットを取得するときに、session3 はすでにコミットされており、テーブルはデータベースにあります。Session2 は、エラーなしでテーブルから読み込むことができます。

書き込みおよび読み取り/書き込みオペレーション

異なるタイプのコマンドを実行するタイミングと方法を決定することで、同時書き込み操作の特定の動作を管理できます。次のコマンドがこの話題に関連します。

- COPY コマンド、(初回または増分) ロードを実行します
- INSERT コマンド、1 つまたは複数の行を 1 回で追加します
- UPDATE コマンド、既存の行を変更します
- DELETE コマンド、行を削除します

COPY および INSERT オペレーションは純粋な書き込み操作です。また、DELETE および UPDATE は読み取り/書き込みオペレーションです。(行を削除または更新するには、最初に読み取りを行う必要があります。) 同時書き込み操作の結果は、同時に実行されている特定のコマンドに依存します。同じテーブルに対する COPY および INSERT 操作はロックが解除されるまで待機状態に置かれます。その後、通常どおり進められます。

UPDATE 操作と DELETE 操作は、書き込み前に初回テーブル読み込みに依存するため、動作が異なります。同時トランザクションが互いに表示されるとすれば、UPDATE と DELETE は最後のコミットからデータのスナップショットを読み取る必要があります。最初の UPDATE または DELETE がそのロックを解除するとき、2 つ目の UPDATE または DELETE はそれがこれから処理するデータが古くなっている可能性がないか決定する必要があります。最初のトランザクションでそのロックが解除されるまで 2 つ目のトランザクションでデータのスナップショットが取得されないため、データは古くなりません。

同時書き込みトランザクションの考えられるデッドロック状況

トランザクションに複数のテーブルの更新が関連するとき、両方が同じテーブルセットに書き込もうとすると、同時実行トランザクションにデッドロックが発生する可能性があります。コミットまたはロールバックするとき、トランザクションはそのすべてのテーブルロックを一度に解除します。1 つずつロックを解除することはありません。

例えば、T1 と T2 というトランザクションが大体同じ時間に開始します。T1 がテーブル A に書き込みを開始し、T2 がテーブル B に書き込みを開始する場合、両方のトランザクションは競合なく進行します。ただし、T1 がテーブル A への書き込みを終了し、テーブル B への書き込みを開始する必要がある場合、T2 が引き続き B をロックしているため、T1 は進行できません。反対に、T2 がテーブル B への書き込みを終了し、テーブル A への書き込みを開始する必要がある場合、T1 が引き続き A をロックしているため、T2 は進行できません。その書込操作がコミットされるまでいずれのトランザクションもそのロックを解除できないため、いずれのトランザクションも進行できません。

この種のデッドロックを回避するには、同時書き込み操作の日程を注意深く計画する必要があります。例えば、常にトランザクションと同じ順序でテーブルを更新する必要があります。ロックを指定する場合、DML 操作を実行する前に同じ順序でテーブルをロックします。

同時書き込みの例

次の疑似コードの例は、同時に実行されたときに、トランザクションが進行または待機する仕組みを示しています。

同じテーブルへの同時 COPY 操作

トランザクション 1 は LISTING テーブルに行をコピーします:

```
begin;
copy listing from ...;
end;
```

トランザクション 2 は別のセッションで同時に開始され、さらに多くの行を LISTING テーブルにコピーしようとします。トランザクション 2 は、トランザクション 1 が LISTING テーブルの書き込みロックを解除するまで待機する必要があります。その後、続行できます。

```
begin;
[waits]
copy listing from ;
end;
```

片方または両方のトランザクションに COPY コマンドの代わりに INSERT コマンドが含まれる場合、同じ動作が起こることがあります。

同じテーブルへの同時 DELETE 操作

トランザクション 1 がテーブルから行を削除します:

```
begin;
delete from listing where ...;
end;
```

トランザクション 2 が同時に開始され、同じテーブルから行を削除しようとします。行の削除を試行する前にトランザクション 1 の完了を待つため、トランザクション 2 は成功します。

```
begin
[waits]
delete from listing where ;
end;
```

片方または両方のトランザクションに DELETE コマンドの代わりに同じテーブルへの UPDATE コマンドが含まれる場合、同じ動作が起こることがあります。

読み取り操作と書き込み操作がミックスされた同時トランザクション

この例では、まずトランザクション 1 は USERS テーブルから行を削除し、テーブルを再ロードします。次に COUNT(*) クエリを実行し、ANALYZE を実行してからコミットします。

```
begin;
delete one row from USERS table;
copy ;
select count(*) from users;
analyze ;
end;
```

その間、トランザクション 2 が開始します。このトランザクションは USERS テーブルへの追加行のコピー、テーブルの分析、最初のトランザクションと同じ COUNT(*) クエリの実行を試行します。

```
begin;
[waits]
copy users from ...;
select count(*) from users;
analyze;
end;
```

2 つ目のトランザクションは最初のトランザクションの完了を待つため、成功します。その COUNT クエリはそれが完了したロードに基づいてカウントを返します。

チュートリアル: Amazon S3 からデータをロードする

このチュートリアルでは、Amazon S3 バケット内のデータファイルから Amazon Redshift データベースのテーブルに、データを最初から最後までロードする手順を説明します。

このチュートリアルでは、以下の作業を行います。

- コンマ区切り (CSV) 形式、文字区切り形式、固定幅形式のデータファイルをダウンロードします。
- Amazon S3 バケットを作成し、データファイルをバケットにアップロードします。
- Amazon Redshift クラスターを起動し、データベーステーブルを作成します。
- COPY コマンドを使用して、Amazon S3 のデータファイルからテーブルをロードします。
- ロードエラーをトラブルシューティングし、COPY コマンドを変更してエラーを修正します。

推定時間: 60 分

推定コスト: クラスターに対して 1.00 USD/時間

前提条件

次の前提条件を満たしている必要があります。

- Amazon Redshift クラスターを起動し、Amazon S3 でバケットを作成するための AWS アカウント。
- Amazon S3 からテストデータをロードするための AWS 認証情報 (IAM ロール)。新しい IAM ロールが必要な場合は、「IAM ロールの作成」を参照してください。
- Amazon Redshift コンソールクエリエディタなどの SQL クライアント。

このチュートリアルはそれだけで実行できるように設計されています。このチュートリアルに加えて Amazon Redshift データベースを設計および使用方法の詳細を理解するには、以下のチュートリアルを完了することをお勧めします。

- Amazon Redshift 入門ガイドでは、Amazon Redshift クラスターを作成してサンプルデータをロードするプロセスについて説明します。

概要

INSERT コマンドを使用するか、または COPY コマンドを使用することで、Amazon Redshift テーブルにデータを追加できます。Amazon Redshift データウェアハウスの規模とスピードでは、COPY コマンドの方が INSERT コマンドよりも何倍も高速で、より効率的です。

COPY コマンドは Amazon Redshift の超並列処理 (MPP) アーキテクチャを使用し、複数のデータソースからデータを並列で読み込んでロードします。Amazon S3 のデータファイル、Amazon

EMR、または Secure Shell (SSH) 接続でアクセス可能なリモートホストからロードできます。あるいは Amazon DynamoDB テーブルから直接ロードできます。

このチュートリアルでは、COPY コマンドを使用して Amazon S3 からデータをロードします。ここで示す原則の多くは、他のデータソースからのロードにも適用されます。

COPY コマンドの使用の詳細については、次のリソースを参照してください。

- データをロードするための Amazon Redshift のベストプラクティス
- Amazon EMR からのデータのロード
- リモートホストからデータをロードする
- Amazon DynamoDB テーブルからのデータのロード

ステップ 1: クラスターを作成する

使用するクラスターがすでにある場合は、この手順を省略できます。

このチュートリアルの演習では、4 ノードクラスターを使用します。

クラスターを作成するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。

   ナビゲーションメニューで [プロビジョニングされたクラスターダッシュボード] を選択します。

   > Important
   >
   > クラスターオペレーションを実行するために必要なアクセス許可を持っていることを確認してください。必要なアクセス許可の付与については、「Amazon Redshift が AWS サービスにアクセスすることを許可する」を参照してください。

2. 右上で、クラスターを作成する AWS リージョンを選択します。このチュートリアルは、[US West (Oregon) (米国西部 (オレゴン))] を選択するためのものです。
3. ナビゲーションメニューで [Clusters] (クラスター)、[Create cluster] (クラスターを作成) の順に選択します。[クラスターの作成] ページが表示されます。

4. [Create cluster] (クラスターの作成) ページで、クラスターのパラメータを入力します。以下の値を除き、パラメータには自身が使用する値を選択します。

   - ノードタイプとして **dc2.large** を選択します。
   - [Number of nodes] (ノード数) には **4** を指定します。
   - [クラスターパラメータ] で、IAM ロールを [Available IAM roles (使用可能な IAM ロール)] から選択します。このロールは前もって作成したものであり、Amazon S3 へのアクセス権限を持っている必要があります。次に、[Associate IAM role] (IAM ロールのアソシエート) をクリックして、クラスターの[Associated IAM roles] (アソシエートされた IAM ロール) のリストに追加します。

5. [クラスターを作成] を選択します。

Amazon Redshift 入門ガイドの手順に従って、SQL クライアントからクラスターに接続し、その接続をテストします。使用開始の残りの手順を完了して、テーブルの作成、データのアップロード、サンプルクエリの試行を行う必要はありません。

ステップ 2: データファイルをダウンロードする

このステップでは、コンピュータに一連のサンプルデータファイルをダウンロードします。次のステップでは、Amazon S3 バケットにファイルをアップロードします。

データファイルをダウンロードするには

1. zip ファイル LoadingDataSampleFiles.zip をダウンロードします。
2. お使いのコンピュータのフォルダにファイルを展開します。
3. 以下のファイルがフォルダに含まれていることを確認します。

```
customer-fw-manifest
customer-fw.tbl-000
customer-fw.tbl-000.bak
customer-fw.tbl-001
customer-fw.tbl-002
customer-fw.tbl-003
customer-fw.tbl-004
customer-fw.tbl-005
customer-fw.tbl-006
customer-fw.tbl-007
customer-fw.tbl.log
dwdate-tab.tbl-000
```

```
dwdate-tab.tbl-001
dwdate-tab.tbl-002
dwdate-tab.tbl-003
dwdate-tab.tbl-004
dwdate-tab.tbl-005
dwdate-tab.tbl-006
dwdate-tab.tbl-007
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

ステップ 3: Amazon S3 バケットにファイルをアップロードする

このステップでは、Amazon S3 バケットを作成し、データファイルをバケットにアップロードします。

Amazon S3 バケットにファイルをアップロードするには

1. Amazon S3 にバケットを作成します。

   バケットの作成の詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Creating a bucket」(バケットの作成) を参照してください。

   a. AWS Management Console にサインインし、Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。

   b. [バケットの作成] を選択します。

   c. [AWS リージョン] を選択します。

      クラスターと同じリージョンでバケットを作成します。使用しているクラスターが米国西部 (オレゴン) リージョンにある場合は、[US West (Oregon) Region (us-west-2)] (米国西部 (オレゴン) リージョン (us-west-2)) を選択します。

   d. [バケットを作成] ダイアログボックスの [バケット名] ボックスに、バケットの名前を入力します。

バケット名は必ず、Amazon S3 内の既存バケット名の中で一意となるようにしてください。一意性を確実にする方法の 1 つは、バケット名を組織名で始めることです。バケット名は一定の規則に沿って命名する必要があります。詳細については、Amazon Simple Storage Service ユーザーガイドのバケットの制約と制限を参照してください。

e. 残りのオプションについては、推奨デフォルトを選択します。

f. [バケットを作成] を選択します。

Amazon S3 が正常にバケットを作成すると、コンソールが [Buckets (バケット)] パネルに空のバケットを表示します。

2. フォルダを作成します。

a. 新しいバケットの名前を選択します。

b. [フォルダを作成] ボタンを選択します。

c. 新しいフォルダに **load** という名前を付けます。

> **Note**
>
> 作成したバケットは、サンドボックスの中にはありません。この演習では、実際のバケットにオブジェクトを追加します。オブジェクトをバケットに格納する時間に対して、名目上の料金が発生します。Amazon S3 の料金に関する詳細については、Amazon S3 の料金を参照してください。

3. データファイルを新しい Amazon S3 バケットにアップロードします。

a. データフォルダの名前を選択します。

b. [アップロード] ウィザードで、[ファイルを追加] をクリックします。

Amazon S3 コンソールの指示に従い、ダウンロードおよび展開したすべてのファイルをアップロードします。

c. [アップロード] を選択します。

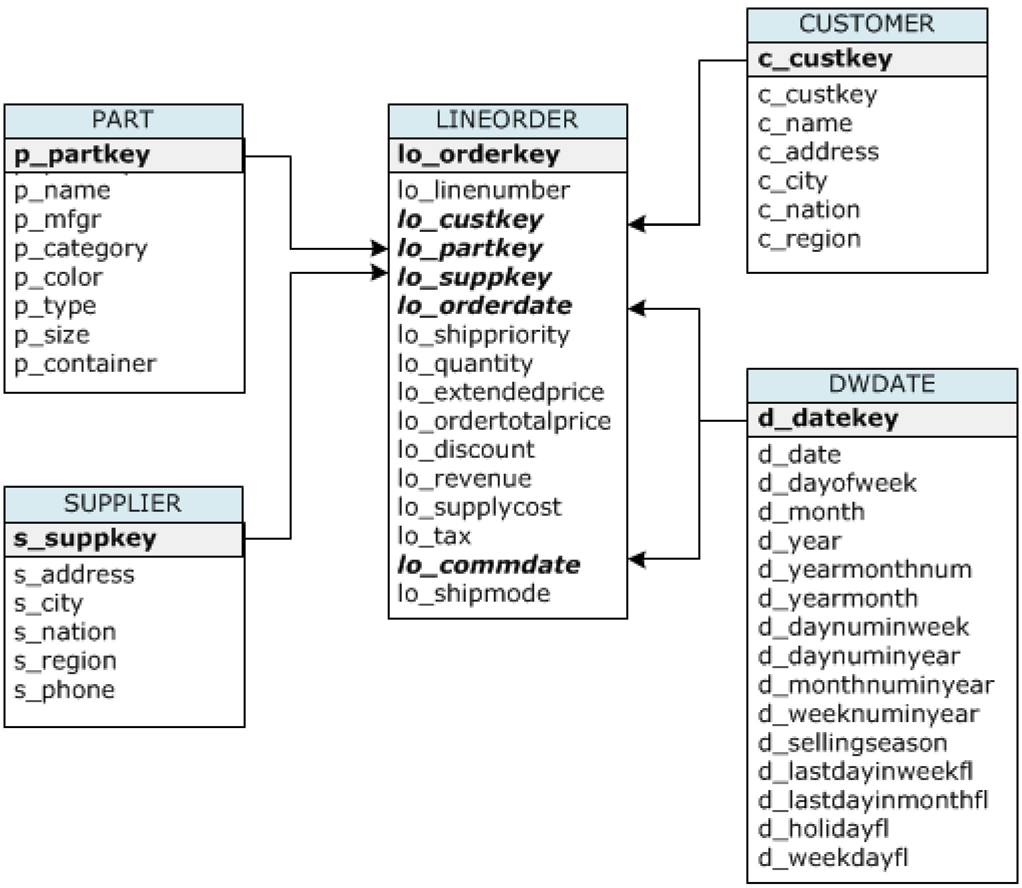
ユーザー認証情報

Amazon Redshift の COPY コマンドでは、Amazon S3 バケットにあるファイルオブジェクトを読み込むためのアクセス権が必要です。Amazon S3 バケットを作成したときと同じユーザー認証情報を使用して Amazon Redshift の COPY コマンドを実行する場合、COPY コマンドには必要なす

べてのアクセス許可があります。異なるユーザーの認証情報を使用する場合は、Amazon S3 のアクセスコントロールを使用して、アクセスを許可できます。Amazon Redshift の COPY コマンドでは、Amazon S3 バケット内のファイルオブジェクトにアクセスするために、少なくとも ListBucket と GetObject のアクセス許可が必要です。Amazon S3 リソースへのアクセスコントロールの詳細については、[Amazon S3 リソースへのアクセス許可の管理](#)を参照してください。

ステップ 4: サンプルテーブルを作成する

このチュートリアルでは、スタースキーマベンチマーク (SSB) スキーマに基づいた 5 つのテーブルをセットとして使用します。以下の図に示しているのは SSB データモデルです。



SSB テーブルは現在のデータベースにすでに存在している場合があります。その場合は、テーブルをドロップしてデータベースから削除してから、次の手順で CREATE TABLE コマンドを使用してテーブルを作成します。このチュートリアルで使用されるテーブルには、既存のテーブルとは異なる属性が含まれている可能性があります。

サンプルテーブルを作成するには

1. SSB テーブルをドロップするには、SQL クライアントで以下のコマンドを実行します。

```
drop table part cascade;
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

2. SQL クライアントで以下の CREATE TABLE コマンドを実行します。

```
CREATE TABLE part
(
  p_partkey     INTEGER NOT NULL,
  p_name        VARCHAR(22) NOT NULL,
  p_mfgr        VARCHAR(6),
  p_category    VARCHAR(7) NOT NULL,
  p_brand1      VARCHAR(9) NOT NULL,
  p_color       VARCHAR(11) NOT NULL,
  p_type        VARCHAR(25) NOT NULL,
  p_size        INTEGER NOT NULL,
  p_container   VARCHAR(10) NOT NULL
);

CREATE TABLE supplier
(
  s_suppkey   INTEGER NOT NULL,
  s_name      VARCHAR(25) NOT NULL,
  s_address   VARCHAR(25) NOT NULL,
  s_city      VARCHAR(10) NOT NULL,
  s_nation    VARCHAR(15) NOT NULL,
  s_region    VARCHAR(12) NOT NULL,
  s_phone     VARCHAR(15) NOT NULL
);

CREATE TABLE customer
(
  c_custkey      INTEGER NOT NULL,
  c_name         VARCHAR(25) NOT NULL,
  c_address      VARCHAR(25) NOT NULL,
  c_city         VARCHAR(10) NOT NULL,
  c_nation       VARCHAR(15) NOT NULL,
  c_region       VARCHAR(12) NOT NULL,
  c_phone        VARCHAR(15) NOT NULL,
  c_mktsegment   VARCHAR(10) NOT NULL
```

```
);

CREATE TABLE dwdate
(
  d_datekey            INTEGER NOT NULL,
  d_date               VARCHAR(19) NOT NULL,
  d_dayofweek          VARCHAR(10) NOT NULL,
  d_month              VARCHAR(10) NOT NULL,
  d_year               INTEGER NOT NULL,
  d_yearmonthnum       INTEGER NOT NULL,
  d_yearmonth          VARCHAR(8) NOT NULL,
  d_daynuminweek       INTEGER NOT NULL,
  d_daynuminmonth      INTEGER NOT NULL,
  d_daynuminyear       INTEGER NOT NULL,
  d_monthnuminyear     INTEGER NOT NULL,
  d_weeknuminyear      INTEGER NOT NULL,
  d_sellingseason      VARCHAR(13) NOT NULL,
  d_lastdayinweekfl    VARCHAR(1) NOT NULL,
  d_lastdayinmonthfl   VARCHAR(1) NOT NULL,
  d_holidayfl          VARCHAR(1) NOT NULL,
  d_weekdayfl          VARCHAR(1) NOT NULL
);
CREATE TABLE lineorder
(
  lo_orderkey          INTEGER NOT NULL,
  lo_linenumber        INTEGER NOT NULL,
  lo_custkey           INTEGER NOT NULL,
  lo_partkey           INTEGER NOT NULL,
  lo_suppkey           INTEGER NOT NULL,
  lo_orderdate         INTEGER NOT NULL,
  lo_orderpriority     VARCHAR(15) NOT NULL,
  lo_shippriority      VARCHAR(1) NOT NULL,
  lo_quantity          INTEGER NOT NULL,
  lo_extendedprice     INTEGER NOT NULL,
  lo_ordertotalprice   INTEGER NOT NULL,
  lo_discount          INTEGER NOT NULL,
  lo_revenue           INTEGER NOT NULL,
  lo_supplycost        INTEGER NOT NULL,
  lo_tax               INTEGER NOT NULL,
  lo_commitdate        INTEGER NOT NULL,
  lo_shipmode          VARCHAR(10) NOT NULL
);
```

ステップ 5: COPY コマンドを実行する

COPY コマンドを実行して、SSB のスキーマの各テーブルをロードします。この COPY コマンドの例は、さまざまなファイル形式からのロード、COPY コマンドのオプションの使用、およびロードエラーのトラブルシューティングの方法を示しています。

COPY コマンドの構文

基本的な COPY コマンドの構文は次のとおりです。

```
COPY table_name [ column_list ] FROM data_source CREDENTIALS access_credentials
 [options]
```

COPY コマンドを実行するには、以下の値を指定します。

テーブル名

COPY コマンドのターゲットテーブル。テーブルはすでにデータベースに存在する必要があります。テーブルは一時テーブルまたは永続的テーブルです。COPY コマンドは、新しい入力データをテーブルの既存の行に追加します。

列リスト

デフォルトでは、COPY はソースデータのフィールドを順番にテーブルの列にロードします。オプションで、列名のカンマ区切りリストである列リストを指定して、データフィールドを特定の列にマッピングすることができます。このチュートリアルでは列リストは使用しません。詳細については、COPY コマンドのリファレンスの「Column List」を参照してください。

データソース

COPY コマンドを使用して、Amazon S3 バケット、Amazon EMR クラスター、リモート ホスト (SSH 接続を使用)、または Amazon DynamoDB テーブルからデータをロードできます。このチュートリアルでは、Amazon S3 バケットのデータファイルからロードします。Amazon S3 からロードする際、バケット名とデータファイルの場所を指定する必要があります。これを行うには、データファイルのオブジェクトパス、または各データファイルとその場所を明示的に一覧表示するマニフェストファイルの場所を指定します。

- キープレフィックス

  Amazon S3 に保存されたオブジェクトはオブジェクトキーによって一意に識別されます。オブジェクトキーには、バケット名、フォルダ名 (存在する場合)、およびオブジェクト名が含まれま

す。キープレフィックスは、同じプレフィックスを持つ一連のオブジェクトを指します。オブジェクトパスは、キープレフィックスを共有するすべてのオブジェクトをロードするために、COPY コマンドで使用するキープレフィックスです。たとえば、キープレフィックス `custdata.txt` は、単一のファイルを指す場合も、`custdata.txt.001`、`custdata.txt.002` など、一連のファイルを指す場合もあります。

- マニフェストファイル

  場合によっては、複数のバケットやフォルダなどから、異なる接頭辞を持つファイルをロードする必要があります。また、接頭辞を共有するファイルを除外する必要がある場合もあります。これらの場合には、マニフェストファイルを使用できます。マニフェストファイルは、ロードする各ファイルとその一意のオブジェクトキーを明示的にリストします。このチュートリアルでは、マニフェストファイルを使用して PART テーブルをロードします。

認証情報

ロードするデータが格納されている AWS リソースにアクセスするには、十分な権限を持つユーザーの AWS アクセス認証情報を指定する必要があります。これらの認証情報には IAM ロール の Amazon リソースネーム (ARN) が含まれます。Amazon S3 からデータをロードするには、認証情報に ListBucket と GetObject のアクセス許可が含まれている必要があります。データが暗号化されている場合は、追加の認証情報が必要です。詳細については、COPY コマンドのリファレンスの「認可パラメータ」を参照してください。アクセスの管理の詳細については、Amazon S3 リソースへの許可の管理を参照してください。

オプション

COPY コマンドで多くのパラメータを指定することによって、ファイル形式の指定、データ形式の管理、エラーの管理、およびその他の機能の制御を行うことができます。このチュートリアルでは、次のような COPY コマンドのオプションおよび機能を使用します。

- キープレフィックス

  キープレフィックスを指定して複数のファイルからロードする方法については、「NULL AS を使用した PART テーブルのロード」を参照してください。

- CSV 形式

  CSV 形式のデータをロードする方法については、「NULL AS を使用した PART テーブルのロード」を参照してください。

- NULL AS

NULL AS オプションを使用して PART をロードする方法については、「NULL AS を使用した PART テーブルのロード」を参照してください。

- 文字区切り形式

  DELIMITER オプションの使用方法については、「REGION を使用した SUPPLIER テーブルのロード」を参照してください。
- REGION

  REGION オプションの使用方法については、「REGION を使用した SUPPLIER テーブルのロード」を参照してください。
- 固定幅形式

  固定幅データから CUSTOMER テーブルをロードする方法については、「MANIFEST を使用した CUSTOMER テーブルのロード」を参照してください。
- MAXERROR

  MAXERROR オプションの使用方法については、「MANIFEST を使用した CUSTOMER テーブルのロード」を参照してください。
- ACCEPTINVCHARS

  ACCEPTINVCHARS オプションの使用方法については、「MANIFEST を使用した CUSTOMER テーブルのロード」を参照してください。
- MANIFEST

  MANIFEST オプションの使用方法については、「MANIFEST を使用した CUSTOMER テーブルのロード」を参照してください。
- DATEFORMAT

  DATEFORMAT オプションの使用方法については、「DATEFORMAT を使用した DWDATE テーブルのロード」を参照してください。
- GZIP、LZOP および BZIP2

  ファイルを圧縮する方法については、「複数ファイルを使用した LINEORDER テーブルのロード」を参照してください。
- COMPUPDATE

COMPUPDATE オプションの使用方法については、「複数ファイルを使用した LINEORDER テーブルのロード」を参照してください。

- 複数のファイル

  複数のファイルをロードする方法については、「複数ファイルを使用した LINEORDER テーブルのロード」を参照してください。

SSB テーブルのロード

SSB スキーマの各テーブルをロードするには、次の COPY コマンドを使用します。各テーブルに対するコマンドは、COPY のさまざまなオプションとトラブルシューティングの手法を示しています。

SSB テーブルをロードするには、以下の手順に従います。

1. バケット名と AWS 認証情報を置き換える
2. NULL AS を使用した PART テーブルのロード
3. REGION を使用した SUPPLIER テーブルのロード
4. MANIFEST を使用した CUSTOMER テーブルのロード
5. DATEFORMAT を使用した DWDATE テーブルのロード
6. 複数ファイルを使用した LINEORDER テーブルのロード

バケット名と AWS 認証情報を置き換える

このチュートリアルの COPY コマンドは次の形式で表示されます。

```
copy table from 's3://<your-bucket-name>/load/key_prefix'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
options;
```

各 COPY コマンドで、以下の作業を行います。

1. *<your-bucket-name>* を、お使いのクラスターと同じリージョンにあるバケットの名前に置き換えます。

   このステップでは、バケットとクラスターが同じリージョンにあることを前提としています。代わりに、COPY コマンドで REGION オプションを使用してリージョンを指定できます。

2. *<aws-account-id>* および *<role-name>* は、使用している AWS アカウント ならびに IAM ロールに置き換えます。一重引用符で囲まれた認証情報文字列に、スペースまたは改行を含めることはできません。ARN の形式はサンプルとは多少異なる場合があることに注意してください。COPY コマンドを実行するときは、正しい ARN を使用するために、IAM コンソールからロールの ARN をコピーすることをお勧めします。

NULL AS を使用した PART テーブルのロード

このステップでは、CSV オプションと NULL AS オプションを使用して、PART テーブルをロードします。

COPY コマンドでは、複数のファイルから並列してデータをロードでき、1 つのファイルからロードする場合よりも高速です。この原理を示すために、このチュートリアルでは、ファイルは非常に小さくなりますが、各テーブルのデータを 8 個のファイルに分割しています。後のステップで、1 つのファイルからのロードと複数のファイルからのロードとの時間の差を比較します。詳細については、「[データファイルをロードする](#)」を参照してください。

キープレフィックス

ファイルセットのキープレフィックスを指定するか、マニフェストファイルにファイルのリストを明示的に指定することで、複数のファイルからロードできます。このステップでは、キープレフィックスを使用します。後のステップでは、マニフェストファイルを使用します。キープレフィックス `'s3://amzn-s3-demo-bucket/load/part-csv.tbl'` によって、`load` フォルダ内の以下のファイルのセットがロードされます。

```
part-csv.tbl-000
part-csv.tbl-001
part-csv.tbl-002
part-csv.tbl-003
part-csv.tbl-004
part-csv.tbl-005
part-csv.tbl-006
part-csv.tbl-007
```

CSV 形式

CSV は、カンマ区切り値を意味し、スプレッドシートのデータをインポートおよびエクスポートする際に使用される一般的な形式です。CSV では、フィールド内に引用符で囲まれた文字列を含めることができるため、カンマ区切り形式よりも柔軟です。CSV 形式から COPY を実行する場合、デフォルトの引用文字は二重引用符 (") ですが、QUOTE AS オプションを使用して別の引用文字を指

定できます。フィールド内で引用符文字を使用する場合は、追加の引用符文字で文字をエスケープしてください。

PART テーブルの CSV 形式のデータファイルから抜粋した次のデータは、二重引用符で囲まれた文字列 (`"LARGE ANODIZED BRASS"`) を示しています。また、引用符で囲まれた文字列内の 2 つの二重引用符で囲まれた文字列 (`"MEDIUM ""BURNISHED"" TIN"`) が表示されます。

```
15,dark sky,MFGR#3,MFGR#47,MFGR#3438,indigo,"LARGE ANODIZED BRASS",45,LG CASE
22,floral beige,MFGR#4,MFGR#44,MFGR#4421,medium,"PROMO, POLISHED BRASS",19,LG DRUM
23,bisque slate,MFGR#4,MFGR#41,MFGR#4137,firebrick,"MEDIUM ""BURNISHED"" TIN",42,JUMBO
 JAR
```

PART テーブルのデータには、COPY が失敗する原因となる文字が含まれています。この演習では、エラーをトラブルシューティングし、修正します。

形式のデータをロードするには、COPY コマンド に csvcsv を追加 します。以下のコマンドを実行して、PART テーブルをロードします。

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv;
```

次のようなエラーメッセージが表示されます。

```
An error occurred when executing the SQL command:
copy part from 's3://amzn-s3-demo-bucket/load/part-csv.tbl'
credentials' ...

ERROR: Load into table 'part' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 1.46s

1 statement(s) failed.
1 statement(s) failed.
```

エラーに関する詳細情報を取得するには、STL_LOAD_ERRORS テーブルに対してクエリを実行します。次のクエリでは、列を短縮して読みやすくするために SUBSTRING 関数を使用し、返される行数を減らすために LIMIT 10 を使用しています。バケット名の長さに合わせて、`substring(filename,22,25)` の値を調整できます。

```
select query, substring(filename,22,25) as filename,line_number as line,
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as reason
from stl_load_errors
order by query desc
limit 10;
```

```
 query  |    filename        | line |  column    |    type    | pos |
--------+-------------------------+-----------+------------+------------+-----+----
 333765 | part-csv.tbl-000 |    1 |            |            |   0 |

 line_text        | field_text |                    reason
------------------+------------+----------------------------------------------
 15,NUL next,     |            | Missing newline: Unexpected character 0x2c f
```

NULL AS

`part-csv.tbl` データファイルでは、NUL ターミネータ文字 (`\x000` または `\x0`) を使用して NULL 値を表します。

> **Note**
>
> スペルやよく似ていますが、NUL と NULL は同じではありません。NUL は、コードポイントが `x000` である UTF-8 文字で、通常、レコードの終わり (EOR) を示すために使用されます。NULL はデータがないことを表す SQL 値です。

デフォルトでは、COPY は NUL ターミネータ文字を EOR 文字として処理し、レコードを終了します。これは通常、予期しない結果やエラーが発生する原因となります。テキストデータで NULL を示す標準的な方法はありません。したがって、NULL AS COPY コマンドオプションを使用すると、テーブルのロード時に NULL で置換する文字を指定できます。この例では、COPY で、NUL ターミネータ文字を NULL 値として処理する必要があります。

> **Note**
>
> NULL 値を受け取るテーブルの列は、NULL が許容されるように設定されている必要があります。つまり、CREATE TABLE の指定に NOT NULL 制約を含めないようにする必要があります。

NULL AS オプションを使って PART をロードするには、以下の COPY コマンドを実行します。

```
copy part from 's3://<your-bucket-name>/load/part-csv.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
csv
null as '\000';
```

COPY によって NULL 値がロードされたことを確認するには、以下のコマンドを実行して NULL が含まれる行のみを選択します。

```
select p_partkey, p_name, p_mfgr, p_category from part where p_mfgr is null;
```

```
 p_partkey |  p_name  | p_mfgr | p_category
-----------+----------+--------+------------
        15 | NUL next |        | MFGR#47
        81 | NUL next |        | MFGR#23
       133 | NUL next |        | MFGR#44
(2 rows)
```

REGION を使用した SUPPLIER テーブルのロード

このステップでは、DELIMITER オプションと REGION オプションを使用して、SUPPLIER テーブルをロードします。

> **Note**
>
> SUPPLIER テーブルを読み込むためのファイルは、AWS サンプルバケットに用意されています。このステップでファイルをアップロードする必要はありません。

文字区切り形式

文字区切りファイルのフィールドは、パイプ文字 (|)、カンマ (,)、タブ (\t) など、特定の文字で区切られます。文字区切りファイルでは、区切り記号として、非表示の ASCII 文字を含め、任意の ASCII 文字 1 文字を使用できます。DELIMITER オプションを使用して区切り文字を指定できます。デフォルトの区切り文字はパイプ文字です。

SUPPLIER テーブルから抜粋した次のデータでは、パイプ区切り形式を使用しています。

```
1|1|257368|465569|41365|19950218|2-HIGH|0|17|2608718|9783671|4|2504369|92072|2|
19950331|TRUCK
1|2|257368|201928|8146|19950218|2-HIGH|0|36|6587676|9783671|9|5994785|109794|6|
19950416|MAIL
```

REGION

可能な限り、ロードデータは Amazon Redshift クラスターと同じ AWS リージョンに配置してください。データとクラスターが同じリージョンにある場合、レイテンシーが短縮され、リージョン間のデータ転送のコストを回避できます。詳細については、「データをロードするための Amazon Redshift のベストプラクティス」を参照してください。

別の AWS リージョンからデータをロードする必要がある場合は、REGION オプションを使用して、そのロードデータが配置されている AWS リージョンを指定します。リージョンを指定する場合は、マニフェストファイルを含むすべてのロードデータが、指定されたリージョンに存在している必要があります。詳細については、「REGION」を参照してください。

クラスターが米国東部 (バージニア北部) リージョンにある場合は、以下のコマンドを実行して米国西部 (オレゴン) リージョンにある Amazon S3 バケット内のパイプ区切りデータから SUPPLIER テーブルをロードします。この例では、バケット名を変更しないでください。

```
copy supplier from 's3://awssampledbuswest2/ssbgz/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '|'
gzip
region 'us-west-2';
```

クラスターが米国東部 (バージニア北部) リージョンにない場合は、以下のコマンドを実行して米国東部 (バージニア北部) リージョンにある Amazon S3 バケット内のパイプ区切りデータから SUPPLIER テーブルをロードします。この例では、バケット名を変更しないでください。

```
copy supplier from 's3://awssampledb/ssbgz/supplier.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

```
delimiter '|'
gzip
region 'us-east-1';
```

MANIFEST を使用した CUSTOMER テーブルのロード

このステップでは、FIXEDWIDTH、MAXERROR、ACCEPTINVCHARS、および MANIFEST オプションを使用して、CUSTOMER テーブルをロードします。

この演習のサンプルデータには、COPY でロードしようとしたときにエラーの原因となる文字が含まれています。MAXERRORS オプションと STL_LOAD_ERRORS システムテーブルを使用してロードエラーのトラブルシューティングを行い、次に ACCEPTINVCHARS オプションと MANIFEST オプションを使用してエラーを排除します。

固定幅形式

固定幅形式は、区切り記号でフィールドを分離するのではなく、各フィールドを固定文字数で定義します。CUSTOMER テーブルから抜粋した次のデータでは、固定幅形式を使用しています。

```
1   Customer#000000001   IVhzIApeRb           MOROCCO  0MOROCCO  AFRICA      25-705
2   Customer#000000002   XSTf4,NCwDVaWNe6tE   JORDAN   6JORDAN   MIDDLE EAST 23-453
3   Customer#000000003   MG9kdTD              ARGENTINA5ARGENTINAAMERICA     11-783
```

ラベル/幅のペアの順序はテーブルの列の順序に正確に一致する必要があります。詳細については、「FIXEDWIDTH」を参照してください。

CUSTOMER テーブルのデータの固定幅指定文字列は、次のとおりです。

```
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
c_region :12, c_phone:15,c_mktsegment:10'
```

固定幅データから CUSTOMER テーブルをロードするには、以下のコマンドを実行します。

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10';
```

次のようなエラーメッセージが表示されます。

```
An error occurred when executing the SQL command:
copy customer
from 's3://amzn-s3-demo-bucket/load/customer-fw.tbl'
credentials'...

ERROR: Load into table 'customer' failed.  Check 'stl_load_errors' system table for
 details. [SQL State=XX000]

Execution time: 2.95s

1 statement(s) failed.
```

MAXERROR

デフォルトでは、COPY で最初にエラーが発生したときに、コマンドは失敗し、エラーメッセージを返します。テスト中は時間を節約するために、MAXERROR オプションを使用して、失敗する前に指定した数のエラーをスキップするように COPY に指示できます。CUSTOMER テーブルのデータのロードを最初にテストするときにはエラーが予想されるため、COPY コマンドに `maxerror 10` を追加します。

FIXEDWIDTH オプションと MAXERROR オプションを使用してテストするには、以下のコマンドを実行します。

```
copy customer
from 's3://<your-bucket-name>/load/customer-fw.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10;
```

今回は、エラーメッセージの代わりに、次のような警告メッセージが表示されます。

```
Warnings:
Load into table 'customer' completed, 112497 record(s) loaded successfully.
Load into table 'customer' completed, 7 record(s) could not be loaded.  Check
 'stl_load_errors' system table for details.
```

この警告は、COPY で 7 個のエラーが発生したことを示します。エラーを確認するには、次の例のように、STL_LOAD_ERRORS テーブルに対してクエリを実行します。

```
select query, substring(filename,22,25) as filename,line_number as line,
```

```
substring(colname,0,12) as column, type, position as pos, substring(raw_line,0,30) as
 line_text,
substring(raw_field_value,0,15) as field_text,
substring(err_reason,0,45) as error_reason
from stl_load_errors
order by query desc, filename
limit 7;
```

STL_LOAD_ERRORS のクエリの結果は次のようになるはずです。

```
 query  |         filename          | line |  column    |    type     | pos |
 line_text            | field_text |              error_reason
--------+---------------------------+------+-----------
+------------+-----+------------------------------+------------
+----------------------------------------------
 334489 | customer-fw.tbl.log       |    2 | c_custkey | int4       |  -1 | customer-
fw.tbl               | customer-f | Invalid digit, Value 'c', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    6 | c_custkey | int4       |  -1 | Complete
                   | Complete   | Invalid digit, Value 'C', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    3 | c_custkey | int4       |  -1 | #Total rows
                   | #Total row | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    5 | c_custkey | int4       |  -1 | #Status
                   | #Status    | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl.log       |    1 | c_custkey | int4       |  -1 | #Load file
                   | #Load file | Invalid digit, Value '#', Pos 0, Type: Integ
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
 334489 | customer-fw.tbl000        |    1 | c_address | varchar    |  34 | 1
 Customer#000000001  | .Mayag.ezR | String contains invalid or unsupported UTF8
(7 rows)
```

この結果を調べると、`error_reasons` 列に 2 件のメッセージがあることがわかります。

-
  ```
  Invalid digit, Value '#', Pos 0, Type: Integ
  ```

  これらのエラーは、`customer-fw.tbl.log` ファイルによって発生しています。問題は、このファイルがデータファイルではなくログファイルであることです。このファイルがロードされないようにする必要があります。マニフェストファイルを使用して、誤ったファイルがロードされることを回避できます。

-
  ```
  String contains invalid or unsupported UTF8
  ```

VARCHAR データ型は、最大 3 バイトのマルチバイト UTF-8 文字をサポートします。ロードデータにサポートされていない文字や無効な文字が含まれている場合、ACCEPTINVCHARS オプションを使用して、すべての無効な文字を指定した代替文字に置き換えることができます。

負荷に関するもう一つの問題は、検出がより難しく、負荷が予期せぬ結果を生み出しました。この問題を調査するには、以下のコマンドを実行して、CUSTOMER テーブルをクエリします。

```
select c_custkey, c_name, c_address
from customer
order by c_custkey
limit 10;
```

```
 c_custkey |          c_name           |         c_address
-----------+---------------------------+---------------------------
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         2 | Customer#000000002        | XSTf4,NCwDVaWNe6tE
         3 | Customer#000000003        | MG9kdTD
         3 | Customer#000000003        | MG9kdTD
         4 | Customer#000000004        | XxVSJsL
         4 | Customer#000000004        | XxVSJsL
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         5 | Customer#000000005        | KvpyuHCplrB84WgAi
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
         6 | Customer#000000006        | sKZz0CsnMD7mp4Xd0YrBvx
(10 rows)
```

行は一意である必要がありますが、重複があります。

予期しない結果を検証するもう 1 つの方法は、ロードされた行数を検証することです。今回のケースでは、100000 行がロードされているはずですが、ロードのメッセージには 112497 件のレコードがロードされたと示されています。ロードされた行が多いのは、COPY で余分なファイル `customer-fw.tbl0000.bak` がロードされたためです。

この演習では、マニフェストファイルを使用して、誤ったファイルがロードされることを回避します。

ACCEPTINVCHARS

デフォルトでは、COPY は列のデータ型でサポートされていない文字を検出した場合、その行をスキップし、エラーを返します。無効な UTF-8 文字の詳細については、「マルチバイト文字のロードエラー」を参照してください。

MAXERRORS オプションによってエラーを無視してロードを続行し、STL_LOAD_ERRORS にクエリを実行して無効な文字を特定した後、データファイルを修正できます。ただし、MAXERRORS はロードの問題のトラブルシューティングには最適ですが、一般的に、本番環境では使用しないでください。

ACCEPTINVCHARS オプションは、通常、無効な文字を管理するのに適しています。ACCEPTINVCHARS によって、無効な文字を指定した有効な文字で置き換え、ロード操作を続行することを COPY コマンドに指示します。置換文字として、NULL 以外の有効な ASCII 文字を指定できます。デフォルトの置換文字は疑問符 (?) です。COPY は、マルチバイト文字を同じ長さの置換文字列に置き換えます。たとえば、4 バイト文字は `'????'` に置き換えられます。

COPY は、無効な UTF-8 文字を含む行の数を返します。また、影響を受ける行ごとに STL_REPLACEMENTS システムテーブルにエントリを追加します (ノードスライスあたり最大 100 行)。さらに多くの無効な UTF-8 文字も置き換えられますが、それらの置換イベントは記録されません。

ACCEPTINVCHARS は VARCHAR 列に対してのみ有効です。

このステップでは、ACCEPTINVCHARS と置換文字 `'^'` を追加します。

MANIFEST

キープレフィックスを使用して Amazon S3 からの COPY を実行すると、不要なテーブルをロードするリスクがあります。たとえば、`'s3://amzn-s3-demo-bucket/load/` フォルダには、キープレフィックス `customer-fw.tbl` を共有する 8 個のデータファイルが含まれます (`customer-fw.tbl0000`、`customer-fw.tbl0001` など)。ただし、同じフォルダには、余分なファイルである `customer-fw.tbl.log` や `customer-fw.tbl-0001.bak` も含まれています。

必要なすべてのファイル、および正しいファイルのみをロードするには、マニフェストファイルを使用します。マニフェストは JSON 形式のテキストファイルで、ロードされる各ソースファイルについて一意のオブジェクトキーを明示的にリストします。ファイルオブジェクトは、異なるフォルダや異なるバケットにあってもかまいませんが、同じリージョンに存在している必要があります。詳細については、「MANIFEST」を参照してください。

以下に `customer-fw-manifest` のテキストを示します。

```
{
  "entries": [
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-000"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-001"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-002"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-003"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-004"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-005"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-006"},
    {"url":"s3://<your-bucket-name>/load/customer-fw.tbl-007"}
    ]
}
```

マニフェストファイルを使用して CUSTOMER テーブルのデータをロードするには

1. テキストエディタで `customer-fw-manifest` ファイルを開きます。
2. *<your-bucket-name>* の部分はお客様のバケットの名前に置き換えます。
3. ファイルを保存します。
4. バケットのロードフォルダにファイルをアップロードします。
5. 以下の COPY コマンドを実行します。

```
copy customer from 's3://<your-bucket-name>/load/customer-fw-manifest'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
fixedwidth 'c_custkey:10, c_name:25, c_address:25, c_city:10, c_nation:15,
 c_region :12, c_phone:15,c_mktsegment:10'
maxerror 10
acceptinvchars as '^'
manifest;
```

DATEFORMAT を使用した DWDATE テーブルのロード

このステップでは、DELIMITER オプションと DATEFORMAT オプションを使用して、DWDATE テーブルをロードします。

DATE 列と TIMESTAMP 列をロードする場合、COPY ではデフォルトの形式を想定しています。日付の場合は YYYY-MM-DD で、タイムスタンプの場合は YYYY-MM-DD HH:MI:SS です。ロードデータでデフォルトの形式が使用されていない場合、DATEFORMAT と TIMEFORMAT を使用して形式を指定できます。

次の例は、DWDATE テーブルの日付形式を示しています。列 2 の日付形式が一貫していないことに注意してください。

```
19920104 1992-01-04          Sunday  January 1992 199201 Jan1992 1 4 4 1...
19920112 January 12, 1992 Monday  January 1992 199201 Jan1992 2 12 12 1...
19920120 January 20, 1992 Tuesday    January 1992 199201 Jan1992 3 20 20 1...
```

DATEFORMAT

日付形式は 1 つだけ指定できます。ロードデータに一貫性のない形式が含まれている場合やその形式が異なる列に含まれている場合、または形式がロード時にわからない場合は、DATEFORMAT を 'auto' 引数と共に使用します。'auto' を指定すると、COPY は有効な日付または時間形式を認識してデフォルト形式に変換します。'auto' オプションは、DATEFORMAT および TIMEFORMAT 文字列を使用する場合にサポートされない形式を認識します。詳細については、「DATEFORMAT と TIMEFORMAT で自動認識を使用する」を参照してください。

DWDATE テーブルをロードするには、以下の COPY コマンドを実行します。

```
copy dwdate from 's3://<your-bucket-name>/load/dwdate-tab.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
delimiter '\t'
dateformat 'auto';
```

複数ファイルを使用した LINEORDER テーブルのロード

このステップでは、GZIP オプションと COMPUPDATE オプションを使用して、LINEORDER テーブルをロードします。

この演習では、LINEORDER テーブルを単一のデータファイルからロードし、次に、複数のファイルからロードします。これにより、2 つの方法のロード時間を比較できます。

> **Note**
>
> LINEORDER テーブルを読み込むためのファイルは、AWS サンプルバケットに用意されています。このステップでファイルをアップロードする必要はありません。

GZIP、LZOP および BZIP2

ファイルは gzip、lzop、または bzip2 圧縮形式を使って圧縮できます。圧縮ファイルからロードする場合、COPY はロードの過程でファイルを解凍します。ファイルを圧縮すると、ストレージ領域を節約し、アップロード時間を短縮できます。

COMPUPDATE

COPY は、圧縮エンコードなしで空のテーブルをロードする場合、ロードデータを分析し、最適なエンコードを決定します。次に、ロードを開始する前にそのエンコードを使用するようにテーブルを変更します。この分析プロセスは時間がかかりますが、分析が行われるのはテーブルごとに多くて 1 回です。時間を節約するには、COMPUPDATE をオフにすることにより、このステップを省略できます。COPY の実行時間を正確に評価するために、このステップでは COMPUPDATE をオフにします。

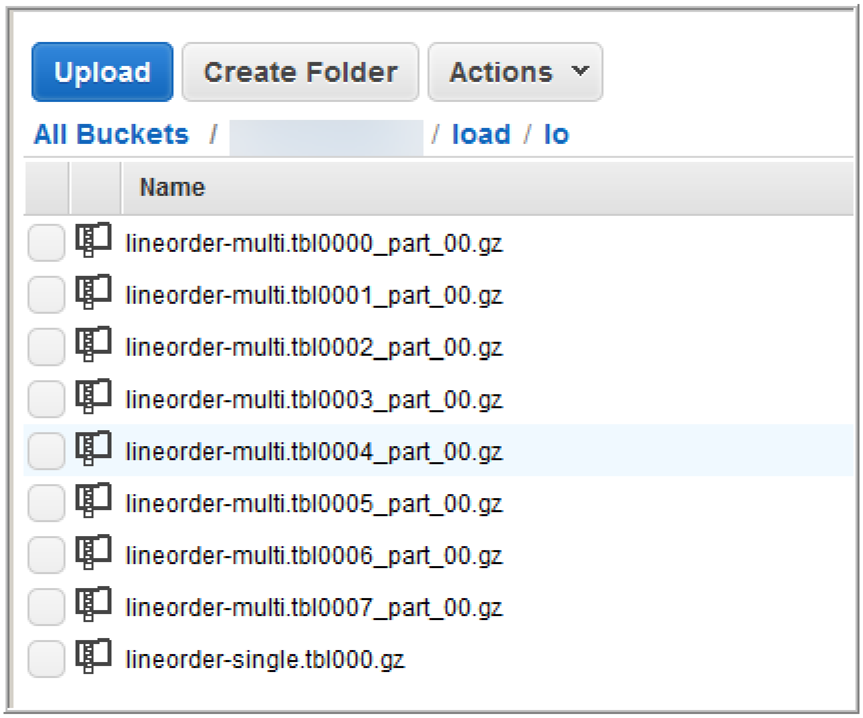
複数のファイル

COPY コマンドでは、1 つのファイルからロードする代わりに、複数のファイルから並列でロードすると、データを非常に効率的にロードできます。ファイルの数がクラスターのスライスの数の倍数になるようにデータをファイルに分割します。そうすることで Amazon Redshift はワークロードを分割し、スライス間で均等にデータを配分します。ノードあたりのスライスの数は、クラスターのノードサイズによって決まります。各ノードサイズに含まれるスライス数の詳細については、「Amazon Redshift 管理ガイド」の「クラスターおよびノードについて」を参照してください。

例えば、このチュートリアルで使用される dc2.large コンピューティングノードにはそれぞれ 2 個のスライスがあるため、4 ノードのクラスターに 8 個のスライスがあります。前の手順では、ロードデータが非常に小さい 8 つのファイルに格納されていました。このステップでは、1 つの大きいファイルからのロードと複数のファイルからのロードとの時間の差を比較します。

このチュートリアルで使用するファイルには、約 1,500 万件のレコードが含まれ、約 1.2 GB の領域を使用します。これらのファイルは Amazon Redshift のスケールでは非常に小さいファイルですが、複数のファイルからロードする際のパフォーマンス上の利点を示すには十分です。これらのファイルは非常に大きいため、このチュートリアルで、ファイルをダウンロードして Amazon S3 にアップロードするには時間がかかりすぎます。このため、AWS サンプルバケットからファイルを直接ロードします。

次のスクリーンショットは LINEORDER のデータ ファイルを示しています。



複数のファイルを使用して COPY のパフォーマンスを評価するには

1. 単一のファイルから COPY を実行するには、以下のコマンドを実行します。バケット名を変更しないでください。

```
copy lineorder from 's3://awssampledb/load/lo/lineorder-single.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

2. 結果は次のようになります。実行時間を書き留めてください。

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 51.56s
```

3. 複数のファイルから COPY を実行するには、以下のコマンドを実行します。バケット名を変更しないでください。

```
copy lineorder from 's3://awssampledb/load/lo/lineorder-multi.tbl'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
gzip
compupdate off
region 'us-east-1';
```

4. 結果は次のようになります。実行時間を書き留めてください。

```
Warnings:
Load into table 'lineorder' completed, 14996734 record(s) loaded successfully.

0 row(s) affected.
copy executed successfully

Execution time: 17.7s
```

5. 実行時間を比較します。

   この例では、1,500 万件のレコードをロードする時間が、51.56 秒から 17.7 秒へと短縮され、65.7% の削減を達成しました。

   これらの結果は 4 ノードクラスターの使用に基づいています。クラスターにさらにノードがある場合は、時間の節約は増加します。数十個から数百個のノードがある、一般的な Amazon Redshift クラスターの場合、違いはさらに明確になります。単一ノードクラスターの場合、実行時間の差はほとんどありません。

ステップ 6: データベースにバキューム操作を実行し、分析する

相当数の行を追加、削除、変更するたびに、VACUUM コマンドを実行してから ANALYZE コマンドを実行する必要があります。バキューム操作によって、削除された行から領域を回復し、ソート順序を復元します。ANALYZE コマンドは統計メタデータを更新し、クエリオプティマイザがさらに正確なクエリプランを生成できるようにします。詳細については、「テーブルのバキューム処理」を参照してください。

ソートキー順序でデータをロードする場合、バキューム処理は高速です。このチュートリアルでは、かなりの行数を追加しましたが、空のテーブルに追加しました。このような場合、再ソートは必要ではなく、行を削除していません。COPY は空のテーブルをロードした後に自動的に統計を更新するため、統計は最新の状態になっているはずです。ただし、適切なハウスキーピングを行うため、データベースにバキューム処理を実行し、分析することで、このチュートリアルを完了します。

データベースをバキュームして分析するには、以下のコマンドを実行します。

```
vacuum;
analyze;
```

ステップ 7: リソースをクリーンアップする

クラスターが実行されている限り料金が発生し続けます。このチュートリアルの完了後は、Amazon Redshift 入門ガイドの「ステップ 5: アクセスを取り消してサンプルクラスターを削除する」の手順に従って、環境を以前の状態に戻す必要があります。

クラスターを維持するが、SSB テーブルで使用されるストレージを復元したいという場合は、以下のコマンドを実行します。

```
drop table part;
drop table supplier;
drop table customer;
drop table dwdate;
drop table lineorder;
```

次へ

[概要]

[概要]

このチュートリアルでは、データファイルを Amazon S3 にアップロードした後、COPY コマンドを使用してデータをファイルから Amazon Redshift テーブルにロードしました。

次の形式を使用してデータをロードしました。

- 文字区切り
- CSV
- 固定幅

STL_LOAD_ERRORS システムテーブルを使用してロードエラーをトラブルシューティングし、次に REGION、MANIFEST、MAXERROR、ACCEPTINVCHARS、DATEFORMAT、および NULL AS オプションを使用してエラーを修正しました。

データのロードに関する以下のベストプラクティスを適用しました。

- COPY コマンドを使用したデータのロード
- データファイルをロードする
- 単一の COPY コマンドを使用した複数のファイルからのロード
- データファイルを圧縮する
- ロードの前後におけるデータファイルの検証

Amazon Redshift のベストプラクティスの詳細については、次のリンクを参照してください。

- データをロードするための Amazon Redshift のベストプラクティス
- Amazon Redshift テーブル設計のベストプラクティス
- Amazon Redshift クエリの設計のベストプラクティス

Amazon Redshift でのデータのアンロード

データベーステーブルからデータをアンロードして Amazon S3 バケットの一連のファイルに出力するには、UNLOAD コマンドを SELECT ステートメントとともに使用します。テキストデータは、ロード時に使用されたデータ形式にかかわらず、区切り文字付き形式と固定幅形式のどちらでもアンロードできます。圧縮された GZIP ファイルを作成するかどうかを指定することもできます。

Amazon S3 バケットに対するユーザーのアクセスを制限するには、一時的なセキュリティ認証情報を使用します。

トピック

データを Amazon S3 にアンロードする

Amazon Redshift には、select ステートメントの結果を分割して一連のファイルに出力する機能があります。ノードスライスごとに 1 つ以上のファイルが作成されるので、データの並列再ロードが容易になります。または、PARALLEL OFF オプションを追加することで、UNLOAD が 1 つ以上のファイルに結果を順次に書き込めるように指定できます。MAXFILESIZE パラメータを指定して、Amazon S3 のファイルサイズを制限できます。UNLOAD は、Amazon S3 サーバー側暗号化 (SSE-S3) 機能を使用してデータを自動的に暗号化します。

Amazon Redshift がサポートする UNLOAD コマンドでは、任意の select ステートメントを使用できます (ただし、外側の select で LIMIT 句を使用するものを除きます)。例えば、select ステートメントの中で特定の列を選択することや、where 句を使用して複数のテーブルを結合することができます。クエリの中に引用符がある (例えば、リテラル値を囲むため) 場合は、クエリテキスト内でエスケープする (\') 必要があります。詳細については、SELECT コマンドのリファレンスを参照してください。LIMIT 句の使用方法の詳細については、「使用に関する注意事項」で UNLOAD コマンドの使用に関する注意事項を参照してください。

たとえば、次に示す UNLOAD コマンドを実行すると、VENUE テーブルの内容が Amazon S3 バケット `s3://amzn-s3-demo-bucket/tickit/unload/` に送信されます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

前の例で作成されたファイル名には、'venue_' というプレフィックスが付いています。

```
venue_0000_part_00
venue_0001_part_00
venue_0002_part_00
venue_0003_part_00
```

デフォルトでは、UNLOAD は、クラスター内のスライスの数に応じて、データを複数のファイルに同時に書き込みます。1 つのファイルにデータを書き込むには、PARALLEL OFF を指定します。UNLOAD はデータを順次に書き込みます。データは ORDER BY 句 (使用されている場合) に従ってソートされます。データファイルの最大サイズは 6.2 GB です。データサイズが最大値よりも大きい場合、UNLOAD は追加のファイルをそれぞれ 6.2 GB を上限として作成します。

以下の例では、コンテンツ VENUE を 1 つのファイルに書き込んでいます。1 つのファイルのみが必要なのは、ファイルサイズが 6.2 GB より小さいためです。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

> **Note**
>
> UNLOAD コマンドは、並列処理を使用するように設計されています。特別な理由がなく、特にファイルが COPY コマンドを使用してテーブルをロードするために使用される場合には、PARALLEL を有効にしたままにすることをお勧めします。

VENUE のデータサイズの合計が 5 GB とすると、次に例では、VENUE のコンテンツを各 100 MB サイズの 50 ファイルに書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off
```

```
maxfilesize 100 mb;
```

Amazon S3 パス文字列の中にプレフィックスがある場合は、UNLOAD 実行時にそのプレフィックスがファイル名に使用されます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/unload/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

UNLOAD コマンドで MANIFEST オプションを指定することで、アンロードファイルをリストするマニフェストファイルを作成できます。マニフェストは、Amazon S3 に書き込まれた各ファイルの URL を明示的にリストする、JSON 形式のテキストファイルです。

次の例では MANIFEST オプションが含めています。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

次の例は、4 個のアンロードファイルについてのマニフェストを示します。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0000_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0001_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0002_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0003_part_00"}
  ]
}
```

COPY コマンドで MANIFEST オプションを使用することで、マニフェストファイルを使用して同じファイルをロードできます。詳細については、「マニフェストを使用し、データファイルを指定する」を参照してください。

UNLOAD オペレーションが完了したら、データが正しくアンロードされたことを確認します。確認するには、UNLOAD でファイルを出力した Amazon S3 バケットに移動します。スライスごとに 1 つ以上のファイルがあり、ファイルには 0 から始まる番号が付けられています。MANIFEST オプションを指定した場合、'manifest' で終わるファイルも表示されます。次に例を示します。

```
amzn-s3-demo-bucket/tickit/venue_0000_part_00
amzn-s3-demo-bucket/tickit/venue_0001_part_00
amzn-s3-demo-bucket/tickit/venue_0002_part_00
amzn-s3-demo-bucket/tickit/venue_0003_part_00
amzn-s3-demo-bucket/tickit/venue_manifest
```

Amazon S3 に書き込まれたファイルのリストをプログラムで取得するには、UNLOAD 完了後に Amazon S3 リストのオペレーションを呼び出します。STL_UNLOAD_LOG をクエリすることもできます。

次のクエリは、UNLOAD によって作成されたファイルのパス名を返します。PG_LAST_QUERY_ID 関数は、最新のクエリを返します。

```
select query, substring(path,0,40) as path
from stl_unload_log
where query=2320
order by path;

query |             path
-------+--------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

データ量が非常に多い場合、Amazon Redshift ではファイルが 1 つのスライスにつき複数の部分に分割されることがあります。次に例を示します。

```
venue_0000_part_00
venue_0000_part_01
venue_0000_part_02
venue_0001_part_00
venue_0001_part_01
venue_0001_part_02
...
```

次に示す UNLOAD コマンドでは、SELECT ステートメントの中に引用符で囲まれた文字列があるため、その引用符はエスケープされています (=\'OH\' ')。

```
unload ('select venuename, venuecity from venue where venuestate=\'OH\' ')
```

```
to 's3://amzn-s3-demo-bucket/tickit/venue/ '
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

デフォルトでは、出力先のバケットにすでにファイルがある場合はファイルが上書きされるのではなく UNLOAD が異常終了します。マニフェストファイルを含めて既存のファイルを上書きするには、ALLOWOVERWRITE オプションを指定します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
allowoverwrite;
```

暗号化されたデータファイルをアンロードする

UNLOAD は、AWS が管理する暗号化キー (SSE-S3) を使用した Amazon S3 のサーバー側暗号化により、ファイルを自動的に作成します。AWS Key Management Service キーを使用したサーバー側の暗号化 (SSE-KMS)、またはカスタマーマネージド型キーを使用したクライアント側の暗号化を指定することもできます。UNLOAD では、お客様が管理のキー による Amazon S3 サーバー側の暗号化は使用できません。詳細については、「[サーバー側の暗号化を使用したデータの保護]」を参照してください。

AWS KMS キーを使用したサーバー側の暗号化によって Amazon S3 にアンロードするには、次の例に示すように KMS_KEY_ID パラメータを使用してキー ID を指定します。

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
KMS_KEY_ID '1234abcd-12ab-34cd-56ef-1234567890ab'
encrypted;
```

独自の暗号化キーを指定する場合は、ENCRYPTED オプションを指定した UNLOAD コマンドを使用することにより、Amazon S3 でクライアント側の暗号化データファイルを作成できます。UNLOAD で使用されるエンベロープ暗号化プロセスは、Amazon S3 のクライアント側暗号化で使用されているものと同じです。この暗号化されたファイルをロードするには、COPY コマンドとともに ENCRYPTED オプションを使用します。

このプロセスの動作は次のようになります。

1. ユーザーは、base64 で暗号化した 256 ビット AES キーを作成し、これをプライベート暗号化キー (ルート対称キー) として使用します。
2. 作成したルート対称キーと ENCRYPTED オプションを指定しながら、UNLOAD コマンドを実行します。
3. UNLOAD により、1 回限り使用の対称キー (エンベロープ対称キーと呼ばれます) と初期化ベクター (IV) が生成され、これが UNLOAD でのデータの暗号化に使用されます。
4. 作成したルート対称キーを使用してエンベロープ対称キーが暗号化されます。
5. 暗号化されたデータファイルが Amazon S3 に保存され、暗号化されたエンベロープキーと IV がオブジェクトメタデータとして各ファイルとともに保存されます。暗号化されたエンベロープキーはオブジェクトメタデータ `x-amz-meta-x-amz-key` として、IV はオブジェクトメタデータ `x-amz-meta-x-amz-iv` として保存されます。

エンベロープ暗号化プロセスの詳細については、「Client-side data encryption with the AWS SDK for Java and Amazon S3」の記事を参照してください。

暗号化されたデータファイルをアンロードするには、ルートキーの値を認証情報文字列に追加するとともに、ENCRYPTED オプションを指定します。MANIFEST オプションを使用すると、マニフェストファイルも暗号化されます。

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
manifest
encrypted;
```

アンロードしようとする暗号化済みデータファイルが GZIP 圧縮されている場合は、GZIP オプションをルートキーの値および ENCRYPTED オプションとともに指定します。

```
unload ('select venuename, venuecity from venue')
to 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted gzip;
```

暗号化されたデータファイルをロードするには、同じルートキーの値を含めながら MASTER_SYMMETRIC_KEY パラメータを追加し、さらに ENCRYPTED オプションを指定します。

```
copy venue from 's3://amzn-s3-demo-bucket/encrypted/venue_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key '<root_key>'
encrypted;
```

区切り文字付きまたは固定幅形式でデータをアンロードする

データのアンロードは、区切り文字付き形式と固定幅形式のどちらでも可能です。デフォルトでは、出力はパイプ文字 (|) で区切られます。

次に示す例では、カンマを区切り文字として指定しています。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/comma'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',';
```

生成される出力ファイルは次のようになります。

```
20,Air Canada Centre,Toronto,ON,0
60,Rexall Place,Edmonton,AB,0
100,U.S. Cellular Field,Chicago,IL,40615
200,Al Hirschfeld Theatre,New York City,NY,0
240,San Jose Repertory Theatre,San Jose,CA,0
300,Kennedy Center Opera House,Washington,DC,0
...
```

同じ結果セットをアンロードしてタブ区切りファイルに出力するには、次のコマンドを実行します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/tab'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

代わりに、FIXEDWIDTH 指定を使用することもできます。この指定は、各テーブル列の識別子と、その列の幅 (文字数) で構成されます。幅が不足している場合は、データが切り捨てられるのではなく UNLOAD コマンドが異常終了するので、指定する幅は、その列の最も長いエントリの長さ以上となるようにしてください。固定幅データのアンロードの動作は、区切り文字付きのデータのアンロー

ドに似ています。異なるのは、生成される出力の中に区切り文字が含まれていない点です。次に例を示します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/tickit/venue/fw'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth '0:3,1:100,2:30,3:2,4:6';
```

固定幅の出力は次のようになります。

```


20 Air Canada Centre         Toronto      ON0
60 Rexall Place              Edmonton     AB0
100U.S. Cellular Field       Chicago      IL40615
200Al Hirschfeld Theatre     New York CityNY0
240San Jose Repertory TheatreSan Jose     CA0
300Kennedy Center Opera HouseWashington   DC0
```

FIXEDWIDTH 仕様についての詳細は、UNLOAD コマンドを参照してください。

アンロードしたデータを再ロードする

アンロード操作の結果を再ロードするには、COPY コマンドを使用します。

次に示す単純な例では、マニフェストファイルを使用して VENUE テーブルをアンロードし、テーブルの全データを削除してから、再ロードしています。

```
unload  ('select * from venue order by venueid')
to 's3://amzn-s3-demo-bucket/tickit/venue/reload_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';

truncate venue;

copy venue
from 's3://amzn-s3-demo-bucket/tickit/venue/reload_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest
delimiter '|';
```

再ロードの後の VENUE テーブルは次のようになります。

```
select * from venue order by venueid limit 5;

 venueid |         venuename          |  venuecity  | venuestate | venueseats
---------+---------------------------+-------------+------------+-----------
       1 | Toyota Park                | Bridgeview  | IL         |          0
       2 | Columbus Crew Stadium      | Columbus    | OH         |          0
       3 | RFK Stadium                | Washington  | DC         |          0
       4 | CommunityAmerica Ballpark | Kansas City | KS         |          0
       5 | Gillette Stadium           | Foxborough  | MA         |      68756
(5 rows)
```

Amazon Redshift のユーザー定義関数

カスタムスカラーのユーザー定義関数 (UDF) は、SQL SELECT 句または Python プログラムを使用して作成できます。新しい関数はデータベースに保存され、適切な実行権限を持つすべてのユーザーが使用できます。既存の Amazon Redshift 関数を実行するのとほぼ同じ方法で、カスタムスカラー UDF を実行します。

Python UDF の場合、標準の Python 機能を使用できるだけではなく、独自のカスタム Python モジュールをインポートできます。詳細については、「UDF のための Python 言語のサポート」を参照してください。Python 3 は Python UDF では使用できないことに注意してください。Amazon Redshift UDF に対する Python 3 のサポートを取得するには、スカラー Lambda UDF を代わりに使用します。

また、SQL クエリの一部として Lambda で定義されたカスタム関数を使用する、AWS Lambda UDF を作成することもできます。Lambda UDF を使用すると、複雑な UDF を作成し、サードパーティーのコンポーネントと統合できます。また、現在の Python および SQL UDF 制限のいくつかを克服することもできます。たとえば、ネットワークリソースやストレージリソースにアクセスし、より本格的な SQL ステートメントを作成することができます。Lambda UDF は、Java、Go、PowerShell、Node.js、C#、Python、Ruby など、Lambda でサポートされている任意のプログラミング言語で作成できます。または、カスタムランタイムを使用できます。

デフォルトでは、すべてのユーザーが UDF を実行できます。権限の詳細については、「UDF のセキュリティとアクセス許可」を参照してください。

トピック

UDF のセキュリティとアクセス許可

UDF を作成するには、SQL または plpythonu (Python) 用の言語で使用のアクセス権限を持っている必要があります。デフォルトでは、USAGE ON LANGUAGE SQL が PUBLIC に許可されますが、特

定のユーザーまたはグループに明示的に USAGE ON LANGUAGE PLPYTHONU を許可する必要があります。

SQL の使用を取り消すには、最初に PUBLIC に対して使用を取り消します。次に、SQL UDF の作成を許可された特定のユーザーやグループにのみ、SQL の使用を許可します。次の例では、PUBLIC から SQL での使用を取り消します。次に、ユーザーグループ udf_devs に使用を許可します。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

UDF を実行するには、関数ごとに実行許可を持っている必要があります。デフォルトでは、新しい UDF を実行する許可が PUBLIC に付与されます。使用を制限するには、関数に対して PUBLIC からこの許可を取り消します。次に、特定の個人またはグループに権限を付与します。

次の例では、PUBLIC から関数 f_py_greater での実行を取り消します。次に、ユーザーグループ udf_devs に使用を許可します。

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

スーパーユーザーは、デフォルトですべての権限を持っています。

詳細については、「GRANT」と「REVOKE」を参照してください。

UDF 名の競合の回避

UDF を実装する前に UDF の命名規則を考慮することにより、競合を防止したり、予期せぬ結果を回避したりできます。関数名は多重定義が可能であるため、現在使用されている、および将来使用される Amazon Redshift 関数名と重複する可能性があります。このトピックでは、多重定義について説明し、競合を回避する方法を示します。

関数名の多重定義

関数は、その名前と、入力引数の数および引数のデータ型で構成される署名によって識別されます。同じスキーマ内に名前は同じで署名は異なる 2 つの関数が存在できます。つまり、関数名は多重定義が可能です。

クエリを実行するとき、クエリエンジンは、指定されている引数の数と引数のデータ型に基づいて、呼び出す関数を決定します。多重定義を使用すると、CREATE FUNCTION コマンドで許容される限界まで、可変の引数を持つ関数をシミュレートできます。

組み込み Amazon Redshift 関数との競合の回避

プレフィックス `f_` を使用して、すべての UDF に名前を付けることをお勧めします。Amazon Redshift は、UDF 専用のプレフィックスとして `f_` を予約しており、UDF 名にプレフィックスとして `f_` を使用することで、UDF 名が現在使用されているか、使用される予定の Amazon Redshift 組み込み SQL 関数名と競合することを回避できます。例えば、新しい UDF に `f_sum` という名前を付けることで、Amazon Redshift SUM 関数との競合を回避できます。同様に、新しい関数に `f_fibonacci` という名前を付けると、将来の Amazon Redshift のリリースで FIBONACCI という名前の関数が追加された場合に競合を回避できます。

既存の Amazon Redshift 組み込み SQL 関数が異なるスキーマに存在する場合に限り、関数名の多重定義を発生させることなく、同じ名前および署名の UDF を作成できます。組み込み関数はシステムカタログスキーマである pg_catalog に存在するので、パブリックスキーマやユーザー定義のスキーマなどの別のスキーマに同じ名前の UDF を作成できます。場合によっては、スキーマ名で明示的に修飾されていない関数を呼び出すことがあります。この場合、Amazon Redshift はデフォルトで最初に pg_catalog スキーマを検索します。したがって、組み込み関数は、同じ名前の新しい UDF の前に実行されます。

この動作は、末尾に pg_catalog を配置するように検索パスを設定することによって変更できます。その場合、UDF は組み込み関数よりも優先されますが、この方法では予期しない結果が生じる可能性があります。予約されている `f_` プレフィックスを使用するなど、名前の重複を回避する方法を採用したほうがより確実です。詳細については、「SET」と「search_path」を参照してください。

スカラー SQL UDF

スカラー SQL UDF には、関数が呼び出されて単一の値を返すときに実行される SQL SELECT 句が組み込まれています。CREATE FUNCTION コマンドは以下のパラメータを定義します。

- (省略可能) 入力引数。各引数にデータ型が存在する必要があります。
- 1 つの戻りデータ型。
- 1 つの SQL SELECT 句。SELECT 句で、関数定義の引数の順序に従って、$1、$2 などを使用して入力引数を参照します。

入力および戻りデータタイプは、任意の標準 Amazon Redshift データタイプを使用できます。

SELECT 句には FROM 句を含めないでください。代わりに、SQL UDF を呼び出す FROM 句を SQL ステートメントに含めます。

SELECT 句には、以下のタイプの句を含めることはできません。

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- 制限

スカラー SQL 関数の例

次の例は、2 つの数値を比較し、大きいほうの数値を返す関数を作成する方法を示しています。詳細については、「CREATE FUNCTION」を参照してください。

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

次のクエリは、新しい f_sql_greater 関数を呼び出して SALES テーブルをクエリし、COMMISSION または PRICEPAID の 20% のどちらか大きいほうを返します。

```
select f_sql_greater(commission, pricepaid*0.20) from sales;
```

スカラー Python UDF

スカラー Python UDF には、関数が呼び出されると実行され、単一値を返す Python プログラムが組み込まれています。CREATE FUNCTION コマンドは以下のパラメータを定義します。

- (省略可能) 入力引数。各引数に名前とデータ型がある必要があります。
- 1 つの戻りデータ型。
- 1 つの実行可能な Python プログラム。

入力データ型および戻りデータ型に
SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE
PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、または TIMESTAMP を使用できます。また、Python UDF は ANYELEMENT データタイプを使用できます。このタイプは、Amazon Redshift によって、ランタイム時に示される引数のデータタイプに基づく標準データタイプに自動的に変換されます。詳細については、「ANYELEMENT データ型」を参照してください。

Amazon Redshift クエリがスカラー UDF を呼び出すと、以下のステップがランタイム時に発生します。

1. 関数が入力引数を Python データ型に変換します。

   Amazon Redshift データ型と Python データ型のマッピングについては、「Python UDF データ型」を参照してください。
2. 関数が Python プログラムを実行し、変換した入力引数を渡します。
3. Python コードが単一値を返します。戻り値のデータ型は、関数定義で指定されている RETURNS データ型と対応している必要があります。
4. 関数が Python の戻り値を、指定されている Amazon Redshift データ型に変換してから、その値をクエリに返します。

> **Note**
>
> Python 3 は Python UDF では使用できません。Amazon Redshift UDF に対する Python 3 のサポートを取得するには、スカラー Lambda UDF を代わりに使用します。

スカラー Python UDF の例

次の例は、2 つの数値を比較し、大きいほうの数値を返す関数を作成する方法を示しています。二重ドル記号 ($$) の間にあるコードのインデントは Python の要件です。詳細については、「CREATE FUNCTION」を参照してください。

```
create function f_py_greater (a float, b float)
  returns float
stable
as $$
  if a > b:
    return a
```

```
  return b
$$ language plpythonu;
```

次のクエリは、新しい `f_greater` 関数を呼び出して SALES テーブルをクエリし、COMMISSION または PRICEPAID の 20% のどちらか大きいほうを返します。

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

Python UDF データ型

Python UDF の入力引数および戻り値には標準の Amazon Redshift データ型を使用できます。標準のデータ型に加えて、UDF ではデータ型 ANYELEMENT をサポートします。これは、Amazon Redshift が実行時に提供される引数に基づいて標準のデータ型に自動的に変換します。スカラー UDF は、ANYELEMENT のデータ型を返すことができます。詳細については、「ANYELEMENT データ型」を参照してください。

実行中に、Amazon Redshift は、処理を行うために引数を Amazon Redshift データ型から Python データ型に変換します。次に、戻り値を Python データ型から対応する Amazon Redshift データ型に変換します。Amazon Redshift のデータ型の詳細については、「データ型」を参照してください。

次の表は Amazon Redshift データ型と Python データ型のマッピングを示しています。

Amazon Redshift のデータ型	Python データ型
smallint integer bigint short long	int
decimal または numeric	decimal
double real	float

Amazon Redshift のデータ型	Python データ型
boolean	ブール
char varchar	文字列
timestamp	datetime

ANYELEMENT データ型

ANYELEMENT は、ポリモーフィックなデータ型です。引数のデータ型として ANYELEMENT を使用して関数が宣言されている場合、その関数は、呼び出されたときに、標準の Amazon Redshift データ型をその引数の入力として受け入れることができます。ANYELEMENT 引数は、関数が呼び出されたときに、それに実際に渡されるデータ型に設定されます。

1 つの関数が複数の ANYELEMENT データ型を使用する場合、これらのデータ型のすべてが、関数が呼び出されたときに、同一の実際のデータ型に解決される必要があります。すべての ANYELEMENT 引数データ型が、ANYELEMENT に最初に渡される引数の実際のデータ型に設定されます。たとえば、`f_equal(anyelement, anyelement)` として宣言されている関数は、これら 2 つの入力値が同じデータ型である限り、両方の入力値をとります。

関数の戻り値が ANYELEMENT として宣言されている場合は、少なくとも 1 つの入力引数が ANYELEMENT である必要があります。戻り値の実際のデータ型は、ANYELEMENT 入力引数に指定されている実際のデータ型と同じになります。

UDF のための Python 言語のサポート

Python プログラミング言語に基づいてカスタム UDF を作成できます。Python 2.7 標準ライブラリは UDF で使用できますが、以下のモジュールは例外です。

- ScrolledText
- Tix
- Tkinter
- tk
- turtle
- smtpd

Python 標準ライブラリに加え、以下のモジュールも Amazon Redshift 実装の一部です。

- numpy 1.8.2
- pandas 0.14.1
- python-dateutil 2.2
- pytz 2014.7
- scipy 0.12.1
- six 1.3.0
- wsgiref 0.1.2

独自のカスタム Python モジュールをインポートし、ライブラリを作成する コマンドを実行することによって UDF で使用できるようにすることも可能です。詳細については、「例: カスタム Python ライブラリモジュールのインポート」を参照してください。

Important

Amazon Redshift は、UDF を介したすべてのネットワークアクセスとファイルシステムへの書き込みアクセスをブロックします。

Note

Python 3 は Python UDF では使用できません。Amazon Redshift UDF に対する Python 3 のサポートを取得するには、スカラー Lambda UDF を代わりに使用します。

例: カスタム Python ライブラリモジュールのインポート

スカラー関数は、Python 言語構文を使用して定義します。Python 標準ライブラリモジュールと Amazon Redshift に事前にインストールされたモジュールを使用できます。独自のカスタム Python ライブラリモジュールを作成してライブラリをクラスターにインポートしたり、Python またはサードパーティーの既存のライブラリを使用したりすることもできます。

Python 標準ライブラリモジュール、またはインストール済みの Amazon Redshift Python モジュールと同じ名前のモジュールを含むライブラリを作成することはできません。ユーザーがインストールした既存のライブラリが独自に作成するライブラリと同じ Python パッケージを使用している場合は、新しいライブラリをインストールする前に既存のライブラリを削除する必要があります。

カスタムライブラリをインストールするには、スーパーユーザーであるか、`USAGE ON LANGUAGE plpythonu` 権限を持っている必要がありますが、関数を作成するために十分な権限を持っているすべてのユーザーが、インストールされているライブラリを使用できます。PG_LIBRARY システムカタログをクエリすることによって、クラスターにインストールされているライブラリに関する情報を確認できます。

カスタム Python モジュールをクラスターにインポートする

このセクションでは、カスタム Python モジュールをクラスターにインポートする例を示します。このセクションの手順を実行するには、ライブラリパッケージをアップロードする Amazon S3 バケットを準備する必要があります。その後に、パッケージをクラスターにインストールします。バケット作成の詳細については、Amazon Simple Storage Service ユーザーガイドのバケットの作成を参照してください。

この例では、データ内の位置と距離を操作する UDF を作成するとします。SQL クライアントツールから Amazon Redshift クラスターに接続し、以下のコマンドを実行して関数を作成します。

```
CREATE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS float
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2)
$$ LANGUAGE plpythonu;

CREATE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float) RETURNS bool
 IMMUTABLE as $$
    def distance(x1, y1, x2, y2):
        import math
        return math.sqrt((y2 - y1) ** 2 + (x2 - x1) ** 2)

    return distance(x1, y1, x2, y2) < 20
$$ LANGUAGE plpythonu;
```

前の関数で数行のコードが重複していることを確認してください。この重複が必要な理由は、UDF が別の UDF のコンテンツを参照できず、両方の関数が同じ機能を必要とするからです。しかし、複数の関数でコードを重複させる代わりに、カスタムライブラリを作成し、関数がそれを使用するように設定できます。

これを行うには、まず次の手順に従ってライブラリパッケージを作成します。

1. geometry という名前のフォルダを作成します。このフォルダーはライブラリの最上位パッケージです。
2. geometry フォルダに `__init__.py` という名前のファイルを作成します。このファイル名には 2 つの二重下線文字が含まれています。このファイルは、パッケージが初期化可能であることを Python に対して示します。
3. geometry フォルダに trig という名前のフォルダを作成します。このフォルダーはライブラリのサブパッケージです。
4. trig フォルダに `__init__.py` ファイルをもう 1 つ作成し、`line.py` という名前のファイルを作成します。このフォルダー内の `__init__.py` は、サブパッケージが初期化可能であり、`line.py` がライブラリコードを含んでいるファイルであることを Python に対して示します。

   フォルダとファイル構造は、次のようにする必要があります。

   ```
   geometry/
      __init__.py
      trig/
         __init__.py
         line.py
   ```

   パッケージ構造についての詳細は、Python ウェブサイトの Python チュートリアルの「Modules」を参照してください。
5. 次のコードにはライブラリのクラスとメンバー関数が含まれています。これをコピーし、`line.py` に貼り付けます。

   ```
   class LineSegment:
     def __init__(self, x1, y1, x2, y2):
       self.x1 = x1
       self.y1 = y1
       self.x2 = x2
       self.y2 = y2
     def angle(self):
       import math
       return math.atan2(self.y2 - self.y1, self.x2 - self.x1)
     def distance(self):
       import math
       return math.sqrt((self.y2 - self.y1) ** 2 + (self.x2 - self.x1) ** 2)
   ```

パッケージの作成が完了したら、次の手順に従ってパッケージを準備し、Amazon S3 にアップロードします。

1. geometry フォルダの内容を圧縮して geometry.zip という名前の .zip ファイルを作成します。geometry フォルダ自体はこれに含めないでください。次に示すように、このフォルダの内容のみを含めます。

```
geometry.zip
   __init__.py
   trig/
      __init__.py
      line.py
```

2. geometry.zip を Amazon S3 バケットにアップロードします。

> **Important**
>
> Amazon S3 バケットが Amazon Redshift クラスターと同じリージョンに存在しない場合は、REGION オプションを使用して、データがあるリージョンを指定する必要があります。詳細については、「ライブラリを作成する」を参照してください。

3. SQL クライアントツールから次のコマンドを実行してライブラリをインストールします。*<bucket_name>* をバケットの名前に置き換え、*<access key id>* と *<secret key>* をそれぞれ、AWS Identity and Access Management (IAM) ユーザー認証情報のアクセスキーとシークレットアクセスキーに置き換えます。

```
CREATE LIBRARY geometry LANGUAGE plpythonu FROM 's3://<bucket_name>/geometry.zip'
 CREDENTIALS 'aws_access_key_id=<access key id>;aws_secret_access_key=<secret key>';
```

クラスターにライブラリをインストールしたら、関数がライブラリを使用するように設定する必要があります。これを行うには、以下のコマンドを実行します。

```
CREATE OR REPLACE FUNCTION f_distance (x1 float, y1 float, x2 float, y2 float) RETURNS
 float IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance()
$$ LANGUAGE plpythonu;
```

```
CREATE OR REPLACE FUNCTION f_within_range (x1 float, y1 float, x2 float, y2 float)
 RETURNS bool IMMUTABLE as $$
    from trig.line import LineSegment

    return LineSegment(x1, y1, x2, y2).distance() < 20
$$ LANGUAGE plpythonu;
```

前のコマンドでは、`import trig/line` が、このセクションの元の関数から、重複しているコードを排除します。このライブラリによって提供される機能は複数の UDF で再利用できます。モジュールをインポートするには、サブパッケージとモジュール名へのパスのみを指定する必要があるだけです (`trig/line`)。

Python UDF の制約

このトピックで示す制約が許す限り、Amazon Redshift 組み込みスカラー関数を使用する場所で UDF を使用できます。詳細については、「SQL 関数リファレンス」を参照してください。

Amazon Redshift Python UDF には以下の制約があります。

- Python UDF は、ネットワークにアクセスすることもファイルシステムに読み書きすることもできません。
- ユーザーがインストールする Python ライブラリの合計サイズは 100 MB を超えられません。
- Amazon Redshift は、自動ワークロード管理 (WLM) を使用するプロビジョニングされたクラスターとサーバーレスワークグループに対して、一度に 1 つの Python UDF しか実行できません。複数の UDF を同時に実行しようとすると、Amazon Redshift は残りの Python UDF をキューに入れ、ワークロード管理キューで実行します。自動 WLM を使用する場合、SQL UDF には同時実行数の制限はありません。
- プロビジョニングされたクラスターに手動 WLM を使用する場合、1 つのクラスターで同時に実行できる Python UDF の数は、そのクラスターの合計同時実行レベルの 4 分の 1 に制限されています。例えば、同時実行数が 15 のプロビジョニングされたクラスターでは、最大 3 つの同時 Python UDF を実行できます。
- Amazon Redshift で Python UDF を使用する場合、SUPER および HLLSKETCH データタイプはサポートされせん。

Python UDF のエラーと警告のログ記録

Python ロギングモジュールを使用して、ユーザー定義のエラーおよび警告メッセージを UDF 内に作成できます。クエリの実行に続いて、SVL_UDF_LOG システムビューをクエリしてログ記録されたメッセージを取得できます。

> **Note**
>
> UDF のログ記録はクラスターのリソースを消費し、システムのパフォーマンスに影響する場合があります。ログ記録は開発およびトラブルシューティングのためにのみ実装することをお勧めします。

クエリの実行中に、ログハンドラは、対応する関数名、ノード、およびスライスとともに、SVL_UDF_LOG システムビューにメッセージを書き込みます。ログハンドラはメッセージごと、スライスごとに 1 行を SVL_UDF_LOG に書き込みます。メッセージは 4096 バイトで切り捨てられます。UDF のログはスライスごとに 500 行に制限されます。ログがいっぱいになると、ログハンドラは古い方からメッセージを破棄し、SVL_UDF_LOG に警告メッセージを追加します。

> **Note**
>
> Amazon Redshift の UDF ログハンドラでは、改行 (\n)、パイプ (|) 文字、バックスラッシュ (\ ) 文字を、バックスラッシュ (\ ) 文字でエスケープします。

デフォルトでは、UDF ログレベルは WARNING に設定されます。ログレベルが WARNING、ERROR、および CRITICAL のメッセージがログ記録されます。重要度がより低い INFO、DEBUG、および NOTSET のメッセージは無視されます。UDF のログレベルを設定するには、Python のロガーメソッドを使用します。次の例では、ログレベルを INFO に設定します。

```
logger.setLevel(logging.INFO)
```

Python ロギングモジュールの使用の詳細については、「Python 用ロギング機能」を参照してください。

次の例は、Python ロギングモジュールをインポートし、ロガーをインスタンス化してエラーを記録する f_pyerror という関数を作成します。

```
CREATE OR REPLACE FUNCTION f_pyerror()
```

```
RETURNS INTEGER
VOLATILE AS
$$
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)
logger.info('Your info message here')
return 0
$$ language plpythonu;
```

次の例では、SVL_UDF_LOG をクエリして前の例でログ記録されたメッセージを表示します。

```
select funcname, node, slice, trim(message) as message
from svl_udf_log;

  funcname   | query | node | slice |   message
------------+-------+------+-------+------------------
  f_pyerror | 12345 |     1|     1 | Your info message here
```

スカラー Lambda UDF

Amazon Redshift は、AWS Lambda で定義されたカスタム関数を SQL クエリの一部として使用することができます。スカラー Lambda UDF は、Java、Go、PowerShell、Node.js、C#、Python、Ruby など、Lambda でサポートされている任意のプログラミング言語で記述できます。または、カスタムランタイムを使用できます。

CREATE EXTERNAL FUNCTION コマンドでは、次のパラメータが作成されます。

- (オプション) データ型を持つ引数のリスト。
- 1 つの戻りデータ型。
- Amazon Redshift によって呼び出される外部関数の 1 つの関数名。
- Amazon Redshift クラスターが Lambda を引き受けて呼び出すことが許可されている 1 つの IAM ロール。
- Lambda UDF によって呼び出される 1 つのLambda 関数名。

CREATE EXTERNAL FUNCTION の詳細については、「CREATE EXTERNAL FUNCTION」を参照してください。

この関数の入力および戻りデータタイプは、どの標準 Amazon Redshift データタイプでも可能です。

Amazon Redshift は、外部関数がバッチ処理された引数と結果を送受信できるようにします。

Lambda UDF は Lambda で定義および管理され、Amazon Redshift でこれらの UDF を呼び出すアクセス権限を制御できます。同じクエリ内で複数の Lambda 関数を呼び出すことも、同じ関数を複数回呼び出すこともできます。

スカラー関数がサポートされている SQL ステートメントの句で Lambda UDF を使用します。また、SELECT、UPDATE、INSERT、DELETE などの SQL ステートメントで Lambda UDF を使用することもできます。

> **Note**
>
> Lambda UDF を使用すると、Lambda サービスから追加料金が発生する可能性があります。そうするかどうかは、Lambda リクエスト数 (UDF 呼び出し) や Lambda プログラム実行の合計時間などの要因によって異なります。ただし、Amazon Redshift で Lambda UDF を使用するための追加料金はありません。AWS Lambda の料金の詳細については、AWS Lambda の料金を参照してください。
> Lambda リクエストの数は、Lambda UDF が使用される特定の SQL ステートメント句によって異なります。例えば、この関数が次の のような WHERE 句 で使用されているとします。
> `SELECT a, b FROM t1 WHERE lambda_multiply(a, b) = 64; SELECT a, b FROM t1 WHERE a*b = lambda_multiply(2, 32)`
> この場合、Amazon Redshift はそれぞれの最初の SELECT ステートメントを呼び出し、2 番目の SELECT ステートメントを 1 回だけ呼び出します。
> ただし、クエリの投影部分で UDF を使用すると、結果セット内の修飾された行または集約された行ごとに Lambda 関数を 1 回だけ呼び出す可能性があります。

UDF のセキュリティとアクセス許可

Lambda UDF を作成するには、LANGUAGE EXFUNC の使用許可があることを確認します。特定のユーザー、グループ、またはパブリックに対して、USAGE ON LANGUAGE EXFUNC を明示的に付与するか、USAGE ON LANGUAGE EXFUNC を取り消す必要があります。

次の例では、EXFUNC での使用を PUBLIC に付与します。

```
grant usage on language exfunc to PUBLIC;
```

次の例では、最初に PUBLIC に対して exfunc の使用を取り消し、次にユーザーグループ lambda_udf_devs に使用を許可します。

```
revoke usage on language exfunc from PUBLIC;
grant usage on language exfunc to group lambda_udf_devs;
```

Lambda UDF を実行するには、呼び出される各関数に対するアクセス許可があることを確認してください。デフォルトでは、新しい Lambda UDF を実行する許可が PUBLIC に付与されます。使用を制限するには、関数に対して PUBLIC からこの許可を取り消します。次に、特定のユーザーまたはグループに権限を付与します。

次の例では、PUBLIC から関数 exfunc_sum での実行を取り消します。次に、ユーザーグループ lambda_udf_devs に使用を許可します。

```
revoke execute on function exfunc_sum(int, int) from PUBLIC;
grant execute on function exfunc_sum(int, int) to group lambda_udf_devs;
```

スーパーユーザーは、デフォルトですべての権限を持っています。

権限の付与と取り消しの詳細については、「GRANT」および「REVOKE」を参照してください。

Lambda UDF の認可パラメータの設定

CREATE EXTERNAL FUNCTION コマンドには、AWS Lambdaで Lambda 関数を呼び出すための認可が必要です。認可を行うには、CREATE EXTERNAL FUNCTION コマンドの実行時に、AWS Identity and Access Management (IAM) ロールを指定します。IAM ロールの詳細については、IAM ユーザーガイドの IAM ロールをご参照ください。

クラスターにアタッチされている Lambda 関数を呼び出す許可を持つ IAM ロールがある場合、コマンドの IAM_ROLE パラメータにロールの Amazon リソースネーム (ARN) を置き換えることができます。以下のセクションでは、CREATE EXTERNAL FUNCTION コマンドで IAM ロールを使用するステップについて説明します。

Lambda 用に IAM ロールを作成する

IAM ロールには、Lambda 関数を呼び出すための許可が必要です。IAM ロールの作成時に、次のいずれかの方法でアクセス許可を提供します。

- IAM ロールの作成中に、[Attach permissions policy (アクセス許可ポリシーのアタッチ)] ページで AWSLambdaRole ポリシーをアタッチします。AWSLambdaRole ポリシーは、最小要件である Lambda 関数を呼び出すためのアクセス許可を付与します。詳細およびその他のポリシーについては、AWS Lambda デベロッパーガイドの AWS Lambda のアイデンティティベースの IAM ポリシーを参照してください。
- すべてのリソースの lambda:InvokeFunction 許可、またはその関数の ARN を持つ特定の Lambda 関数を使用して、IAM ロールにアタッチする独自のカスタムポリシーを作成します。ポリシー作成方法の詳細については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

  次のポリシー例では、特定の Lambda 関数で Lambda を呼び出すことができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

  Lambda 関数のリソースの詳細については、「IAM API リファレンス」の「Lambda アクションのリソースと条件」を参照してください。

  必要なアクセス許可を使用してカスタムポリシーを作成した後、IAM ロールの作成中に、[Attach permissions policy (アクセス許可ポリシーのアタッチ)] ページでポリシーを IAM ロールにアタッチできます。

IAM ロールを作成する手順については、「Amazon Redshift 管理ガイド」の「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを許可する」を参照してください。

新しい IAM ロールを作成しない場合は、前述したアクセス許可を既存の IAM ロールに追加できます。

クラスターを使用して IAM ロールを関連付ける

IAM ロールをクラスターにアタッチします。クラスターにロールを追加するか、Amazon Redshift マネジメントコンソール、CLI、または API を使用してクラスターに関連付けられるロールを表示できます。詳細については、「Amazon Redshift 管理ガイド」の「IAM ロールをクラスターと関連付ける」を参照してください。

コマンドに IAM ロールを含める

CREATE EXTERNAL FUNCTION コマンドに IAM ロール ARN を含めます。IAM ロールを作成する場合、IAM はロールの Amazon リソースネーム (ARN) を返します。IAM ロールを指定するには、`IAM_ROLE` パラメータでロールの ARN を指定します。以下に、`IAM_ROLE` パラメータの構文を示します。

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

同じリージョン内の他のアカウントに存在する Lambda 関数を呼び出すには、Amazon Redshift での IAM ロールの連鎖を参照してください。

Amazon Redshift と AWS Lambda 間で JSON インターフェイスを使用する

Amazon Redshift は、Amazon Redshift が通信するすべての Lambda 関数に共通のインターフェイスを使用します。

次のテーブルに、JSON ペイロードで期待できる指定済みの Lambda 関数の入力フィールド一覧を示しています。

フィールド名	説明	値の範囲
request_id	各呼び出しリクエストを一意に識別する汎用一意識別子 (UUID)。	有効な UUID。
クラスター	クラスターのフル Amazon リソースネーム（ARN）。	有効なクラスター ARN。

フィールド名	説明	値の範囲
ユーザー	コールを発信するユーザーの名前。	有効なユーザー名
データベース	クエリが実行されているデータベースの名前。	有効なデータベース名。
external_function	呼び出しを行う外部関数の完全修飾名。	有効な完全修飾関数名。
query_id	呼び出しを行うクエリのクエリ ID。	有効なクエリ ID。
num_records	ペイロードでの引数の数。	1～2^64 の値。
引数	指定した形式のデータペイロード。	配列形式のデータは JSON 配列である必要があります。各要素は、引数の数が 1 より大きい場合、配列であるレコードです。配列を使用することで、Amazon Redshift はペイロード内でのレコードの順序を保持します。

JSON 配列の順序によって、バッチ処理の順序が決まります。Lambda 関数では、引数を繰り返し取得して、正確な数のレコードを生成する必要があります。ペイロードの例を次に示します。

```
{
  "request_id" : "23FF1F97-F28A-44AA-AB67-266ED976BF40",
  "cluster" : "arn:aws:redshift:xxxx",
  "user" : "adminuser",
  "database" : "db1",
  "external_function": "public.foo",
  "query_id" : 5678234,
  "num_records" : 4,
  "arguments" : [
     [ 1, 2 ],
     [ 3, null],
```

```
    null,
    [ 4, 6]
  ]
}
```

Lambda 関数の戻り出力には、次のフィールドが含まれます。

フィールド名	説明	値の範囲
success	関数の成功または失敗表示。	`"true"` または `"false"` 値。
error_msg	成功値が `"false"` の場合 (関数が失敗した場合) のエラーメッセージです。それ以外の場合、このフィールドは無視されます。	有効なメッセージ。
num_records	ペイロードのレコード数。	1～2^64 の値。
結果	指定された形式の呼び出し結果。	該当なし

Lambda 関数出力の例を次に示します。

```
{
  "success": true,   // true indicates the call succeeded
  "error_msg" : "my function isn't working",  // shall only exist when success != true
  "num_records": 4,      // number of records in this payload
  "results" : [
     1,
     4,
     null,
     7
   ]
}
```

SQL クエリから Lambda 関数を呼び出す場合、Amazon Redshift は次の点を考慮して、接続のセキュリティを確保します。

- GRANT および REVOKE 権限。UDF のセキュリティとアクセス許可の詳細については、「UDF のセキュリティとアクセス許可」を参照してください。
- Amazon Redshift は、指定された Lambda 関数に最低限のデータセットのみを送信します。
- Amazon Redshift は、指定された IAM ロールを持つ指定済みの Lambda 関数のみを呼び出します。

ユーザー定義関数 (UDF) のユースケース例

Amazon Redshift を他のコンポーネントと統合することによって、ビジネス問題の解決にユーザー定義関数を使用できるようになります。以下は、他のユーザーによるユースケースへの UDF の使用方法の例です。

- Accessing external components using Amazon Redshift Lambda UDFs (Amazon Redshift Lambda UDF を使用した外部コンポーネントへのアクセス) – Amazon Redshift Lambda UDF の仕組みと、Lambda UDF の作成手順を説明します。
- [Translate and analyze text using SQL functions with Amazon Redshift, Amazon Translate, and Amazon Comprehend] (Amazon Redshift、Amazon Translate、および Amazon Comprehend で SQL 関数を使用してテキストを翻訳および分析する) — あらかじめ構築された Amazon Redshift Lambda UDF を数回のクリックでインストールし、テキストフィールドの翻訳、編集、分析を行うことができます。
- Access Amazon Location Service from Amazon Redshift (Amazon Redshift から Amazon Location Service にアクセスする) – Amazon Redshift Lambda UDF を使用して Amazon Location Service と統合する方法を説明します。
- Data Tokenization with Amazon Redshift and Protegrity (Amazon Redshift と Protegrity を使用したデータのトークナイゼーション) – Amazon Redshift Lambda UDF を Protegrity Serverless 製品に統合する方法を説明します。
- Amazon Redshift UDF – Amazon Redshift SQL UDF、Lambda UDF、および Python UDF のコレクションです。

Amazon Redshift のストアドプロシージャの作成

このトピックでは、Amazon Redshift でストアドプロシージャを作成して使用する方法について説明します。ストアドプロシージャとは、複数のプログラムが使用できる SQL ステートメントのコレクションです。

Amazon Redshift のストアドプロシージャを PostgreSQL プロシージャ言語 PL/pgSQL を使用して定義し、一連の SQL クエリと論理オペレーションを実行できます。プロシージャはデータベースに保存され、適切なデータベース権限を持つすべてのユーザーが使用できます。

ユーザー定義関数 (UDF) とは異なり、ストアドプロシージャは SELECT クエリに加えてデータ定義言語 (DDL) とデータ操作言語 (DML) を組み込むことができます。ストアドプロシージャは値を返す必要がありません。ループ式や条件式などのプロシージャ言語を使用して論理フローを制御できます。

ストアドプロシージャを作成および管理するための SQL コマンドの詳細については、以下のコマンドトピックを参照してください。

- CREATE PROCEDURE
- ALTER PROCEDURE
- DROP PROCEDURE
- SHOW PROCEDURE
- CALL
- GRANT
- REVOKE
- ALTER DEFAULT PRIVILEGES

トピック

Amazon Redshift でのストアドプロシージャの概要

このトピックでは、ストアドプロシージャの目的と使用について詳しく説明します。

ストアドプロシージャは、通常、データ変換やデータ検証のロジック、ビジネス固有のロジックをカプセル化するために使用します。複数の SQL ステップをストアドプロシージャにまとめることで、アプリリケーションとデータベースを往復する回数を減らすことができます。

細かいアクセスコントロールを行う場合、ストアドプロシージャを作成して、基礎となるテーブルへのアクセスをユーザーに許可することなく関数を実行できます。例えば、所有者またはスーパーユーザーのみがテーブルを切り詰めることができます。また、ユーザーがテーブルにデータを挿入する場合は書き込み権限が必要です。基となるテーブルへの権限をユーザーに付与することなく、タスクを実行するストアドプロシージャを作成できます。次に、このストアドプロシージャを実行するための権限をユーザーに付与します。

ストアドプロシージャは、DEFINER セキュリティ属性が設定されている場合、ストアドプロシージャの所有者の権限で実行されます。デフォルトでは、ストアドプロシージャには INVOKER セキュリティが設定されます。この場合、プロシージャはプロシージャを呼び出したユーザーの権限を使用します。

ストアドプロシージャを作成するには、CREATE PROCEDURE コマンドを使用します。プロシージャを実行するには、CALL コマンドを使用します。このセクションで後ほど例を紹介します。

Note

一部のクライアントは、Amazon Redshift のストアドプロシージャの作成時に次のエラーを表示する場合があります。

```
ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
```

このエラーは、セミコロン区切りのステートメントとドル記号 ($) 引用符を使用して CREATE PROCEDURE ステートメントをクライアントが正しく解析できないために発生します。これにより、ステートメントの一部のみ Amazon Redshift サーバーに送信されます。通常、このエラーは、クライアントの `Run as batch` オプションまたは `Execute selected` オプションを使用することで回避できます。
たとえば、Aginity クライアントを使用する場合は、`Run entire script as batch` オプションを使用します。SQL Workbench/J を使用する場合は、バージョン 124 をお勧めします。SQL Workbench/J バージョン 125 を使用する場合は、回避策として代替の区切り文字を指定することを検討してください。
CREATE PROCEDURE(手順の作成)には、セミコロン (;) で区切られた SQL ステートメントが含まれています。CREATE PROCEDURE にはセミコロン (;) で区切られた SQL ステートメントが含まれているため、スラッシュ (/) など、代替の区切り文字を定義し、CREATE

PROCEDURE ステートメントの最後に配置すると、ステートメントが Amazon Redshift サーバーに送信され、処理されます。次に例を示します。

```
CREATE OR REPLACE PROCEDURE test()
AS $$
BEGIN
  SELECT 1 a;
END;
$$
LANGUAGE plpgsql
;
/
```

詳細については、SQL Workbench/J ドキュメントの「別の区切り文字」を参照してください。または、Amazon Redshift コンソールのクエリエディタや TablePlus など、CREATE PROCEDURE ステートメントの解析をより良くサポートするクライアントを使用します。

トピック

- ストアドプロシージャの名前付け
- ストアドプロシージャのセキュリティおよび権限
- ストアドプロシージャから結果セットを返す
- トランザクションの管理
- エラーのトラップ
- ストアドプロシージャのログ記録
- ストアドプロシージャの制限事項

次の例は、出力引数を使用しないプロシージャを示しています。デフォルトでは、引数は入力 (IN) 引数です。

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar)
AS $$
BEGIN
  RAISE INFO 'f1 = %, f2 = %', f1, f2;
END;
$$ LANGUAGE plpgsql;
```

```
call test_sp1(5, 'abc');
INFO: f1 = 5, f2 = abc
CALL
```

> **Note**
>
> ストアドプロシージャを記述する場合は、機密の値を保護するためのベストプラクティスに従うことをお勧めします。
>
> ストアドプロシージャロジックに機密情報をハードコーディングしないでください。例えば、ストアドプロシージャの本文の CREATE USER ステートメントにユーザーパスワードを割り当てないでください。ハードコードした値は、カタログテーブルにスキーマメタデータとして記録される可能性があるため、セキュリティ上のリスクが生じます。代わりに、パスワードなどの機密の値は、パラメータを使用して引数として、ストアドプロシージャに渡します。
>
> ストアドプロシージャの詳細については、「CREATE PROCEDURE」および「Amazon Redshift のストアドプロシージャの作成」を参照してください。カタログテーブルの詳細については、「システムカタログテーブル」を参照してください。

次の例は、出力引数を使用するプロシージャを示しています。引数は、入力 (IN)、入力および出力 (INOUT)、出力 (OUT) です。

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
END;
```

```
$$ LANGUAGE plpgsql;


call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

ストアドプロシージャの名前付け

このトピックでは、ストアドプロシージャ名の詳細について説明します。

プロシージャを定義する際に同じ名前を付けて、異なる入力引数のデータ型 (署名) を使用すると、別のプロシージャが作成されます。その結果、プロシージャ名はオーバーロードされます。詳細については、「プロシージャ名の多重定義」を参照してください。Amazon Redshift では、出力引数に基づくプロシージャのオーバーロードを有効にしません。2 つのプロシージャで名前と入力引数のデータ型を同じにして、出力引数の型を異なるものにすることはできません。

所有者またはスーパーユーザーは、ストアドプロシージャの本文を、同じ署名を使用する新しいものに置き換えることができます。ストアドプロシージャの署名または戻り値の型を変更するには、ストアドプロシージャを削除して再作成します。詳細については、「DROP PROCEDURE」および「CREATE PROCEDURE」を参照してください。

ストアドプロシージャを実装する前に、その命名規則を考慮することで、競合の可能性や予期しない結果を回避できます。プロシージャ名は多重定義できるため、Amazon Redshift の既存および将来のプロシージャ名と競合する可能性があります。

プロシージャ名の多重定義

プロシージャは、その名前と署名 (入力引数の数および引数のデータ型) で識別されます。名前が同じでも署名が異なる 2 つのプロシージャは、同じスキーマ内に存在できます。つまり、プロシージャ名は多重定義できます。

プロシージャを実行する際に、クエリエンジンは、指定された引数の数と引数のデータ型に基づいて、呼び出すプロシージャを決定します。多重定義を使用すると、CREATE PROCEDURE コマンドで許可されている制限まで、可変数の引数を使用してプロシージャをシミュレートできます。詳細については、「CREATE PROCEDURE」を参照してください。

名前の競合の回避

プレフィックス sp_ を使用してすべてのプロシージャに名前を付けることをお勧めします。Amazon Redshift は、ストアドプロシージャ専用に sp_ プレフィックスを予約します。プロシージャ名に sp_ プレフィックスを付けることで、プロシージャ名は Amazon Redshift の既存または将来のプロシージャ名と競合しなくなります。

ストアドプロシージャのセキュリティおよび権限

このトピックでは、ストアドプロシージャの作成と実行に必要なデータベース認証情報について説明します。

デフォルトでは、すべてのユーザーにプロシージャを作成する権限があります。プロシージャを作成するには、言語 PL/pgSQL に対する USAGE 権限が必要です。この権限はデフォルトで PUBLIC に付与されます。デフォルトでは、スーパーユーザーと所有者のみに、プロシージャを呼び出す権限があります。スーパーユーザーは、特定のユーザーに対してストアドプロシージャの作成を禁止する場合、このユーザーに対して PL/pgSQL の REVOKE USAGE を実行できます。

プロシージャを呼び出すには、プロシージャに対する EXECUTE 権限が付与されている必要があります。デフォルトでは、新しいプロシージャに対する EXECUTE 権限はプロシージャの所有者とスーパーユーザーに付与されます。詳細については、「GRANT」を参照してください。

プロシージャを作成したユーザーがデフォルトで所有者になります。所有者は、デフォルトで、プロシージャに対する CREATE、DROP、EXECUTE の権限を持ちます。スーパーユーザーはすべての権限を持ちます。

SECURITY 属性は、データベースオブジェクトにアクセスするプロシージャの権限を制御します。ストアドプロシージャの作成時に、SECURITY 属性を DEFINER または INVOKER のいずれかに設定できます。SECURITY INVOKER を指定すると、プロシージャはプロシージャを呼び出したユーザーの権限を使用します。SECURITY DEFINER を指定すると、プロシージャはプロシージャの所有者の権限を使用します。デフォルトは INVOKER です。

SECURITY DEFINER プロシージャは、それを所有するユーザーの権限で実行されるため、プロシージャを誤用しないよう注意してください。SECURITY DEFINER プロシージャの誤用を確実に避けるには、以下を行います。

- SECURITY DEFINER プロシージャに対する EXECUTE を特定のユーザーに付与し、PUBLIC には付与しません。
- プロシージャがアクセスする必要があるすべてのデータベースオブジェクトをスキーマ名で修飾します。たとえば、単に `myschema.mytable` としないで、`mytable` を使用します。

- オブジェクト名をスキーマで修飾できない場合は、プロシージャの作成時に SET オプションを使用して search_path を設定します。信頼されていないユーザーが書き込むことができるすべてのスキーマを除外するように search_path を設定します。このアプローチにより、プロシージャで使用する予定のオブジェクトをマスクするオブジェクト (テーブルやビューなど) を、このプロシージャの呼び出し元が作成できないようにします。SET オプションの詳細については、「CREATE PROCEDURE」を参照してください。

次の例では、search_path を admin に設定することで、user_creds テーブルに対して admin スキーマからはアクセスできるが、パブリックや呼び出し元の search_path の他のスキーマからはアクセスできないようにします。

```
CREATE OR REPLACE PROCEDURE sp_get_credentials(userid int, o_creds OUT varchar)
AS $$
BEGIN
  SELECT creds INTO o_creds
  FROM user_creds
  WHERE user_id = $1;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER
-- Set a secure search_path
SET search_path = admin;
```

ストアドプロシージャから結果セットを返す

このトピックでは、ストアドプロシージャがデータを返す方法について説明します。

カーソルや一時テーブルを使用して結果セットを返すことができます。

カーソルを返す

カーソルを返すには、refcursor データ型で定義した INOUT 引数を使用してプロシージャを作成します。このプロシージャを呼び出す場合は、カーソルに名前を付けます。そして、カーソルから名前を指定して結果を取得することができます。

次の例では、get_result_set という名前のプロシージャを作成し、rs_out をデータ型として refcursor という名前の INOUT 引数を使用します。プロシージャは SELECT ステートメントを使用してカーソルを開きます。

```
CREATE OR REPLACE PROCEDURE get_result_set (param IN integer, rs_out INOUT refcursor)
```

```
AS $$
BEGIN
  OPEN rs_out FOR SELECT * FROM fact_tbl where id >= param;
END;
$$ LANGUAGE plpgsql;
```

次の CALL コマンドは mycursor という名前のカーソルを開きます。カーソルはトランザクション内でのみ使用します。

```
BEGIN;
CALL get_result_set(1, 'mycursor');
```

カーソルが開いたら、次の例に示すように、カーソルから結果を取得できます。

```
FETCH ALL FROM mycursor;

    id | secondary_id | name
-------+--------------+---------
     1 |            1 | Joe
     1 |            2 | Ed
     2 |            1 | Mary
     1 |            3 | Mike
(4 rows)
```

最後に、トランザクションはコミットまたはロールバックされます。

```
COMMIT;
```

ストアドプロシージャから返されるカーソルには、DECLARE CURSOR で説明している同じ制約とパフォーマンスの考慮事項が適用されます。。詳細については、「カーソルの制約」を参照してください。

次の例では、JDBC から get_result_set データ型を使用して refcursor ストアドプロシージャを呼び出しています。リテラル 'mycursor' (カーソル名) は prepareStatement に渡されます。次に、結果が ResultSet から取得されます。

```
static void refcursor_example(Connection conn) throws SQLException {
    conn.setAutoCommit(false);
    PreparedStatement proc = conn.prepareStatement("CALL get_result_set(1,
 'mycursor')");
    proc.execute();
```

```
    ResultSet rs = statement.executeQuery("fetch all from mycursor");
    while (rs.next()) {
      int n = rs.getInt(1);
      System.out.println("n " + n);
    }
```

一時テーブルの使用

結果を返すには、結果行を含む一時テーブルへのハンドルを返すことができます。クライアントは、パラメータとして名前をストアドプロシージャに渡します。ストアドプロシージャ内では、動的 SQL を使用して一時テーブルを操作できます。例を以下に示します。

```
CREATE PROCEDURE get_result_set(param IN integer, tmp_name INOUT varchar(256)) as $$
DECLARE
  row record;
BEGIN
  EXECUTE 'drop table if exists ' || tmp_name;
  EXECUTE 'create temp table ' || tmp_name || ' as select * from fact_tbl where id <= '
 || param;
END;
$$ LANGUAGE plpgsql;

CALL get_result_set(2, 'myresult');
 tmp_name
-----------
 myresult
(1 row)

SELECT * from myresult;
 id | secondary_id | name
----+--------------+------
  1 |            1 | Joe
  2 |            1 | Mary
  1 |            2 | Ed
  1 |            3 | Mike
(4 rows)
```

トランザクションの管理

ストアドプロシージャは、デフォルトのトランザクション管理動作または非アトミック動作で作成できます。

デフォルトモードのストアドプロシージャのトランザクション管理

デフォルトのトランザクションモードの自動コミット動作では、個別に実行された SQL コマンドが個別にコミットされます。ストアドプロシージャへの呼び出しは、単一の SQL コマンドとして扱われます。プロシージャ内の SQL ステートメントは、呼び出しの開始時および終了時に暗黙的に開始および終了するトランザクションブロックに存在するかのように動作します。別のプロシージャに対するネストされた呼び出しは、他の任意の SQL ステートメントのように扱われ、呼び出し元と同じトランザクションのコンテキスト内で動作します。自動コミット動作の詳細については、「直列化可能分離」を参照してください。

ただし、ユーザー指定のトランザクションブロック (BEGIN... COMMIT で定義される) 内からストアドプロシージャを呼び出すとします。この場合、ストアドプロシージャのすべてのステートメントはユーザー指定のトランザクションのコンテキストで実行されます。プロシージャは終了時に暗黙でコミットしません。呼び出し元がプロシージャのコミットまたはロールバックを制御します。

ストアドプロシージャの実行中にエラーが発生した場合は、現在のトランザクションで行われたすべての変更がロールバックされます。

ストアドプロシージャでは、次のトランザクション制御ステートメントを使用できます。

- COMMIT – 現在のトランザクションで行われたすべての作業をコミットし、暗黙的に新しいトランザクションを開始します。詳細については、「COMMIT」を参照してください。
- ROLLBACK – 現在のトランザクションで行われた作業をロールバックし、暗黙的に新しいトランザクションを開始します。詳細については、「ROLLBACK」を参照してください。

TRUNCATE は、ストアドプロシージャ内から発行できる別のステートメントであり、トランザクション管理に影響します。Amazon Redshift では、TRUNCATE は暗黙でコミットを発行します。この動作は、ストアドプロシージャのコンテキストでも変わりません。TRUNCATE ステートメントは、ストアドプロシージャ内から発行されると、現在のトランザクションをコミットして新しいものを開始します。詳細については、「TRUNCATE」を参照してください。

COMMIT、ROLLBACK、または TRUNCATE ステートメントに続くステートメントはすべて、新しいトランザクションのコンテキストで実行されます。これらは、COMMIT、ROLLBACK、または TRUNCATE ステートメントが検出されるか、ストアドプロシージャが終了するまで行われます。

ストアドプロシージャ内から COMMIT、ROLLBACK、または TRUNCATE ステートメントを使用する場合は、次の制約が適用されます。

- ストアドプロシージャがトランザクションブロック内から呼び出された場合、COMMIT、ROLLBACK、または TRUNCATE ステートメントを発行することはできません。この制限は、ストアドプロシージャ自体の本体内およびネストされたプロシージャ呼び出し内に適用されます。
- ストアドプロシージャが `SET config` オプション付きで作成されている場合は、COMMIT、ROLLBACK、または TRUNCATE ステートメントを発行することはできません。この制限は、ストアドプロシージャ自体の本体内およびネストされたプロシージャ呼び出し内に適用されます。
- すべての (明示的または暗黙的に) 開いているカーソルは、COMMIT、ROLLBACK、または TRUNCATE ステートメントの処理時に自動的に閉じられます。明示的および暗黙的なカーソルの制約については、「ストアドプロシージャの制限事項」を参照してください。

また、動的 SQL を使用して COMMIT または ROLLBACK を実行することはできません。ただし、動的 SQL を使用して TRUNCATE を実行することができます。詳細については、「動的 SQL」を参照してください。

ストアドプロシージャを使用する場合、PL/pgSQL の BEGIN および END ステートメントはグループ化のみを目的としている点を考慮してください。トランザクションを開始または終了することはありません。詳細については、「ブロック」を参照してください。

次の例は、ストアドプロシージャを明示的なトランザクションブロック内から呼び出した場合のトランザクション動作を示しています。ストアドプロシージャの外部から発行された 2 つの挿入ステートメントと内部から発行された 1 つの挿入ステートメントは、すべて同じトランザクション (3382) に属します。トランザクションは、ユーザーが明示的なコミットを発行したときにコミットされます。

```
CREATE OR REPLACE PROCEDURE sp_insert_table_a(a int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
END;
$$;

Begin;
  insert into test_table_a values (1);
  Call sp_insert_table_a(2);
  insert into test_table_a values (3);
Commit;
```

```
select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |                stmt_text
--------+------+-----+---------+----------------------------------------
    103 | 3382 | 599 | UTILITY | Begin;
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3382 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3382 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3382 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3382 | 599 | UTILITY | COMMIT
```

それに対して、明示的なトランザクションブロックの外部から同じステートメントが発行され、セッションの自動コミットがオンに設定されている場合の例を示します。この場合、各ステートメントは独自のトランザクションで実行されます。

```
insert into test_table_a values (1);
Call sp_insert_table_a(2);
insert into test_table_a values (3);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  | pid |  type   |
             stmt_text
--------+------+-----+---------
+-----------------------------------------------------------------------------------
    103 | 3388 | 599 | QUERY   | insert into test_table_a values (1);
    103 | 3388 | 599 | UTILITY | COMMIT
    103 | 3389 | 599 | UTILITY | Call sp_insert_table_a(2);
    103 | 3389 | 599 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3389 | 599 | UTILITY | COMMIT
    103 | 3390 | 599 | QUERY   | insert into test_table_a values (3);
    103 | 3390 | 599 | UTILITY | COMMIT
```

次の例では、TRUNCATE ステートメントを `test_table_a` 内に挿入後に発行しています。。TRUNCATE ステートメントは、現在のトランザクション (3335) をコミットして新しいトランザクション (3336) を開始する暗黙的なコミットを発行します。新しいトランザクションは、プロシージャの終了時にコミットされます。

```
CREATE OR REPLACE PROCEDURE sp_truncate_proc(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table_a values (a);
  TRUNCATE test_table_b;
  INSERT INTO test_table_b values (b);
END;
$$;

Call sp_truncate_proc(1,2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |  pid  |  type   |
                                        stmt_text
--------+------+-------+---------
+----------------------------------------------------------------------------------------------------
    103 | 3335 | 23636 | UTILITY | Call sp_truncate_proc(1,2);
    103 | 3335 | 23636 | QUERY   | INSERT INTO test_table_a values ( $1 )
    103 | 3335 | 23636 | UTILITY | TRUNCATE test_table_b
    103 | 3335 | 23636 | UTILITY | COMMIT
    103 | 3336 | 23636 | QUERY   | INSERT INTO test_table_b values ( $1 )
    103 | 3336 | 23636 | UTILITY | COMMIT
```

次の例では、ネストされた呼び出しから TRUNCATE を発行しています。TRUNCATE は、トランザクション (3344) の外部および内部のプロシージャでそれまでに処理されたすべての作業をコミットします。また、新しいトランザクション (3345) を開始します。新しいトランザクションは、外部のプロシージャの終了時にコミットされます。

```
CREATE OR REPLACE PROCEDURE sp_inner(c int, d int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO inner_table values (c);
  TRUNCATE outer_table;
  INSERT INTO inner_table values (d);
END;
$$;

CREATE OR REPLACE PROCEDURE sp_outer(a int, b int, c int, d int) LANGUAGE plpgsql
AS $$
```

```
BEGIN
  INSERT INTO outer_table values (a);
  Call sp_inner(c, d);
  INSERT INTO outer_table values (b);
END;
$$;

Call sp_outer(1, 2, 3, 4);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |  pid  |  type   |
                                          stmt_text
--------+------+-------+---------
+-----------------------------------------------------------------------------------------------
    103 | 3344 | 23636 | UTILITY | Call sp_outer(1, 2, 3, 4);
    103 | 3344 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | CALL sp_inner( $1 , $2 )
    103 | 3344 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3344 | 23636 | UTILITY | TRUNCATE outer_table
    103 | 3344 | 23636 | UTILITY | COMMIT
    103 | 3345 | 23636 | QUERY   | INSERT INTO inner_table values ( $1 )
    103 | 3345 | 23636 | QUERY   | INSERT INTO outer_table values ( $1 )
    103 | 3345 | 23636 | UTILITY | COMMIT
```

次の例は、TRUNCATE ステートメントのコミット時にカーソル `cur1` が閉じられたことを示しています。

```
CREATE OR REPLACE PROCEDURE sp_open_cursor_truncate()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  open cur1;
  TRUNCATE table test_table_b;
  Loop
    fetch cur1 into rec;
    raise info '%', rec.c1;
    exit when not found;
```

```
  End Loop;
END
$$;

call sp_open_cursor_truncate();
ERROR: cursor "cur1" does not exist
CONTEXT: PL/pgSQL function "sp_open_cursor_truncate" line 8 at fetch
```

次の例では、TRUNCATE ステートメントを発行します。明示的なトランザクションブロック内から呼び出すことはできません。

```
CREATE OR REPLACE PROCEDURE sp_truncate_atomic() LANGUAGE plpgsql
AS $$
BEGIN
  TRUNCATE test_table_b;
END;
$$;

Begin;
  Call sp_truncate_atomic();
ERROR: TRUNCATE cannot be invoked from a procedure that is executing in an atomic
 context.
HINT: Try calling the procedure as a top-level call i.e. not from within an explicit
 transaction block.
Or, if this procedure (or one of its ancestors in the call chain) was created with SET
 config options, recreate the procedure without them.
CONTEXT: SQL statement "TRUNCATE test_table_b"
PL/pgSQL function "sp_truncate_atomic" line 2 at SQL statement
```

次の例は、スーパーユーザーまたはテーブルの所有者ではないユーザーが、テーブルに対して TRUNCATE ステートメントを発行できることを示しています。ユーザーは、`Security Definer`ストアドプロシージャを使用して、これを行います。この例では以下のアクションを示します。

- ユーザー 1 はテーブル `test_tbl` を作成します。
- ユーザー 1 はストアドプロシージャ `sp_truncate_test_tbl` を作成します。
- ユーザー 1 は、ストアドプロシージャに対する `EXECUTE` 権限をユーザー 2 に付与します。
- ユーザー 2 は、ストアドプロシージャを実行してテーブル `test_tbl` を切り捨てます。この例は、`TRUNCATE` コマンドの前後の行数を示しています。

```
set session_authorization to user1;
create table test_tbl(id int, name varchar(20));
insert into test_tbl values (1,'john'), (2, 'mary');
CREATE OR REPLACE PROCEDURE sp_truncate_test_tbl() LANGUAGE plpgsql
AS $$
DECLARE
  tbl_rows int;
BEGIN
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount before Truncate: %', tbl_rows;
  TRUNCATE test_tbl;
  select count(*) into tbl_rows from test_tbl;
  RAISE INFO 'RowCount after Truncate: %', tbl_rows;
END;
$$ SECURITY DEFINER;
grant execute on procedure sp_truncate_test_tbl() to user2;
reset session_authorization;


set session_authorization to user2;
call sp_truncate_test_tbl();
INFO:  RowCount before Truncate: 2
INFO:  RowCount after Truncate: 0
CALL
reset session_authorization;
```

次の例では、COMMIT を 2 回発行しています。最初の COMMIT では、トランザクション 10363 で行われたすべての作業をコミットし、トランザクション 10364 を暗黙的に開始します。トランザクション 10364 は、2 番目の COMMIT ステートメントによってコミットされます。

```
CREATE OR REPLACE PROCEDURE sp_commit(a int, b int) LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO test_table values (a);
  COMMIT;
  INSERT INTO test_table values (b);
  COMMIT;
END;
$$;

call sp_commit(1,2);
```

```
select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;
 userid |  xid  | pid  |  type   |
                               stmt_text
--------+-------+------+---------
+----------------------------------------------------------------------------------------
    100 | 10363 | 3089 | UTILITY | call sp_commit(1,2);
    100 | 10363 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10363 | 3089 | UTILITY | COMMIT
    100 | 10364 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10364 | 3089 | UTILITY | COMMIT
```

次の例では、sum_vals が 2 より大きい場合に ROLLBACK ステートメントを発行します。最初の ROLLBACK ステートメントでは、トランザクション 10377 で行われたすべての作業をロールバックし、新しいトランザクション 10378 を開始します。プロシージャが終了すると、トランザクション 10378 がコミットされます。

```
CREATE OR REPLACE PROCEDURE sp_rollback(a int, b int) LANGUAGE plpgsql
AS $$
DECLARE
  sum_vals int;
BEGIN
  INSERT INTO test_table values (a);
  SELECT sum(c1) into sum_vals from test_table;
  IF sum_vals > 2 THEN
    ROLLBACK;
  END IF;

  INSERT INTO test_table values (b);
END;
$$;

call sp_rollback(1, 2);

select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

userid |  xid  | pid  |  type   |
                               stmt_text
--------+-------+------+---------
+----------------------------------------------------------------------------------------
```

```
    100 | 10377 | 3089 | UTILITY | call sp_rollback(1, 2);
    100 | 10377 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10377 | 3089 | QUERY   | SELECT sum(c1) from test_table
    100 | 10377 | 3089 | QUERY   | Undoing 1 transactions on table 133646 with current
 xid 10377 : 10377
    100 | 10378 | 3089 | QUERY   | INSERT INTO test_table values ( $1 )
    100 | 10378 | 3089 | UTILITY | COMMIT
```

非アトミックモードのストアドプロシージャのトランザクション管理

NONATOMIC モードで作成されたストアドプロシージャは、デフォルトモードで作成されたプロシージャとはトランザクション制御動作が異なります。ストアドプロシージャ外の SQL コマンドの自動コミット動作と同様に、NONATOMIC プロシージャの内側の各 SQL ステートメントは、独自のトランザクションで実行され、自動的にコミットします。ユーザーが NONATOMIC ストアドプロシージャ内で明示的なトランザクションブロックを開始した場合、ブロック内の SQL ステートメントは自動的にコミットされません。トランザクションブロックは、その中のステートメントのコミットまたはロールバックを制御します。

NONATOMIC ストアドプロシージャでは、START TRANSACTION ステートメントを使用してプロシージャ内の明示的なトランザクションブロックを開くことができます。ただし、既に開いているトランザクションブロックがある場合、Amazon Redshift はサブトランザクションをサポートしていないため、このステートメントは何もしません。前のトランザクションは続行されます。

NONATOMIC プロシージャ内でカーソルの FOR ループを操作する場合は、クエリの結果を繰り返し処理する前に、必ず明示的なトランザクションブロックを開いてください。それ以外の場合は、ループ内の SQL ステートメントが自動的にコミットされたときにカーソルが閉じられます。

NONATOMIC モードの動作を使用する際の考慮事項は次のとおりです。

- ストアドプロシージャ内の各 SQL ステートメントは、開いているトランザクションブロックがなく、セッションの自動コミットがオンに設定されている場合、自動的にコミットされます。
- ストアドプロシージャがトランザクションブロック内から呼び出された場合、COMMIT、ROLLBACK、TRUNCATE ステートメントを発行してトランザクションを終了できます。これはデフォルトモードでは不可能です。
- START TRANSACTION ステートメントを発行すると、ストアドプロシージャ内のトランザクションブロックを開始できます。

次の例は、NONATOMIC ストアドプロシージャを使用する場合のトランザクション動作を示しています。以下のすべての例のセッションでは、オートコミットが ON に設定されています。

次の例では、NONATOMIC ストアドプロシージャに 2 つの INSERT ステートメントがあります。プロシージャがトランザクションブロックの外部で呼び出されると、プロシージャ内のすべての INSERT ステートメントが自動的にコミットされます。

```
CREATE TABLE test_table_a(v int);
CREATE TABLE test_table_b(v int);

CREATE OR REPLACE PROCEDURE sp_nonatomic_insert_table_a(a int, b int) NONATOMIC AS
$$
BEGIN
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_insert_table_a(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |               stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1792 | 1073807554 | UTILITY | Call sp_nonatomic_insert_table_a(1,2);
      1 | 1792 | 1073807554 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1792 | 1073807554 | UTILITY | COMMIT
      1 | 1793 | 1073807554 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1793 | 1073807554 | UTILITY | COMMIT
(5 rows)
```

ただし、プロシージャが BEGIN..COMMIT ブロック内から呼び出された場合、すべてのステートメントは同じトランザクションの一部です (xid=1799)。

```
Begin;
  INSERT INTO test_table_a values (10);
  Call sp_nonatomic_insert_table_a(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;
```

```
 userid | xid  |    pid     |  type   |                 stmt_text
--------+------+------------+---------+------------------------------------------
      1 | 1799 | 1073914035 | UTILITY | Begin;
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1799 | 1073914035 | UTILITY | Call sp_nonatomic_insert_table_a(20,30);
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1799 | 1073914035 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1799 | 1073914035 | UTILITY | COMMIT
(7 rows)
```

この例では、トランザクションの開始とコミットの間に 2 つの INSERT ステートメントがあります。プロシージャがトランザクションブロックの外部で呼び出されると、2 つの INSERT ステートメントは同じトランザクションに含まれます (xid=1866)。

```
CREATE OR REPLACE PROCEDURE sp_nonatomic_txn_block(a int, b int) NONATOMIC AS
$$
BEGIN
    START TRANSACTION;
    INSERT INTO test_table_a values (a);
    INSERT INTO test_table_b values (b);
    COMMIT;
END;
$$
LANGUAGE plpgsql;

Call sp_nonatomic_txn_block(1,2);

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |                stmt_text
--------+------+------------+---------+-----------------------------------------
      1 | 1865 | 1073823998 | UTILITY | Call sp_nonatomic_txn_block(1,2);
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1866 | 1073823998 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1866 | 1073823998 | UTILITY | COMMIT
(4 rows)
```

プロシージャが BBEGIN...COMMIT ブロック内から呼び出された場合、プロシージャ内の START TRANSACTION は、既に開いているトランザクションがあるため、何も実行しません。このプロシージャ内の COMMIT は、現在のトランザクション (xid=1876) をコミットして、新しいトランザクションをコミットします。

```
Begin;
  INSERT INTO test_table_a values (10);
  Call sp_nonatomic_txn_block(20,30);
  INSERT INTO test_table_b values (40);
Commit;

Select userid, xid, pid, type, trim(text) as stmt_text
from svl_statementtext where pid = pg_backend_pid() order by xid , starttime ,
 sequence;

 userid | xid  |    pid     |  type   |                stmt_text
--------+------+------------+---------+----------------------------------------
      1 | 1876 | 1073832133 | UTILITY | Begin;
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values (10);
      1 | 1876 | 1073832133 | UTILITY | Call sp_nonatomic_txn_block(20,30);
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_a values ( $1 )
      1 | 1876 | 1073832133 | QUERY   | INSERT INTO test_table_b values ( $1 )
      1 | 1876 | 1073832133 | UTILITY | COMMIT
      1 | 1878 | 1073832133 | QUERY   | INSERT INTO test_table_b values (40);
      1 | 1878 | 1073832133 | UTILITY | COMMIT
(8 rows)
```

この例では、カーソルループを操作する方法を示します。テーブル test_table_a には 3 つの値があります。目的は、3 つの値を繰り返し処理して、テーブル test_table_b に挿入することです。NONATOMIC ストアドプロシージャを次の方法で作成すると、最初のループで INSERT ステートメントを実行したあとに、「"cur1" は存在しません」というエラーカーソルが表示されます。これは、INSERT の自動コミットによって開いているカーソルが閉じるためです。

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
```

```
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
ERROR:  cursor "cur1" does not exist
CONTEXT:  PL/pgSQL function "sp_nonatomic_cursor" line 7 at fetch
```

カーソルをループさせるには、トランザクションの開始とコミットの間にカーソルを置いてください。

```
insert into test_table_a values (1), (2), (3);

CREATE OR REPLACE PROCEDURE sp_nonatomic_cursor() NONATOMIC
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  cur1 cursor for select * from test_table_a order by 1;
BEGIN
  START TRANSACTION;
  open cur1;
  Loop
    fetch cur1 into rec;
    exit when not found;
    raise info '%', rec.v;
    insert into test_table_b values (rec.v);
  End Loop;
  COMMIT;
END
$$;

CALL sp_nonatomic_cursor();

INFO:  1
```

```
INFO:  2
INFO:  3
CALL
```

エラーのトラップ

このトピックでは、Amazon Redshift がエラーを処理する方法について説明します。

ストアドプロシージャのクエリまたはコマンドでエラーが発生すると、それ以降のクエリが実行されず、トランザクションがロールバックされます。しかし、 EXCEPTION ブロックを使用してエラーに対処できます。

> **Note**
>
> デフォルトの動作では、ストアドプロシージャにエラーを生成する条件が他にない場合でも、エラーが発生すると後続のクエリは実行されません。

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  WHEN OTHERS THEN
    statements
END;
```

例外が発生し、例外処理ブロックを追加すると、REASE ステートメントおよび他のほとんどの PL/pgSQL ステートメントを書き込みできます。たとえば、カスタムメッセージで例外を発生させたり、ロギングテーブルにレコードを挿入したりできます。

例外処理ブロックに入ると、現在のトランザクションがロールバックされ、新しいトランザクションがブロック内のステートメントを実行するために作成されます。ブロック内のステートメントがエラーなしで実行されると、トランザクションはコミットされ、例外が再スローされます。最後に、ストアドプロシージャが終了します。

例外ブロックでサポートされている条件は OTHERS のみです。この条件は、クエリのキャンセルを除いて、すべてのエラータイプに一致します。また、例外処理ブロックでエラーが発生すると、エラーは外部の例外処理ブロックでキャッチできます。

NONATOMIC プロシージャ内でエラーが発生しても、例外ブロックで処理されればエラーは再スローされません。例外処理ブロックでキャッチされた例外をスローする方法については、「PL/pgSQL ステートメント RAISE」を参照してください。このステートメントは例外処理ブロックでのみ有効です。詳細については、「RAISE」を参照してください。

CONTINUE ハンドラを使用して、ストアドプロシージャでエラーが発生した後の処理を制御する

CONTINUE ハンドラは、NONATOMIC ストアドプロシージャ内の実行フローを制御する例外ハンドラの一種です。これを使用すると、既存のステートメントブロックを終了せずに例外をキャッチして処理できます。通常、ストアドプロシージャでエラーが発生すると、フローが中断され、エラーが呼び出し元に返されます。ただし、ユースケースによっては、フローの中断を必要とするほどエラー状態が深刻ではない場合があります。別のトランザクションで選択したエラー処理ロジックを使用してエラーをスムーズに処理し、エラーの後にステートメントを実行し続けることもできます。以下にその構文を示します。

```
[ DECLARE
  declarations ]
BEGIN
  statements
EXCEPTION
  [ CONTINUE_HANDLER | EXIT_HANDLER ] WHEN OTHERS THEN
    handler_statements
END;
```

さまざまなタイプのエラーに関する情報を収集するのに役立つシステムテーブルがいくつかあります。詳細については、STL_LOAD_ERRORS、STL_ERROR、およびSYS_STREAM_SCAN_ERRORSを参照してください。エラーのトラブルシューティングに使用できるその他のシステムテーブルもあります。これらに関する情報は、「システムテーブルとビューのリファレンス」を参照してください。

例

次の例は、例外処理ブロックでステートメントを書き込みする方法を示しています。ストアドプロシージャはデフォルトのトランザクション管理動作を使用しています。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp() AS
$$
```

```
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp();

INFO:  An exception occurred.
ERROR:  column "invalid" does not exist
CONTEXT:  SQL statement "select invalid"
PL/pgSQL function "update_employee_sp" line 3 at execute statement
```

この例では、update_employee_spを呼び出すと、情報メッセージ例外が発生しました。が生成され、エラーメッセージがロギングテーブルのemployee_error_logログに挿入されます。元の例外は、ストアドプロシージャが終了する前に再度スローされます。次のクエリは、この例を実行した結果のレコードを示しています。

```
SELECT * from employee;

firstname | lastname
-----------+----------
 Tomas     | Smith

SELECT * from employee_error_log;

          message
------------------------------------------------
 Error message: column "invalid" does not exist
```

フォーマットのヘルプや追加レベルのリストなど、RAISE の詳細については、「サポートされている PL/pgSQL ステートメント」を参照してください。

次の例は、例外処理ブロックでステートメントを書き込みする方法を示しています。ストアドプロシージャは NONATOMIC トランザクション管理動作を使用しています。この例では、プロシージャコールが完了しても呼び出し元にエラーは返されません。次のステートメントのエラーのため、UPDATE ステートメントはロールバックされません。情報メッセージが表示され、エラーメッセージがロギングテーブルに挿入されます。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

-- Create the SP in NONATOMIC mode
CREATE OR REPLACE PROCEDURE update_employee_sp_2() NONATOMIC AS
$$
BEGIN
    UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
    EXECUTE 'select invalid';
EXCEPTION WHEN OTHERS THEN
    RAISE INFO 'An exception occurred.';
    INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_2();
INFO:  An exception occurred.
CALL

SELECT * from employee;

 firstname | lastname
-----------+----------
 Adam      | Smith
(1 row)

SELECT * from employee_error_log;

                    message
------------------------------------------------
 Error message: column "invalid" does not exist
(1 row)
```

この例では、2 つのサブブロックを持つプロシージャを作成する方法を示します。ストアドプロシージャが呼び出されると、最初のサブブロックからのエラーはその例外処理ブロックによって処理されます。最初のサブブロックが完了すると、プロシージャは引き続き 2 番目のサブブロックを実行します。結果から、プロシージャコールが完了してもエラーは発生しないことがわかります。テーブル従業員の UPDATE オペレーションと INSERT オペレーションがコミットされます。両方の例外ブロックからのエラーメッセージがログテーブルに挿入されます。

```
CREATE TABLE employee (firstname varchar, lastname varchar);
INSERT INTO employee VALUES ('Tomas','Smith');
CREATE TABLE employee_error_log (message varchar);

CREATE OR REPLACE PROCEDURE update_employee_sp_3() NONATOMIC AS
$$
BEGIN
    BEGIN
        UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
        EXECUTE 'select invalid1';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the first block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
    BEGIN
        INSERT INTO employee VALUES ('Edie','Robertson');
        EXECUTE 'select invalid2';
    EXCEPTION WHEN OTHERS THEN
        RAISE INFO 'An exception occurred in the second block.';
        INSERT INTO employee_error_log VALUES ('Error message: ' || SQLERRM);
    END;
END;
$$
LANGUAGE plpgsql;

CALL update_employee_sp_3();
INFO:  An exception occurred in the first block.
INFO:  An exception occurred in the second block.
CALL

SELECT * from employee;

 firstname | lastname
-----------+-----------
 Adam      | Smith
 Edie      | Robertson
(2 rows)

SELECT * from employee_error_log;

                     message
-------------------------------------------------
 Error message: column "invalid1" does not exist
```

```
 Error message: column "invalid2" does not exist
(2 rows)
```

次の例は、CONTINUE 例外ハンドラの使用方法を示しています。このサンプルでは 2 つのテーブルを作成し、ストアドプロシージャで使用します。CONTINUE ハンドラは、NONATOMIC トランザクション管理動作でストアドプロシージャの実行フローを制御します。

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_1() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
    INSERT INTO tbl_1 VALUES (2);
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

ストアドプロシージャを呼び出します。

```
CALL sp_exc_handling_1();
```

フローは以下のように進行します。

1. 互換性のないデータ型を列に挿入しようとして、エラーが発生します。EXCEPTION ブロックに制御が移ります。例外処理ブロックが入力されると、現在のトランザクションがロールバックされ、新しい暗黙的トランザクションが作成されてステートメントを実行します。
2. CONTINUE_HANDLER のステートメントがエラーなく実行された場合、例外が発生したステートメントの直後のステートメントに制御が移ります。(CONTINUE_HANDLER のステートメントによって新しい例外が発生した場合は、EXCEPTION ブロック内の例外ハンドラを使用してその例外を処理できます。)

サンプルストアドプロシージャを呼び出すと、テーブルに次のレコードが格納されます。

- `SELECT * FROM tbl_1;` を実行すると、2 つのレコードが返されます。これらには値 `1` と `2` が含まれます。

• `SELECT * FROM tbl_error_logging;` を実行すると、次の値を含むレコードが 1 つ返されます。「エラーが発生しました」、「42703」、「val 列は tbl_1 に存在しません」

以下の追加のエラー処理例では、EXIT ハンドラと CONTINUE ハンドラの両方を使用しています。データテーブルとロギングテーブルの 2 つのテーブルを作成します。また、エラー処理を示すストアドプロシージャも作成します。

```
CREATE TABLE tbl_1 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_2() NONATOMIC AS
$$
BEGIN
    INSERT INTO tbl_1 VALUES (1);
    BEGIN
        INSERT INTO tbl_1 VALUES (100);
        -- Expect an error for the insert statement following, because of the invalid
 value
        INSERT INTO tbl_1 VALUES ("val");
        INSERT INTO tbl_1 VALUES (101);
    EXCEPTION EXIT_HANDLER WHEN OTHERS THEN
        INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
    END;
    INSERT INTO tbl_1 VALUES (2);
    -- Expect an error for the insert statement following, because of the invalid value
    INSERT INTO tbl_1 VALUES ("val");
    INSERT INTO tbl_1 VALUES (3);
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

ストアドプロシージャを作成したら、次のように呼び出します。

```
CALL sp_exc_handling_2();
```

BEGIN と END の内部セットで囲まれた内部例外ブロックでエラーが発生すると、EXIT ハンドラによって処理されます。外側のブロックで発生したエラーは、CONTINUE ハンドラによって処理されます。

サンプルストアドプロシージャを呼び出すと、テーブルに次のレコードが格納されます。

- SELECT * FROM tbl_1; を実行すると、値 1、2、3、および 100 を持つ 4 つのレコードが返されます。
- SELECT * FROM tbl_error_logging; を実行すると、2 つのレコードが返されます。これらの値は次のとおりです。「エラーが発生しました」、「42703」、「val 列は tbl_1 に存在しません」

tbl_error_logging テーブルが存在しない場合、例外が発生します。

次の例は、CONTINUE 例外ハンドラを FOR ループで使用する方法を示しています。このサンプルでは 3 つのテーブルを作成し、ストアドプロシージャ内の FOR ループで使用します。FOR ループは結果セットバリアントであり、クエリの結果を反復処理します。

```
CREATE TABLE tbl_1 (a int);
INSERT INTO tbl_1 VALUES (1), (2), (3);
CREATE TABLE tbl_2 (a int);
CREATE TABLE tbl_error_logging(info varchar, err_state varchar, err_msg varchar);

CREATE OR REPLACE PROCEDURE sp_exc_handling_loop() NONATOMIC AS
$$
DECLARE
 rec RECORD;
BEGIN
    FOR rec IN SELECT a FROM tbl_1
    LOOP
        IF rec.a = 2 THEN
            -- Expect an error for the insert statement following, because of the
 invalid value
            INSERT INTO tbl_2 VALUES("val");
        ELSE
            INSERT INTO tbl_2 VALUES (rec.a);
        END IF;
    END LOOP;
EXCEPTION CONTINUE_HANDLER WHEN OTHERS THEN
    INSERT INTO tbl_error_logging VALUES ('Encountered error', SQLSTATE, SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

ストアドプロシージャを呼び出します。

```
CALL sp_exc_handling_loop();
```

サンプルストアドプロシージャを呼び出すと、テーブルに次のレコードが格納されます。

- `SELECT * FROM tbl_2;` を実行すると、2 つのレコードが返されます。これらには値 1 と 3 が含まれます。
- `SELECT * FROM tbl_error_logging;` を実行すると、次の値を含むレコードが 1 つ返されます。「エラーが発生しました」「42703」、「val 列は tbl_2 に存在しません」

CONTINUE ハンドラに関する使用上の注意事項:

- CONTINUE_HANDLER キーワードと EXIT_HANDLER キーワードは、NONATOMIC ストアドプロシージャでのみ使用できます。
- CONTINUE_HANDLER キーワードと EXIT_HANDLER キーワードはオプションです。EXIT_HANDLER がデフォルトです。

ストアドプロシージャのログ記録

このトピックでは、Amazon Redshift がストアドプロシージャのログ記録に使用するストアドプロシージャとビューについて説明します。

ストアドプロシージャに関する詳細は、以下のシステムテーブルやビューに記録されます。

- SVL_STORED_PROC_CALL – ストアドプロシージャ呼び出しの開始時刻や終了時刻、呼び出しが完了前に終了したかどうかなどの詳細が記録されます。詳細については、「SVL_STORED_PROC_CALL」を参照してください。
- SVL_STORED_PROC_MESSAGES – RAISE クエリによって生成されるストアドプロシージャ内のメッセージが、対応するログ記録レベルで記録されます。詳細については、「SVL_STORED_PROC_MESSAGES」を参照してください。
- SVL_QLOG – ストアドプロシージャから呼び出されたクエリごとに、プロシージャ呼び出しのクエリ ID が記録されます。詳細については、「SVL_QLOG」を参照してください。
- STL_UTILITYTEXT – ストアドプロシージャ呼び出しが、完了後に記録されます。詳細については、「STL_UTILITYTEXT」を参照してください。
- PG_PROC_INFO – このシステムカタログビューは、ストアドプロシージャに関する情報を示します。詳細については、「PG_PROC_INFO」を参照してください。

ストアドプロシージャの制限事項

このトピックでは、Amazon Redshift ストアドプロシージャの制限事項について説明します。

Amazon Redshift のストアドプロシージャを使用する場合は、以下の考慮事項が適用されます。

Amazon Redshift および PostgreSQL のストアドプロシージャサポートに関する違い

Amazon Redshift および PostgreSQL 間のストアドプロシージャサポートの違いは以下のとおりです。

- Amazon Redshift はサブトランザクションをサポートしていないため、例外処理ブロックに対するサポートは制限されます。

考慮事項と制限

Amazon Redshift のストアドプロシージャに対する考慮事項は以下のとおりです。

- データベースのストアドプロシージャの最大数は 10,000 です。
- プロシージャのソースコードの最大サイズは 2 MB です。
- ユーザーセッションで同時に開くことができる明示的および暗黙的なカーソルの最大数は 1 です。SQL ステートメントの結果セットを反復処理する FOR ループは、暗黙的なカーソルを開きます。ネストされたカーソルはサポートされていません。
- 明示的および暗黙的なカーソルには、結果セットのサイズについて Amazon Redshift の標準カーソルと同じ制限が適用されます。詳細については、「カーソルの制約」を参照してください。
- ネストされた呼び出しの最大レベル数は 16 です。
- プロシージャパラメータの最大数は、入力引数の場合は 32、出力引数の場合は 32 です。
- ストアドプロシージャの変数の最大数は 1,024 です。
- 独自のトランザクションコンテキストを必要とするすべての SQL コマンドは、ストアドプロシージャ内でサポートされていません。その例を以下に示します。
  - PREPARE
  - データベースの作成/削除
  - CREATE EXTERNAL TABLE
  - VACUUM
  - ローカルに設定
  - ALTER TABLE APPEND

- Java Database Connectivity (JDBC) ドライバー経由の `registerOutParameter` メソッド呼び出しは、`refcursor` データ型ではサポートされていません。データ型の使用例については、「`refcursor`」を参照してください。。ストアドプロシージャから結果セットを返す

PL/pgSQL 言語リファレンス

Amazon Redshift のストアドプロシージャは PostgreSQL PL/pgSQL プロシージャ言語に基づいています。ただし、いくつかの重要な違いがあります。このリファレンスでは、Amazon Redshift で実装されている PL/pgSQL 構文の詳細を確認できます。PL/pgSQL の詳細については、PostgreSQL ドキュメントの PL/pgSQL - SQL 手続き言語を参照してください。

トピック

- PL/pgSQL リファレンスの規則
- PL/pgSQL の構造
- サポートされている PL/pgSQL ステートメント

PL/pgSQL リファレンスの規則

このセクションでは、PL/pgSQL ストアドプロシージャ言語の構文を記述するために使用する規則を確認できます。

文字	説明
CAPS	大文字の単語はキーワードです。
[ ]	角括弧はオプションの引数を示します。角括弧に複数の引数が含まれる場合は、任意の個数の引数を選択できることを示します。さらに、複数の行に角括弧で囲まれた引数がある場合、Amazon Redshift パーサーは、それらの引数が構文の順番どおりに出現するものと想定します。
{ }	中括弧は、括弧内の引数の 1 つを選択する必要があることを示します。
|	縦線は、どちらかの引数を選択できることを示します。
red italics	赤色のイタリック体の単語は、プレースホルダを示します。赤色のイタリック体の単語に代えて適切な値を挿入します。

文字	説明
...	省略符号は、先行する要素の繰り返しが可能であることを示します。
'	一重引用符に囲まれた単語は、引用符の入力が必要であることを示します。

PL/pgSQL の構造

PL/pgSQL は手続き言語であり、他の手続き言語と同じ構造体が多数含まれています。

トピック

- ブロック
- 変数宣言
- エイリアス宣言
- 組み込み変数
- レコード型

ブロック

PL/pgSQL はブロック構造言語です。プロシージャの完全な本文は、ブロック内で定義されます。ブロックには、変数宣言と PL/pgSQL ステートメントが含まれます。ステートメントは、ネストされたブロックまたはサブブロックである場合もあります。

宣言とステートメントはセミコロンで終了します。ブロックまたはサブブロックの END キーワードの後にセミコロンを続けます。DECLARE キーワードと BEGIN キーワードの後にセミコロンを使用しないでください。

すべてのキーワードと識別子は大文字と小文字を混用して記述できます。識別子は、二重引用符で囲まれていない場合、暗黙で小文字に変換されます。

二重ハイフン (--) は、行末までのコメントを開始します。/* は、次の */ が発生するまでのブロックコメントを開始します。ブロックコメントはネストできません。ただし、二重ハイフンコメントはブロックコメントで囲むことができます。また、二重ハイフンはブロックコメントの区切り記号 /* および */ を隠すことができます。

ブロックのステートメントセクションのいずれのステートメントもサブブロックにすることができます。サブブロックを使用して論理グループを構成したり、変数を小グループのステートメントにローカライズしたりできます。

```
[ <<label>> ]
[ DECLARE
  declarations ]
BEGIN
  statements
END [ label ];
```

ブロックに先行する宣言セクションで宣言された変数は、ブロックが入力されるたびにデフォルト値に初期化されます。つまり、初期化される回数は関数呼び出しごとに 1 回とは限りません。

例を以下に示します。

```
CREATE PROCEDURE update_value() AS $$
DECLARE
  value integer := 20;
BEGIN
  RAISE NOTICE 'Value here is %', value;  -- Value here is 20
  value := 50;
  --
  -- Create a subblock
  --
  DECLARE
    value integer := 80;
  BEGIN
    RAISE NOTICE 'Value here is %', value;  -- Value here is 80
  END;

  RAISE NOTICE 'Value here is %', value;  -- Value here is 50
END;
$$ LANGUAGE plpgsql;
```

EXIT ステートメントで使用するブロックを識別したり、ブロックで宣言する変数の名前を修飾したりするには、ラベルを使用します。

PL/pgSQL でステートメントをグループ分けする BEGIN/END と、トランザクションコントロール用のデータベースコマンドを混同しないでください。PL/pgSQL の BEGIN および END は単にグループ分けするためのものです。トランザクションを開始または終了することはありません。

変数宣言

ループ変数を例外として、ブロック内のすべての変数をブロックの DECLARE セクションで宣言します。変数は任意の有効な Amazon Redshift データ型を使用できます。サポートされているデータ型については、データ型 を参照してください。

PL/pgSQL 変数は Amazon Redshift でサポートされている任意のデータ型と、さらに RECORD および refcursor を使用できます。RECORD の詳細については、レコード型 を参照してください。refcursor の詳細については、カーソル を参照してください。

```
DECLARE
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expression ];
```

変数宣言の例を以下に示します。

```
customerID integer;
numberofitems numeric(6);
link varchar;
onerow RECORD;
```

整数の範囲を反復処理する FOR ループのループ変数は、自動的に整数変数として宣言されます。

DEFAULT 句 (ある場合) は、ブロックの入力時に変数に代入する初期値を指定します。DEFAULT 句がない場合、変数は SQL の NULL 値に初期化されます。CONSTANT オプションは、変数への代入を禁止して、ブロックの存続期間中、変数の値を一定に維持します。NOT NULL を指定すると、NULL 値を代入したときにランタイムエラーが発生します。NOT NULL として宣言されたすべての変数には、NULL 以外のデフォルト値を指定する必要があります。

デフォルト値はブロックを入力するたびに評価されます。例えば、now() を timestamp 型の変数に代入すると、変数は関数がプリコンパイルされた時刻ではなく、現在の関数呼び出しの時刻になります。

```
quantity INTEGER DEFAULT 32;
url VARCHAR := 'http://mysite.com';
user_id CONSTANT INTEGER := 10;
```

refcursor データ型は、ストアドプロシージャ内のカーソル変数のデータ型です。refcursor の値はストアドプロシージャ内から返される場合があります。詳細については、「ストアドプロシージャから結果セットを返す」を参照してください

エイリアス宣言

ストアドプロシージャの署名から引数名が省略されている場合、その引数のエイリアスを宣言できます。

```
name ALIAS FOR $n;
```

組み込み変数

以下の組み込み変数がサポートされています。

- FOUND
- SQLSTATE
- SQLERRM
- GET DIAGNOSTICS integer_var := ROW_COUNT;

FOUND は Boolean 型の特殊な変数です。FOUND は各プロシージャ呼び出しで最初は false に設定されます。FOUND は以下のタイプのステートメントで設定されます。

- SELECT INTO

  行を返す場合は FOUND を true に設定し、行を返さない場合は false に設定します。
- UPDATE、INSERT、および DELETE

  少なくとも 1 つの行が影響を受ける場合は FOUND を true に設定し、どの行も影響を受けない場合は false に設定します。
- FETCH

  行を返す場合は FOUND を true に設定し、行を返さない場合は false に設定します。
- FOR ステートメント

  FOR ステートメントが 1 回以上反復処理を行う場合は FOUND を true に設定し、それ以外の場合は false に設定します。これは FOR ステートメントの 3 つすべてのバリアント (整数 FOR ループ、レコードセット FOR ループ、動的レコードセット FOR ループ) に該当します。

  FOUND は FOR ループの終了時に設定されます。ループのランタイム内では、FOUND は FOR ステートメントによって変更されません。ただし、ループ本体内の他のステートメントの実行によって変更される場合があります。

例を以下に示します。

```
CREATE TABLE employee(empname varchar);
CREATE OR REPLACE PROCEDURE show_found()
AS  $$
DECLARE
  myrec record;
BEGIN
  SELECT INTO myrec * FROM employee WHERE empname = 'John';
  IF NOT FOUND THEN
    RAISE EXCEPTION 'employee John not found';
  END IF;
END;
$$ LANGUAGE plpgsql;
```

例外ハンドラ内では、特殊な変数 SQLSTATE に、発生した例外に対応するエラーコードが含まれます。特殊な変数 SQLERRM には、例外に関連するエラーメッセージが含まれます。これらの変数は、例外ハンドラー外では未定義であり、使用した場合はエラーが表示されます。

例を以下に示します。

```
CREATE OR REPLACE PROCEDURE sqlstate_sqlerrm() AS
$$
BEGIN
  UPDATE employee SET firstname = 'Adam' WHERE lastname = 'Smith';
  EXECUTE 'select invalid';
  EXCEPTION WHEN OTHERS THEN
  RAISE INFO 'error message SQLERRM %', SQLERRM;
  RAISE INFO 'error message SQLSTATE %', SQLSTATE;
END;
$$ LANGUAGE plpgsql;
```

ROW_COUNT が GET DIAGNOSTICS コマンドと一緒に使用されています。これは、SQL エンジンに送信された最後の SQL コマンドで処理された行の数を示します。

例を以下に示します。

```
CREATE OR REPLACE PROCEDURE sp_row_count() AS
$$
DECLARE
  integer_var int;
```

```
BEGIN
  INSERT INTO tbl_row_count VALUES(1);
  GET DIAGNOSTICS integer_var := ROW_COUNT;
  RAISE INFO 'rows inserted = %', integer_var;
END;
$$ LANGUAGE plpgsql;
```

レコード型

RECORD 型は、実際のデータ型ではなく、単なるプレースホルダです。レコード型の変数は、SELECT または FOR コマンドの実行中に割り当てられた行の実際の行構造を取ります。レコード変数のサブ構造は、値が割り当てられるたびに変わる場合があります。レコード変数が最初に割り当てられるまでは、変数にサブ構造はありません。サブ構造のフィールドにアクセスしようとすると、ランタイムエラーが発生します。

```
name RECORD;
```

例を以下に示します。

```
CREATE TABLE tbl_record(a int, b int);
INSERT INTO tbl_record VALUES(1, 2);
CREATE OR REPLACE PROCEDURE record_example()
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
BEGIN
  FOR rec IN SELECT a FROM tbl_record
  LOOP
    RAISE INFO 'a = %', rec.a;
  END LOOP;
END;
$$;
```

サポートされている PL/pgSQL ステートメント

PL/pgSQL ステートメントは、SQL コマンドをプロシージャ構造体 (ループや条件式など) で強化し、論理フローを制御します。大半の SQL コマンドを使用できます。これには、COPY、UNLOAD、INSERT などのデータ操作言語 (DML) と CREATE TABLE などのデータ定義言語 (DDL) が含まれます。包括的な SQL コマンドのリストについては、「SQL コマンド」を参

照してください。さらに、以下の PL/pgSQL ステートメントが Amazon Redshift でサポートされています。

トピック

代入

代入ステートメントは変数に値を代入します。式は単一の値を返す必要があります。

```
identifier := expression;
```

= の代わりに、標準ではない := を代入に使用することもできます。

式のデータ型が変数のデータ型に一致しない場合や、変数にサイズや精度が含まれている場合、結果の値は暗黙で変換されます。

次に例を示します。

```
customer_number := 20;
tip := subtotal * 0.15;
```

SELECT INTO

SELECT INTO ステートメントは、複数の列 (ただし 1 つの行) の結果をレコード変数、またはスカラー変数のリスト内に割り当てます。

```
SELECT INTO target select_expressions FROM ...;
```

上の構文で、*target* はレコード変数であるか、カンマ区切りのシンプルな変数やレコードフィールドのリストです。*select_expressions* リストおよびコマンドの残りは、通常の SQL と同じです。

変数リストを *target* として使用する場合、選択する値はターゲットの構造と厳密に一致する必要があります。そうでないと、ランタイムエラーが発生します。レコード変数がターゲットである場合、レコード変数はクエリの結果列の行タイプに自動的に設定されます。

INTO 句は、SELECT ステートメントのほとんどあらゆる場所で使用できます。通常は、SELECT 句の直後または FROM 句の直前で使用します。つまり、*select_expressions* リストの直前または直後に使用します。

クエリが 0 行を返した場合は、 NULL 値が *target* に割り当てられます。クエリが複数の行を返した場合は、最初の行が *target* に割り当てられ、残りは破棄されます。ステートメントに ORDER BY が含まれている場合を除いて、最初の行は決定的ではありません。

割り当てが少なくとも 1 行を返したかどうかを確認するには、特殊な FOUND 変数を使用します。

```
SELECT INTO customer_rec * FROM cust WHERE custname = lname;
IF NOT FOUND THEN
  RAISE EXCEPTION 'employee % not found', lname;
END IF;
```

レコード結果が NULL であるかどうかをテストするには、IS NULL 条件を使用できます。その他の行が廃棄されたかどうかを確認する方法はありません。次の例は、行が 1 つも返されない場合を示しています。

```
CREATE OR REPLACE PROCEDURE select_into_null(return_webpage OUT varchar(256))
AS $$
DECLARE
  customer_rec RECORD;
BEGIN
  SELECT INTO customer_rec * FROM users WHERE user_id=3;
  IF customer_rec.webpage IS NULL THEN
    -- user entered no webpage, return "http://"
    return_webpage = 'http://';
  END IF;
END;
```

```
$$ LANGUAGE plpgsql;
```

no-op

no-op ステートメント (NULL;) は何もしないプレースホルダステートメントです。no-op ステートメントは、IF-THEN-ELSE チェーンの 1 つの分岐が空であることを示す場合があります。

```
NULL;
```

動的 SQL

PL/pgSQL ストアドプロシージャから実行するたびに関与するテーブルやデータ型が異なる動的コマンドを生成するには、EXECUTE ステートメントを使用します。

```
EXECUTE command-string [ INTO target ];
```

上の構文で、*command-string* は実行対象のコマンドを含む文字列 (テキスト型) を生成する式です。この *command-string* の値は SQL エンジンに送信されます。コマンド文字列で PL/pgSQL 変数の代入は行われません。変数の値は、コマンド文字列の構築時に挿入する必要があります。

> **Note**
>
> 動的 SQL 内から COMMIT ステートメント、および ROLLBACK ステートメントを使用することはできません。ストアドプロシージャ内での COMMIT ステートメントおよび ROLLBACK ステートメントの使用方法については、[トランザクションの管理](#) を参照してください。

動的コマンドを使用する場合は、単一引用符のエスケープ処理が必要になることがあります。ドル引用符付けを使用して関数本体の固定テキストを引用符で囲むことをお勧めします。構築されたクエリ内に挿入する動的な値は、それ自体に引用符が含まれている場合があるため、特別な処理が必要になります。次の例では、関数全体のドル引用符付けを行うため、引用符を二重化する必要はありません。

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
  || ' = '
  || quote_literal(newvalue)
  || ' WHERE key = '
```

```
  || quote_literal(keyvalue);
```

上の例は、関数 quote_ident(text) と関数 quote_literal(text) を示しています。この例では、列およびテーブルの識別子を含む変数を quote_ident 関数に渡します。また、構築されたコマンドにリテラル文字列を含む変数を quote_literal 関数に渡します。どちらの関数も適切なステップに従って二重引用符または単一引用符で囲まれた入力テキストをそれぞれ返します。埋め込まれた特殊文字は適切にエスケープされます。

ドル引用符付けは固定テキストの引用符付けに限り有効です。上の例を次の形式で記述しないでください。

```
EXECUTE 'UPDATE tbl SET '
  || quote_ident(colname)
  || ' = $$'
  || newvalue
  || '$$ WHERE key = '
  || quote_literal(keyvalue);
```

これを行うと、newvalue の内容に $$ が含まれていた場合に、この例は破綻します。同じ問題は、他のすべてのドル引用符区切り記号を選択した場合にも該当します。事前に確認できないテキストに対して安全に引用符付けを行うには、quote_literal 関数を使用します。

return

RETURN ステートメントは、ストアドプロシージャから呼び出し元に戻ります。

```
RETURN;
```

例を以下に示します。

```
CREATE OR REPLACE PROCEDURE return_example(a int)
AS $$
BEGIN
  FOR b in 1..10 LOOP
    IF b < a THEN
      RAISE INFO 'b = %', b;
    ELSE
      RETURN;
    END IF;
  END LOOP;
END;
```

```
$$ LANGUAGE plpgsql;
```

条件: IF

IF 条件ステートメントは、Amazon Redshift で使用する PL/pgSQL 言語で以下の形式を取ることができます。

- IF ..。THEN

  ```
  IF boolean-expression THEN
    statements
  END IF;
  ```

  例を以下に示します。

  ```
  IF v_user_id <> 0 THEN
    UPDATE users SET email = v_email WHERE user_id = v_user_id;
  END IF;
  ```

- IF ..。THEN ..。ELSE

  ```
  IF boolean-expression THEN
    statements
  ELSE
    statements
  END IF;
  ```

  例を以下に示します。

  ```
  IF parentid IS NULL OR parentid = ''
  THEN
    return_name = fullname;
    RETURN;
  ELSE
    return_name = hp_true_filename(parentid) || '/' || fullname;
    RETURN;
  END IF;
  ```

- IF ..。THEN ..。ELSIF ..。THEN ..。ELSE

  キーワード ELSIF は ELSEIF と表記される場合もあります。

```
IF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
[ ELSIF boolean-expression THEN
  statements
    ...] ]
[ ELSE
  statements ]
END IF;
```

例を以下に示します。

```
IF number = 0 THEN
  result := 'zero';
ELSIF number > 0 THEN
  result := 'positive';
ELSIF number < 0 THEN
  result := 'negative';
ELSE
  -- the only other possibility is that number is null
  result := 'NULL';
END IF;
```

条件: CASE

CASE 条件ステートメントは、Amazon Redshift で使用する PL/pgSQL 言語で以下の形式を取ることができます。

- シンプル CASE

```
CASE search-expression
WHEN expression [, expression [ ... ]] THEN
  statements
[ WHEN expression [, expression [ ... ]] THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

シンプル CASE ステートメントは、オペランドの等値に基づいて条件実行を提供します。

search-expression の値は、1 回評価され、WHEN 句の *expression* と順次比較されます。一致が見つかると、対応する *statements* が実行されます。次に、END CASE の後に続くステートメントに制御が渡されます。後続の WHEN expressions は評価されません。一致が見つからない場合は、ELSE *statements* が実行されます。ただし、ELSE が存在しない場合は、CASE_NOT_FOUND 例外が発生します。

例を以下に示します。

```
CASE x
WHEN 1, 2 THEN
  msg := 'one or two';
ELSE
  msg := 'other value than one or two';
END CASE;
```

• 検索 CASE

```
CASE
WHEN boolean-expression THEN
  statements
[ WHEN boolean-expression THEN
  statements
  ... ]
[ ELSE
  statements ]
END CASE;
```

検索形式の CASE は、Boolean 式の真理値に基づいて条件実行を提供します。

WHEN 句の *boolean-expression* のいずれかが真を生成するまで順に評価されます。次に、対応するステートメントが実行され、その後に END CASE に続くステートメントに制御が渡されます。後続の WHEN *expressions* は評価されません。真の結果が見つからない場合は、ELSE *statements* が実行されます。ただし、ELSE が存在しない場合は、CASE_NOT_FOUND 例外が発生します。

例を以下に示します。

```
CASE
```

```
WHEN x BETWEEN 0 AND 10 THEN
  msg := 'value is between zero and ten';
WHEN x BETWEEN 11 AND 20 THEN
  msg := 'value is between eleven and twenty';
END CASE;
```

loop

loop ステートメントは、Amazon Redshift で使用する PL/pgSQL 言語で以下の形式を取ることができます。

- simple loop

```
[<<label>>]
LOOP
  statements
END LOOP [ label ];
```

  simple loop は、EXIT または RETURN ステートメントによって終了されるまで無制限に繰り返される無条件ループを定義します。EXIT および CONTINUE ステートメントは、ネストされたループ内でオプションのラベルを使用することで、参照先のループを指定できます。

  例を以下に示します。

```
CREATE OR REPLACE PROCEDURE simple_loop()
LANGUAGE plpgsql
AS $$
BEGIN
  <<simple_while>>
  LOOP
    RAISE INFO 'I am raised once';
    EXIT simple_while;
    RAISE INFO 'I am not raised';
  END LOOP;
  RAISE INFO 'I am raised once as well';
END;
$$;
```

- Exit loop

```
EXIT [ label ] [ WHEN expression ];
```

label が存在しない場合は、最も内側のループが終了され、END LOOP に続くステートメントが次に実行されます。*label* が存在する場合、それはネストされたループやブロックの現在のレベルまたは他の外側のレベルのラベルであることが必要です。次に、指名されたループやブロックが終了され、そのループやブロックの対応する END に続くステートメントに制御が移ります。

WHEN が指定されている場合は、*expression* が真である場合に限り、ループの終了が起こります。それ以外の場合は、EXIT の後に続くステートメントに制御が移ります。

EXIT はすべてのタイプのループで使用できます。無条件ループでの使用に限定されません。

BEGIN ブロックと一緒に使用した場合、EXIT はブロックの最後に続くステートメントに制御を渡します。この目的のためにラベルを使用する必要があります。ラベルなしの EXIT は、BEGIN ブロックに一致するとは見なされません。

例を以下に示します。

```
CREATE OR REPLACE PROCEDURE simple_loop_when(x int)
LANGUAGE plpgsql
AS $$
DECLARE i INTEGER := 0;
BEGIN
  <<simple_loop_when>>
  LOOP
    RAISE INFO 'i %', i;
    i := i + 1;
    EXIT simple_loop_when WHEN (i >= x);
  END LOOP;
END;
$$;
```

- Continue loop

```
CONTINUE [ label ] [ WHEN expression ];
```

label が存在しない場合、実行は最も内側であるループの次の反復処理にジャンプします。つまり、ループ本体に残っているすべてのステートメントはスキップされます。この場合、制御はルー

プ制御式 (ある場合) に戻り、別のループ反復処理が必要であるかどうかが確認されます。*label* が存在する場合は、実行を継続するループのラベルが指定されます。

WHEN が指定されている場合は、*expression* が真であるときに限り、次のループ反復処理が開始されます。指定されていない場合は、CONTINUE の後に続くステートメントに制御が移ります。

CONTINUE はすべてのタイプのループで使用できます。無条件ループでの使用に限定されません。

```
CONTINUE mylabel;
```

- WHILE loop

```
[<<label>>]
WHILE expression LOOP
  statements
END LOOP [ label ];
```

WHILE ステートメントは、*boolean-expression* の評価が真である間は、ステートメントのシーケンスを繰り返します。式は、ループ本体に入る直前にチェックされます。

例を以下に示します。

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
  -- some computations here
END LOOP;

WHILE NOT done LOOP
  -- some computations here
END LOOP;
```

- FOR loop (整数バリアント)

```
[<<label>>]
FOR name IN [ REVERSE ] expression .. expression LOOP
  statements
END LOOP [ label ];
```

FOR loop (整数バリアント) は、整数値の範囲を反復処理するループを作成します。変数 name は整数型として自動的に定義され、ループ内にのみ存在します。変数 name の既存の定義はループ内では無視されます。範囲の下限と上限を指定する 2 つの式は、ループに入るときに 1 回評価されます。EVERSE を指定すると、各反復処理後に、ステップ値は加算されずに減算されます。

下限が上限より大きい (REVERSE の場合はより小さい) と、ループ本体は実行されません。エラーは発生しません。

ラベルが FOR ループにアタッチされている場合は、そのラベルを使用して、修飾名で整数ループ変数を参照できます。

例を以下に示します。

```
FOR i IN 1..10 LOOP
  -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop
END LOOP;

FOR i IN REVERSE 10..1 LOOP
  -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop
END LOOP;
```

- FOR loop (結果セットバリアント)

```
[<<label>>]
FOR target IN query LOOP
  statements
END LOOP [ label ];
```

target はレコード変数であるか、カンマ区切りのスカラー変数のリストです。target には、query の結果である各行が順次代入され、行ごとにループ本体が実行されます。

FOR loop (結果セットバリアント) を使用すると、ストアドプロシージャはクエリの結果を反復処理し、そのデータを適切に操作できます。

例を以下に示します。

```
CREATE PROCEDURE cs_refresh_reports() AS $$
DECLARE
  reports RECORD;
BEGIN
```

```
  FOR reports IN SELECT * FROM cs_reports ORDER BY sort_key LOOP
    -- Now "reports" has one record from cs_reports
    EXECUTE 'INSERT INTO ' || quote_ident(reports.report_name) || ' ' ||
 reports.report_query;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

- FOR loop (動的 SQL を使用)

```
[<<label>>]
FOR record_or_row IN EXECUTE text_expression LOOP
  statements
END LOOP;
```

動的 SQL を使用する FOR loop の場合、ストアドプロシージャは動的クエリの結果を反復処理し、そのデータを適切に処理できます。

例を以下に示します。

```
CREATE OR REPLACE PROCEDURE for_loop_dynamic_sql(x int)
LANGUAGE plpgsql
AS $$
DECLARE
  rec RECORD;
  query text;
BEGIN
  query := 'SELECT * FROM tbl_dynamic_sql LIMIT ' || x;
  FOR rec IN EXECUTE query
  LOOP
    RAISE INFO 'a %', rec.a;
  END LOOP;
END;
$$;
```

カーソル

クエリ全体を一度に実行せずに、カーソルを設定できます。カーソル は、クエリをカプセル化し、クエリの結果を一度に数行ずつ読み取ります。これを行う理由の 1 つは、結果内に多数の行がある場合のメモリの枯渇を防ぐことです。別の理由は、呼び出し元が行を読み取ることができるように、

ストアドプロシージャが作成したカーソルへの参照を返すことです。これにより、ストアドプロシージャから大きな行セットを返す際の効率が向上します。

NONATOMIC ストアドプロシージャでカーソルを使用するには、START TRANSACTION...COMMIT の間にカーソルループを置きます。

カーソルを設定するには、まずカーソル変数を宣言します。PL/pgSQL では、カーソルへのすべてのアクセスがカーソル変数を経由します。カーソル変数は常に特殊な `refcursor` データ型です。`refcursor` データ型は単にカーソルへの参照を保持します。

カーソル変数を作成するには、`refcursor` 型の変数としてカーソルを宣言します。または、次に示すカーソル宣言構文を使用することもできます。

```
name CURSOR [ ( arguments ) ] FOR query ;
```

上の構文で、*`arguments`* (指定されている場合) はカンマ区切りの *`name datatype`* ペアのリストです。各ペアは、*`query`* のパラメータ値で置き換えられる名前を定義します。これらの名前と置き換わる実際の値は、後で、カーソルを開いたときに指定されます。

次に例を示します。

```
DECLARE
  curs1 refcursor;
  curs2 CURSOR FOR SELECT * FROM tenk1;
  curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;
```

これら 3 つすべての変数は `refcursor` データ型を持ちますが、最初の変数はすべてのクエリで使用できます。一方、2 番目の変数には完全に指定されたクエリがバインドされており、3 番目の変数にはパラメータ化クエリがバインドされています。key 値は、カーソルを開いたときに、整数パラメータ値に置き換えられます。変数 `curs1` は、特定のクエリにバインドされていないため、非バインドと呼ばれます。

カーソルを使用して行を取得する前に、カーソルを開く必要があります。PL/pgSQL には 3 つの形式の OPEN ステートメントがあります。そのうちの 2 つでは、非バインドカーソル変数を使用し、残りの 1 つではバインドカーソル変数を使用します。

- Open for select: カーソル変数を開き、これに対して実行するクエリを指示します。すでに開いているカーソルを開くことはできません。また、カーソルは非バインドカーソル (つまり、シンプル

な refcursor 変数) として宣言済みであることが必要です。SELECT クエリは、PL/pgSQL の他の SELECT ステートメントと同じ方法で扱われます。

```
OPEN cursor_name FOR SELECT ...;
```

例を以下に示します。

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
```

- Open for execute: カーソル変数を開き、これに対して実行するクエリを指示します。すでに開いているカーソルを開くことはできません。また、カーソルは非バインドカーソル (つまり、シンプルな refcursor 変数) として宣言済みであることが必要です。クエリは、EXECUTE コマンドと同じ方法で、文字列式として指定されます。これに伴う柔軟性により、今回実行したものとは異なるクエリを次回に実行できます。

```
OPEN cursor_name FOR EXECUTE query_string;
```

例を以下に示します。

```
OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident($1);
```

- Open a bound cursor: この形式の OPEN では、宣言時にクエリがバインド済みであるカーソル変数を開きます。すでに開いているカーソルを開くことはできません。実際の引数値式が表示されるのは、カーソルが引数を取るように宣言済みである場合に限る必要があります。これらの値はクエリ内で置き換えられます。

```
OPEN bound_cursor_name [ ( argument_values ) ];
```

例を以下に示します。

```
OPEN curs2;
OPEN curs3(42);
```

カーソルが開いたら、以下に説明するステートメントを使用してカーソルを操作できます。これらのステートメントは、カーソルを開いた同じストアドプロシージャで使用する必要はありません。ストアドプロシージャから refcursor 値を返して、呼び出し元でカーソルを操作できます。すべての

ポータルは、トランザクションの終わりに暗黙で閉じられます。したがって、`refcursor` 値を使用して開いているカーソルを参照できるのは、トランザクションの終わりまでです。

- FETCH は、カーソルから次に続く行を取得して target に格納します。この target は、SELECT INTO の場合と同じように、行変数、レコード変数、カンマ区切りのシンプルな変数のリストのいずれかです。SELECT INTO の場合と同様に、特殊な FOUND 変数をチェックして、行が取得されたかどうかを確認できます。

  ```
  FETCH cursor INTO target;
  ```

  例を以下に示します。

  ```
  FETCH curs1 INTO rowvar;
  ```

- CLOSE は、開いているカーソルの基になっているポータルを閉じます。このステートメントを使用してトランザクションの終了より前にリソースを解放できます。また、このステートメントを使用してカーソル変数を解放し、再度開くこともできます。

  ```
  CLOSE cursor;
  ```

  例を以下に示します。

  ```
  CLOSE curs1;
  ```

RAISE

`RAISE level` ステートメントを使用して、メッセージを報告してエラーをレイズします。

```
RAISE level 'format' [, variable [, ...]];
```

使用可能なレベルは、NOTICE、INFO、LOG、WARNING、および EXCEPTION です。EXCEPTION はエラーを発生させ、このエラーは通常、現行のトランザクションをキャンセルします。その他のレベルでは、さまざまな優先度レベルのメッセージのみを生成します。

format 文字列内の % は、次の省略可能な引数の文字列表現に置き換えられます。リテラル % を表すには「%%」と記述します。現在、省略可能な引数はシンプルな変数にする必要があり (式ではなく)、format はシンプルな文字列リテラルにする必要があります。

次の例では、文字列内の % が v_job_id の値に置き換えられます。

```
RAISE NOTICE 'Calling cs_create_job(%)', v_job_id;
```

RAISE ステートメントを使用して、例外処理ブロックでキャッチされた例外を再スローします。このステートメントは、NONATOMIC モードのストアドプロシージャの例外処理ブロックでのみ有効です。

```
RAISE;
```

トランザクションの制御

Amazon Redshift で使用する PL/pgSQL 言語のトランザクション制御ステートメントを使用できます。ストアドプロシージャ内での COMMIT、ROLLBACK、TRUNCATE ステートメントの使用方法については、トランザクションの管理 を参照してください。

NONATOMIC モードのストアドプロシージャでは、START TRANSACTION を使用してトランザクションブロックを開始します。

```
START TRANSACTION;
```

> **Note**
>
> PL/pgSQL ステートメントの START TRANSACTION は、SQL コマンドの START TRANSACTION と次の点で異なります。
>
> - ストアドプロシージャ内では、START TRANSACTION は BEGIN と同義ではありません。
> - PL/pgSQL ステートメントは、オプションの分離レベルとアクセス許可キーワードをサポートしていません。

Amazon Redshift でのマテリアライズドビュー

このセクションでは、Amazon Redshift でマテリアライズドビューを作成して使用する方法について説明します。マテリアライズドビューとは、クエリの結果を保存するデータベースオブジェクトであり、パフォーマンスと効率を向上させるために使用できます。

データウェアハウスの環境でのアプリケーションでは、多く場合、大規模なテーブルに対し複雑なクエリを実行する必要が生じます。例えば SELECT ステートメントでは、数百万の行を含むテーブルにおいて、複数の テーブルの結合や集計が行われます。このようなクエリの処理は、システムリソース、そして結果を返すためにかかる時間の点から、コストが高くなる可能性があります。

Amazon Redshift のマテリアライズドビューを使用することで、こうした課題に対処できます。マテリアライズドビューには、1 つ以上のベーステーブルで実行された SQL クエリに基づいて事前計算された結果が含まれています。SELECT ステートメントを使用すれば、データベースで他のテーブルやビューをクエリするのと同じ方法でマテリアライズドビューをクエリすることができます。Amazon Redshift は、ベーステーブルに一切アクセスすることなく、マテリアライズドビューから事前計算された結果を返します。ユーザーの観点から見て、クエリの結果は、ベーステーブルから同じデータを取得するのにかかる時間に比べてはるかに短時間で返ってきます。

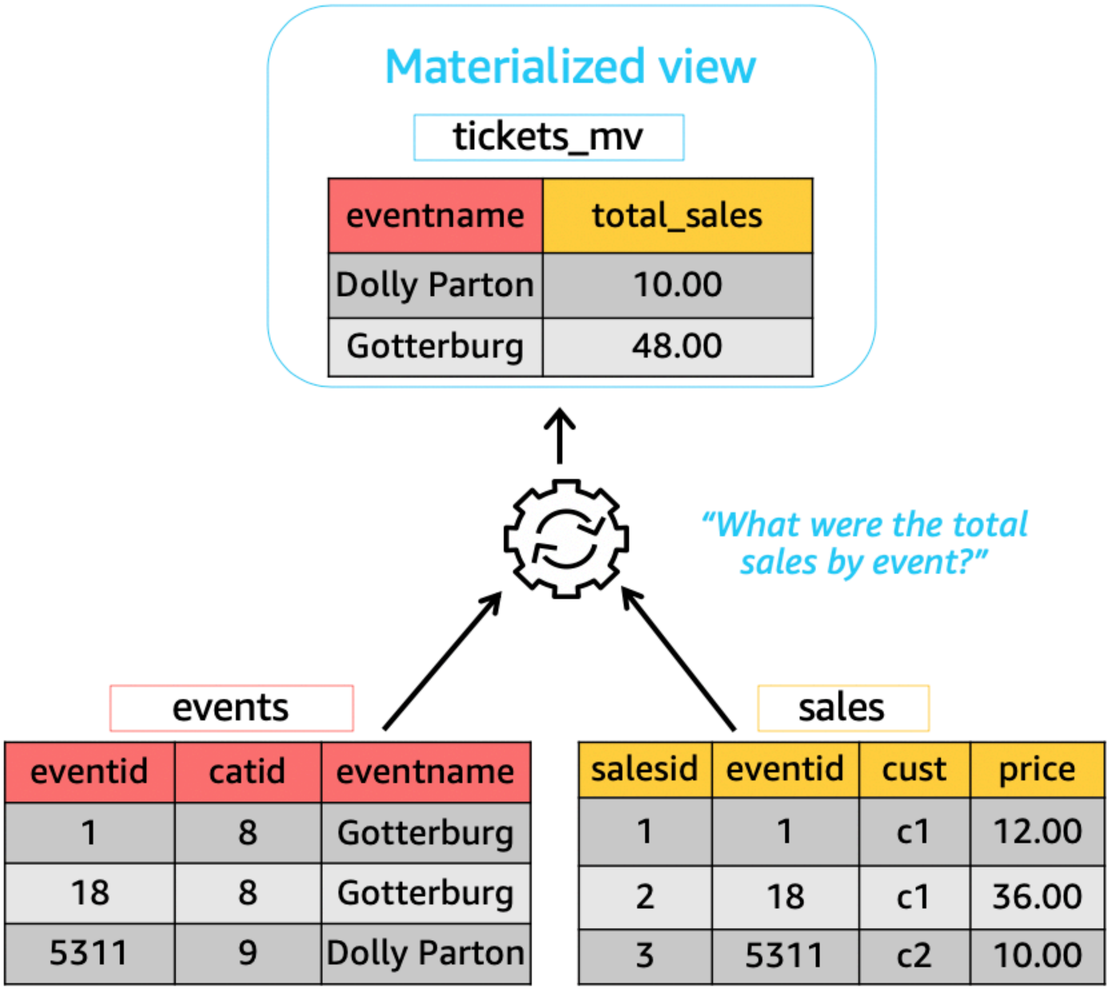
マテリアライズドビューは、予期可能で繰り返し実行されるクエリの速度を上げるために特に役立ちます。アプリケーションは、集計や複数の結合など、大きなテーブルに対してリソースを大量に消費するクエリを実行する代わりに、マテリアライズドビューをクエリし、事前計算された結果を取得することができます。例えば、Amazon QuickSight などのダッシュボードにデータを入力するために一連のクエリを使用するシナリオについて考えてみます。このユースケースは、クエリが定期的で何度も繰り返されることから、マテリアライズドビューに適しています。

マテリアライズドビューは、他のマテリアライズドビューに関連させて定義できます。マテリアライズドビュー上のマテリアライズドビューを使用して、マテリアライズドビューの機能を拡張できます。この手法では、既存のマテリアライズドビューは、クエリがデータを取得するためのベーステーブルと同じ役割を果たします。

この手法はさまざまな集計オプションや GROUP BY オプションのために事前計算されている結合を再利用する場合に特に便利です。例えば、顧客情報 (数百万行を含む) と品目オーダー詳細情報 (数十億行を含む) を結合するための、マテリアライズドビューを考えてみます。これは、オンデマンドで繰り返し計算を行うために、高価なクエリとなります。既存のマテリアライズドビューの上に作成されたマテリアライズドビューに対して、異なるGROUP BYオプションを使用することで、異なるテーブルを結合することができます。これにより、コストの高い結合を、基盤レベルで毎回実行する

ために消費される計算時間を節約できます。1 つのマテリアライズドビューの、他のマテリアライズドビューに対する依存関係は、STV_MV_DEPSテーブルに記述されます。

マテリアライズドビューを作成すると、Amazon Redshift は、ユーザーが指定した SQL ステートメントを実行します。そして、ベーステーブルまたはテーブルからデータを収集し、結果を保存します。次の図に、SQL クエリが 2 つのベーステーブル (`events` および `sales`) を使用して定義する、マテリアライズドビュー `tickets_mv` の概要を示します。



その後、これらのマテリアライズドビューをクエリで使用して、高速化できます。さらに、Amazon Redshift は、クエリが明示的にマテリアライズドビューを参照していない場合でも、これらのクエリを自動的に書き換えてマテリアライズドビューを使用できます。クエリの自動書き換えは、マテリアライズドビューを使用するためにクエリを変更できない場合に、パフォーマンスを向上させる上で特に強力です。

マテリアライズドビューのデータを更新するには、REFRESH MATERIALIZED VIEW ステートメントを使用して、マテリアライズドビューを手動で更新できます。Amazon Redshift は、ベーステーブ

ルまたはテーブルで行われた変更を特定し、特定した変更をマテリアライズドビューに適用します。クエリの自動書き換えでは、マテリアライズドビューは 最新状態 に保たれる必要があるため、マテリアライズドビューの所有者は、ベーステーブルが変更されるたびに対象のビューを更新する必要があります。

Amazon Redshift には、自動書き換えのためにマテリアライズドビューを 最新の状態 に保つための方法がいくつか用意されています。マテリアライズドビューのベーステーブルが更新されたときにマテリアライズドビューが更新されるよう自動更新オプションを使用して、マテリアライズドビューを構成できます。自動更新オペレーションは、クラスターリソースが使用可能なときに実行され、他のワークロードの中断を最小限に抑えます。自動更新のスケジュールはワークロードに依存するため、Amazon Redshift がマテリアライズドビューを更新するタイミングをより詳細に制御できます。Amazon Redshift スケジューラ API とコンソール統合を使用して、マテリアライズドビューの更新ジョブをスケジュールできます。クエリのスケジューリングの詳細については、Amazon Redshift コンソールでのクエリのスケジューリングを参照してください。

このスケジューリングは、マテリアライズドビューからの最新データに関し、サービスレベルアグリーメント (SLA) 要件がある場合に特に便利です。また、自動更新できるマテリアライズドビューを手動で更新することもできます。マテリアライズドビューの作成方法については、「CREATE MATERIALIZED VIEW」を参照してください。

SELECT ステートメントを発行して、マテリアライズドビューを照会することができます。マテリアライズドビューの照会方法については、「マテリアライズドビューのクエリ」を参照してください。こうして保存された結果は、ベーステーブルやテーブルでデータが挿入、更新、削除されるうちに古くなります。マテリアライズドビューは、いつでもリフレッシュして、ベーステーブルからの最新の変更で更新できます。マテリアライズドビューをリフレッシュする方法については、「REFRESH MATERIALIZED VIEW」を参照してください。

マテリアライズドビューを作成および管理するための SQL コマンドの詳細については、以下のコマンドトピックを参照してください。

- CREATE MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW
- REFRESH MATERIALIZED VIEW
- DROP MATERIALIZED VIEW

マテリアライズドビューを監視するシステムテーブルとビューに関しては、次のトピックを参照してください。

- STV_MV_INFO
- STL_MV_STATE
- SVL_MV_REFRESH_STATUS
- STV_MV_DEPS

トピック

マテリアライズドビューのクエリ

マテリアライズドビューは、テーブルやスタンダードビューなどのデータソースとしてマテリアライズドビューの名前を参照することにより、任意の SQL クエリで使用することができます。

クエリがマテリアライズドビューにアクセスする際は、最新の更新時にマテリアライズドビューに保存されているデータのみが参照されます。そのため、クエリは、マテリアライズドビューに対応するベーステーブルからのすべての最新の変更を参照しない場合があります。

他のユーザーがマテリアライズドビューをクエリする場合、このビューの所有者は、それらのユーザーに対し SELECT のアクセス許可を付与する必要があります。これら他のユーザーには、基盤のベーステーブルに対する SELECT のアクセス許可は必要ありません。マテリアライズドビューの所有者は、他のユーザーから SELECT のアクセス許可を取り消して、マテリアライズドビューへのクエリを拒否することもできます。

マテリアライズドビューの所有者が、基盤のベーステーブルに対する SELECT のアクセス許可を失った場合は、以下のような状態になります。

- 所有者はマテリアライズドビューをクエリできなくなります。
- マテリアライズドビューに対する SELECT のアクセス許可を持つ他のユーザーは、そのマテリアライズドビューをクエリできなくなります。

次の例では、マテリアライズドビュー tickets_mv をクエリします。マテリアライズドビューの作成に使用される SQL コマンドの詳細については、「CREATE MATERIALIZED VIEW」を参照してください。

```
SELECT sold
FROM tickets_mv
WHERE catgroup = 'Concerts';
```

クエリの結果が事前計算されているため、基となるテーブル (category、event、および sales) にアクセスする必要はありません。Amazon Redshift は、tickets_mvから直接結果を返すことができます。

マテリアライズドビューを使用するための自動クエリ書き換え

Amazon Redshift でマテリアライズドビューの自動クエリ書き換えを使用すると、その Amazon Redshift がマテリアライズドビューを使用するようにクエリを修正できます。これにより、マテリアライズドビューを明示的に参照しないクエリについても、そのワークロードが高速化されます。Amazon Redshift は、クエリを書き換える際に、最新 のマテリアライズドビューのみを使用します。

使用に関する注意事項

クエリに対してクエリの自動書き換えが使用されているかどうかを確認するには、クエリプランまたは STL_EXPLAIN を調べます。次に、元のクエリプランの SELECT ステートメントと、EXPLAIN 出力を示します。

```
SELECT catgroup, SUM(qtysold) AS sold
FROM category c, event e, sales s
WHERE c.catid = e.catid AND e.eventid = s.eventid
GROUP BY 1;

EXPLAIN
 XN HashAggregate  (cost=920021.24..920021.24 rows=1 width=35)
   ->  XN Hash Join DS_BCAST_INNER  (cost=440004.53..920021.22 rows=4 width=35)
         Hash Cond: ("outer".eventid = "inner".eventid)
         ->  XN Seq Scan on sales s  (cost=0.00..7.40 rows=740 width=6)
         ->  XN Hash  (cost=440004.52..440004.52 rows=1 width=37)
               ->  XN Hash Join DS_BCAST_INNER  (cost=0.01..440004.52 rows=1 width=37)
                     Hash Cond: ("outer".catid = "inner".catid)
                     ->  XN Seq Scan on event e  (cost=0.00..2.00 rows=200 width=6)
```

```
                    ->  XN Hash  (cost=0.01..0.01 rows=1 width=35)
                          ->  XN Seq Scan on category c  (cost=0.00..0.01 rows=1
 width=35)
```

次に、自動書き換えに成功した後の EXPLAIN からの出力を示します。この出力には、既存のクエリプランの一部を置き換えるクエリプランによる、マテリアライズドビューに対するスキャンが含まれます。

```
* EXPLAIN
     XN HashAggregate  (cost=11.85..12.35 rows=200 width=41)
       ->  XN Seq Scan on mv_tbl__tickets_mv__0 derived_table1  (cost=0.00..7.90
 rows=790 width=41)
```

自動、スケジュール済み、手動などの更新戦略に関係なく、クエリの自動書き換えには、最新の (新しい) マテリアライズドビューのみが考慮されます。したがって、元のクエリは最新の結果を返します。クエリでマテリアライズドビューが明示的に参照されている場合、Amazon Redshift はマテリアライズドビュー内に現在保存されているデータにアクセスします。このデータには、マテリアライズドビューのベーステーブルからの最新の変更が、反映されていない場合があります。

クラスターバージョン 1.0.20949 以降で作成されたマテリアライズドビューの自動クエリ書き換えを使用できます。

セッションレベルでの自動クエリ書き換えを停止する場合は、SET mv_enable_aqmv_for_session を FALSE に使用します。

制約事項

マテリアライズドビューでの、自動クエリ書き換えの使用に関する制限事項は次のとおりです。

- 自動クエリ書き換えは、以下のいずれかを参照しない、または含まないマテリアライズドビューで動作します。
  - サブクエリ
  - 左、右、またはフル外部結合
  - セットオペレーション
  - SUM、COUNT、MIN、MAX および AVG 以外の集計関数。(これらは、自動クエリ書き換えで動作する唯一の集計関数です。)
  - DISTINCT を使用する集計関数
  - Window 関数

  - SELECT DISTINCT または HAVING 句
  - 外部テーブル
  - その他のマテリアライズドビュー
- 自動クエリ書き換えにより、ユーザー定義の Amazon Redshift テーブルを参照する SELECT クエリが書き換えられます。Amazon Redshift では、次のクエリは書き換えられません。
  - CREATE TABLE AS ステートメント
  - SELECT INTO ステートメント
  - カタログまたはシステムテーブルに対するクエリ
  - 外部結合または SELECT DISTINCT 句を使用したクエリ
- クエリが自動的に書き換えられない場合は、指定したマテリアライズドビューに対する SELECT 許可があり、mv_enable_aqmv_for_session オプションが TRUE に設定されているかどうかを確認します。

  STV_MV_INFO を調べることで、マテリアライズドビューがクエリの自動書き換えに適しているかどうかを確認することもできます。詳細については、「STV_MV_INFO」を参照してください。

Amazon Redshift Spectrum の外部データレイクテーブルのマテリアライズドビュー

マテリアライズドビューでは、外部データレイクテーブルの増分メンテナンスを提供できます。増分メンテナンスでは、Amazon Redshift によって、前回の更新以降にベーステーブルのデータに加えられた変更のみをマテリアライズドビューのデータに反映します。増分メンテナンスは、ベーステーブルでデータが変更されるたびにマテリアライズドビューをすべて再計算するよりもコスト効率が高くなります。

少なくても 1 つの外部テーブルにマテリアライズドビューを使用している場合、マテリアライズドビューの作成は、次のものに対して増分となります。

- 標準データレイクテーブル (パーティション化あり/なし)、サポートされている任意の形式 (Parquet、Avro、CSV など) のデータファイル。
- Apache Iceberg テーブル (パーティション化あり/なし) は、コピーオンライトとマージオンリードです。
- Amazon Redshift Spectrum テーブルは、同じデータベース内の任意の Amazon Redshift テーブルと結合します。

マテリアライズドビューは、次のものに対して増分となります。

- マテリアライズドビューで集約を実行しない場合、S3 DELETE または PUT 上書き (データファイルの削除) 後の標準データレイクテーブル。
- INSERT、DELETE、UPDATE、またはテーブル圧縮後の Apache Iceberg テーブル。

Amazon Redshift Spectrum の詳細については、「Amazon Redshift Spectrum」を参照してください。

制約事項

データレイクテーブルのマテリアライズドビューには、引き続き、マテリアライズドビューに関する一般的な制限が適用されます。詳細については、「マテリアライズドビューの更新」を参照してください。また、外部データレイクテーブルでマテリアライズドビューを使用する場合は、次の制限を考慮してください。

- マテリアライズドビューの作成では、次のものは増分されません。
  - Hudi または Delta Lake テーブル。
  - ネストされた Spectrum のデータアクセス。
- マテリアライズドビューの更新は、次の場合に完全な再計算にフォールバックします。
  - マテリアライズドビューが集約を実行する場合、必要なスナップショットが期限切れになった Apache Iceberg テーブル。
  - マテリアライズドビューが集約を実行する場合、Amazon S3 のデータファイルの削除または更新後の標準データレイクテーブル。
  - トランザクションブロックで複数回更新された標準データレイクテーブル。
  - マニフェストによって管理される標準データレイクテーブル。マニフェストの詳細については、「マニフェストを使用し、データファイルを指定する」を参照してください。
  - Amazon Redshift は、パフォーマンスがさらに向上すると予想される場合、特に、結合を含むマテリアライズドビューで、最後の更新以降に 1 つ以上のベーステーブルが更新された場合、完全な再計算にフォールバックします。
- Apache Iceberg テーブルでは、マテリアライズドビューの更新は、1 つのデータファイルで削除された最大 400 万の位置までしか処理できません。この制限に達した場合、マテリアライズドビューの更新を続行するには、Apache Iceberg ベーステーブルを圧縮する必要があります。
- Apache Iceberg テーブルでは、マテリアライズドビューの作成および更新時に同時実行スケーリングはサポートされていません。

- Autonomics 機能はサポートされていません。これには、自動マテリアライズドビュー、自動更新、自動クエリ書き換えが含まれます。
- 増分マテリアライズドビューが更新されると、IAM アクセス許可は Amazon Redshift ベーステーブルのアクセス部分にのみに適用されます。

マテリアライズドビューの更新

このトピックでは、基になるテーブルからマテリアルビューのデータを更新する方法について説明します。

マテリアライズドビューを作成する際、そのコンテンツには、その時点での基となるデータベーステーブルまたはテーブルの状態が反映されます。基となるテーブルにあるデータが、アプリケーションにより変更されても、マテリアライズドビューのデータは変更されません。マテリアライズドビューのデータを更新する場合は、`REFRESH MATERIALIZED VIEW`ステートメントを使用してマテリアライズドビューを手動で随時更新できます。このステートメントを使用する際は、Amazon Redshift は、ベーステーブルまたはテーブルで行われた変更を特定し、特定した変更をマテリアライズドビューに適用します。

Amazon Redshift には、マテリアライズドビューを更新するための 2 つの方法があります。

- 多くの場合、Amazon Redshift は、増分更新を実行します。増分更新では、Amazon Redshift によって前回の更新からのベーステーブルのデータに対する変更がすぐに特定され、マテリアライズドビューのデータが更新されます。増分更新は、マテリアライズドビューの定義時にクエリで使用する以下の SQL コンストラクトでサポートされています。
  - SELECT 句、FROM 句、[INNER] JOIN 句、WHERE 句、GROUP BY 句、HAVING 句が含まれるコンストラクト。
  - SUM、MIN、MAX、AVG および COUNT などの集計が含まれるコンストラクト。
  - ほとんどの組み込み SQL 関数、特にイミュータブルな関数には、同じ入力引数を持ち、常に同じ出力が生成されることを前提としています。

  増分更新は、データ共有テーブルに基づくマテリアライズドビューでもサポートされています。
- 増分更新ができない場合、Amazon Redshift はフル更新を実行します。フル更新では、基となる SQL ステートメントを再実行し、マテリアライズドビューにあるすべてのデータを置き換えます。
- Amazon Redshift は、マテリアライズドビューの定義に使用される SELECT クエリに応じて、マテリアライズドビューの更新方法を自動的に選択します。

マテリアライズドビュー上のマテリアライズドビューをリフレッシュすることは、縦につながるプロセスではありません。例えば、マテリアライズドビュー B に依存するマテリアライズドビュー A があるとします。この場合、REFRESH MATERIZED VIEW A が呼び出されると、B が古くなっていても、A はその時点の B のバージョンを使用してリフレッシュされます。A を完全に 最新の状態 にするには、A をリフレッシュする前に、まず別のトランザクションで B をリフレッシュします。

次の例では、マテリアライズドビューの完全なリフレッシュプランを、プログラム的に作成する方法を示しています。マテリアライズドビュー v をリフレッシュするには、最初にマテリアライズドビュー u をリフレッシュします。マテリアライズドビュー w をリフレッシュするには、まずマテリアライズドビュー u を、次にマテリアライズドビュー v をリフレッシュします。

```
CREATE TABLE t(a INT);
CREATE MATERIALIZED VIEW u AS SELECT * FROM t;
CREATE MATERIALIZED VIEW v AS SELECT * FROM u;
CREATE MATERIALIZED VIEW w AS SELECT * FROM v;

WITH RECURSIVE recursive_deps (mv_tgt, lvl, mv_dep) AS
( SELECT trim(name) as mv_tgt, 0 as lvl, trim(ref_name) as mv_dep
  FROM stv_mv_deps
  UNION ALL
  SELECT R.mv_tgt, R.lvl+1 as lvl, trim(S.ref_name) as mv_dep
  FROM stv_mv_deps S, recursive_deps R
  WHERE R.mv_dep = S.name
)

SELECT mv_tgt, mv_dep from recursive_deps
ORDER BY mv_tgt, lvl DESC;

 mv_tgt | mv_dep
--------+--------
 v      | u
 w      | u
 w      | v
(3 rows)
```

次の例は、古くなったマテリアライズドビューに依存するマテリアライズドビューに対して、REFRESH MATERIZED VIEW を実行する際に表示される情報メッセージを示しています。

```
create table a(a int);
```

```
create materialized view b as select * from a;
```

```
create materialized view c as select * from b;
```

```
insert into a values (1);
```

```
refresh materialized view c;

INFO:  Materialized view c is already up to date.  However, it depends on another
 materialized view that is not up to date.
```

```
REFRESH MATERIALIZED VIEW b;
INFO:  Materialized view b was incrementally updated successfully.
```

```
REFRESH MATERIALIZED VIEW c;
INFO:  Materialized view c was incrementally updated successfully.
```

Amazon Redshift には、現時点では、マテリアライズドビューの増分更新に関する次の制限があります。

Amazon Redshift では、現在のところ、次の SQL 要素のいずれかを使用してクエリで定義されたマテリアライズドビューの増分更新はサポートされていません。

- OUTER JOIN (右、左、またはフル)。
- 集合演算のセットは、UNION、INTERSECT、EXCEPT、MINUS です。
- 集計関数は、MEDIAN、PERCENTILE_CONT、LISTAGG、STDDEV_SAMP、STDDEV_POP、APPROXIMATE COUNT、APPROXIMATE PERCENTILE、およびビット単位の集計関数です。

> **Note**
>
> COUNT、SUM および AVG 集計関数がサポートされています。

- DISTINCT COUNT、DISTINCT SUM などの DISTINCT 集計関数。
- ウィンドウ関数。
- 共通部分式の最適化など、クエリの最適化に一時テーブルを使用するクエリ。

- サブクエリ。
- マテリアライズドビューを定義するクエリで以下の形式を参照する外部テーブル。
  - Delta Lake
  - Hudi

  上記以外のフォーマットを使用して定義されたマテリアライズドビューでは、増分更新がサポートされます。詳細については、「Amazon Redshift Spectrum の外部データレイクテーブルのマテリアライズドビュー」を参照してください。

マテリアライズドビューの自動更新

Amazon Redshift は、自動更新オプションを使用してマテリアライズドビューを作成または変更すると、ベーステーブルの最新データでマテリアライズドビューを自動的に更新できます。Amazon Redshift は、ベーステーブルが変更された後、できるだけ早くマテリアライズドビューを自動更新します。

クラスター内のアクティブなワークロードへの影響を最小限に抑えながら、最も重要なマテリアライズドビューの更新を完了するために、Amazon Redshift は複数の要素を考慮します。これらの要素には、現在のシステム負荷、更新に必要なリソース、使用可能なクラスターリソース、マテリアライズドビューの使用頻度が含まれます。

Amazon Redshift は自動更新よりもワークロードの優先順位を設定し、ユーザーのワークロードのパフォーマンスを維持するために自動更新を停止する場合があります。このアプローチでは、一部のマテリアライズドビューの更新が遅れる可能性があります。場合によっては、マテリアライズドビューに対してより決定的な更新動作が必要になることがあります。その場合は、REFRESH MATERIALIZED VIEWで説明されている手動更新、またはAmazon Redshift スケジューラ API オペレーションまたはコンソールを使用したスケジュール更新の使用を検討してください。

CREATE MATERIALIZED VIEW を使用して、マテリアライズドビューの自動更新を設定できます。また、AUTO REFRESH 句を使用して、マテリアライズドビューを自動的に更新することもできます。マテリアライズドビュー作成の詳細については、「CREATE MATERIALIZED VIEW」を参照してください。現在のマテリアライズドビューの自動更新を有効にするには、ALTER MATERIALIZED VIEWを使用します。

マテリアライズドビューを更新するときは、次の点を考慮してください。

- マテリアライズドビューの自動更新を有効にしていない場合でも、REFRESH MATERIALIZED VIEW コマンドを使用してマテリアライズドビューを明示的に更新できます。

- Amazon Redshift では、外部テーブルで定義されているマテリアライズドビューは自動更新されません。
- 更新ステータスについては、SVL_MV_REFRESH_STATUS をチェックできます。これは、ユーザーが開始したクエリまたは自動更新されたクエリを記録します。
- 再計算専用のマテリアライズドビューで REFRESH を実行するには、スキーマに対する CREATE のアクセス許可が付与されている必要があります。詳細については、「GRANT」を参照してください。

自動マテリアライズドビュー

このトピックでは、Amazon Redshift が自動マテリアライズドビューを使用してパフォーマンスを向上させる方法について説明します。Amazon Redshift は、データベースのアクティビティとパフォーマンスに基づいてマテリアライズドビューを自動的に作成します。Amazon Redshift は、デフォルトで自動マテリアライズドビューを使用します。

マテリアライズドビューは、Amazon Redshift でのクエリのパフォーマンスを向上させるための強力なツールです。これは、事前に計算された結果セットを保存することによって実現されます。既存の結果セットからレコードを取得できるため、類似したクエリが、毎回同じロジックを再実行する必要はありません。デベロッパーとアナリストは、ワークロードを分析した後にマテリアライズドビューを作成します。これにより、どのクエリが利益をもたらすか、各マテリアライズドビューのメンテナンスにコストをかけるべきかどうかを判断します。ワークロードの増加や変化にともない、パフォーマンス上のメリットを引き続き明確に提供するためには、これらのマテリアライズドビューを見直す必要があります。

Redshift の自動マテリアライズドビュー (AutoMV) 機能では、ユーザーが作成したマテリアライズドビューと同様のパフォーマンス上のメリットが提供されます。Amazon Redshift は、機械学習を使用してワークロードの継続的なモニタリングを行い、メリットが認められる場合には、新しいマテリアライズドビューを作成します。AutoMVは、マテリアライズドビューを作成して最新の状態に保つ際のコストと、その処理がクエリーレイテンシーに与えると思われるメリットの間でバランスを取ります。また、システムは以前に作成された AutoMV のモニタリングを行い、有益性が認められない場合はそれを削除します。

AutoMV の動作と機能は、ユーザーにより作成されたマテリアライズドビューと同じです。これらは、同じ基準と制限を使用しており、更新は自動的かつ増分的に行われます。ユーザーが作成したマテリアライズドビューと同様に、マテリアライズドビューを使用するための自動クエリ書き換え はシステムが作成した AutoMV から利点を享受できるクエリを特定します。これらのクエリは

AutoMVS を使用するように自動的に書き換えられるため、クエリのパフォーマンスが向上します。デベロッパーは AutoMV を利用するためにクエリを更新する必要はありません。

> **Note**
>
> 自動マテリアライズドビューは断続的に更新されます。AutoMV を使用するように書き換えられたクエリは、常に最新の結果を返します。データが最新でないことを Redshift が検出した場合、クエリは自動マテリアライズドビューから読み取るように書き換えられることはありません。代わりに、クエリはベーステーブルから最新のデータを選択します。

繰り返し使用されるクエリを含むワークロードで、AutoMV のメリットが活かされます。一般的ユースケースには以下が含まれます。

- ダッシュボード – ダッシュボードは、主要なビジネス指標 (KPI)、イベント、傾向、およびその他のメトリクスのビューを素早く提供するために広く使用されます。多くの場合、チャートやテーブルによる共通のレイアウトが使用されますが、フィルタリング、あるいはドリルダウンなどのディメンション選択操作では、異なるビューが表示されます。通常ダッシュボードには、異なるパラメータで繰り返し使用される共通のクエリセットが備わっています。ダッシュボードクエリは、自動化されたマテリアライズドビューから大きなメリットを得ることができます。
- レポート – レポートクエリでは、ビジネス要件とレポートの種類に基づくさまざまな頻度により、スケジューリングが行えます。さらに、自動化することもオンデマンドにすることもできます。レポートクエリに通じる一般的な特徴は、クエリが長時間実行され、リソースを大量に消費するという点です。AutoMV では、クエリの実行ごとに再計算をする必要がないため、Redshift における各クエリの実行時間とリソースの使用率が削減されます。

自動マテリアライズドビューをオフにするには、`auto_mv` パラメータグループを `false` に更新します。詳細については、Amazon Redshift クラスター管理ガイドの [Amazon Redshift パラメータグループ](#)を参照してください。

自動マテリアライズドビューの SQL スコープと考慮事項

- 自動マテリアライズドビューは、`GROUP BY` 句または SUM、COUNT、MIN、MAX、AVG のいずれかの集計関数が含まれていれば、クエリまたはサブクエリによって開始および作成できます。ただし、以下のいずれかを含めることはできません。
  - 左、右、またはフル外部結合

  - SUM、COUNT、MIN、MAX、AVG を除く集計関数 (これらの関数は、自動クエリ書き換えで動作します)。
  - DISTINCT を含むすべての集計関数
  - Window 関数
  - SELECT DISTINCT または HAVING 句
  - その他のマテリアライズドビュー

  条件を満たすクエリが自動マテリアライズドビューの作成を開始することは保証されません。システムは、ワークロードに対して期待されるメリットと、維持するリソースのコスト (更新するシステムのコストを含む) に基づいて、ビューの作成元となる候補を決定します。結果の各マテリアライズドビューは、自動クエリ書き換えによって使用可能です。
- AutoMV は、サブクエリまたはセット演算子の個々のレッグによって開始される場合がありますが、結果のマテリアライズドビューにはサブクエリまたはセット演算子は含まれません。
- AutoMV がクエリに使用されたかどうかを判断するには、EXPLAIN プランを表示し、出力で `%_auto_mv_%` を探します。詳細については、「EXPLAIN」を参照してください。
- データ共有やフェデレーションテーブルなどの外部テーブルでは、自動マテリアライズドビューはサポートされていません。

自動マテリアライズドビューの制限

自動マテリアライズドビューの操作に関する制限は次のとおりです。

- AutoMV の最大数 - 自動マテリアライズドビューの制限はクラスター内のデータベースごとに 200 です。
- ストレージスペースとキャパシティ - AutoMV は、ユーザーのワークロードに影響を与えないように、予備のバックグラウンドサイクルを使用して実行されるという点が重要な特徴です。クラスターがビジー状態であるか、ストレージ領域が不足している場合、AutoMV はアクティビティを停止します。具体的には、クラスターの総容量が 80% の場合、新しい自動マテリアライズドビューは作成されません。総容量が 90% になると、ユーザーのワークロードがパフォーマンスを低下させることなく継続できるように、これらのビューが削除されることがあります。クラスター容量の確認方法の詳細については、「STV_NODE_STORAGE_CAPACITY」を参照してください。

自動マテリアライズドビューの料金

Amazon Redshift の自動最適化機能により、自動マテリアライズドビューが作成および更新されます。このプロセスのコンピューティングリソースには料金はかかりません。自動マテリアライズドビューのストレージには、通常のストレージ料金が課金されます。詳細については、Amazon Redshift の料金を参照してください。

追加リソース

次のブログ記事では、自動マテリアライズドビューについて詳しく説明しています。それらがどのように作成、管理、削除されるかを詳しく説明しています。また、こうした意思決定を左右する基礎となるアルゴリズムについても説明しています。「Optimize your Amazon Redshift query performance with automated materialized views」」(自動マテリアライズドビューで Amazon Redshift のクエリパフォーマンスを最適化する)

このビデオでは、まずマテリアライズドビューについて説明し、マテリアライズドビューによってどのようにパフォーマンスが向上し、リソースを節約できるかを説明します。次に、プロセスフローアニメーションとライブデモンストレーションを使用して、自動マテリアライズドビューについて詳しく説明します。

マテリアライズドビューでのユーザー定義関数 (UDF) の使用

Amazon Redshift マテリアライズドビューではスカラー UDF を使用できます。これらを Python または SQL で定義し、マテリアライズドビュー定義で参照します。

マテリアライズドビューでの UDF の参照

以下の手順は、マテリアライズドビュー定義で簡単な算術比較を行う UDF を使用する方法を示しています。

1. マテリアライズドビュー定義で使用するテーブルを作成します。

```
CREATE TABLE base_table (a int, b int);
```

2. Python で、整数が比較整数より大きいかどうかを示すブール値を返すスカラーユーザー定義関数を作成します。

```
CREATE OR REPLACE FUNCTION udf_python_bool(x1 int, x2 int) RETURNS bool IMMUTABLE
AS $$
```

```
  return x1 > x2
$$ LANGUAGE plpythonu;
```

オプションで、SQL で機能的に類似した UDF を作成し、それを使用して最初の UDF と結果を比較できます。

```
CREATE OR REPLACE FUNCTION udf_sql_bool(int, int) RETURNS bool IMMUTABLE
AS $$
  select $1 > $2;
$$ LANGUAGE SQL;
```

3. 作成したテーブルから選択し、UDF を参照するマテリアライズドビューを作成します。

```
CREATE MATERIALIZED VIEW mv_python_udf AS SELECT udf_python_bool(a, b) AS a FROM
 base_table;
```

オプションで、SQL UDF を参照するマテリアライズドビューを作成できます。

```
CREATE MATERIALIZED VIEW mv_sql_udf AS SELECT udf_sql_bool(a, b) AS a FROM
 base_table;
```

4. テーブルにデータを追加し、マテリアライズドビューを更新します。

```
INSERT INTO base_table VALUES (1,2), (1,3), (4,2);
```

```
REFRESH MATERIALIZED VIEW mv_python_udf;
```

オプションで、SQL UDF を参照するマテリアライズドビューを更新できます。

```
REFRESH MATERIALIZED VIEW mv_sql_udf;
```

5. マテリアライズドビューからデータをクエリします。

```
SELECT * FROM mv_python_udf ORDER BY a;
```

このクエリの結果は以下のようになります。

```
a
-----
```

```
false
false
true
```

列 a の値 (4) は列 b の値 (2) より大きいため、これは最後の値のセットに対して `true` を返します。

6. オプションで、SQL UDF を参照するマテリアライズドビューをクエリできます。SQL 関数の結果は、Python バージョンの結果と一致します。

```
SELECT * FROM mv_sql_udf ORDER BY a;
```

このクエリの結果は以下のようになります。

```
a
-----
false
false
true
```

これは、比較する最後の値のセットについて `true` を返します。

7. CASCADE を指定した DROP ステートメントを使用して、ユーザー定義関数とそれを参照するマテリアライズドビューを削除します。

```
DROP FUNCTION udf_python_bool(int, int) CASCADE;
```

```
DROP FUNCTION udf_sql_bool(int, int) CASCADE;
```

マテリアライズドビューへのストリーミング取り込み

このトピックでは、マテリアライズドビューを使用してストリーミングデータにすばやくアクセスする方法について説明します。

ストリーミング取り込みでは、[Amazon Kinesis Data Streams](#) や [Amazon Managed Streaming for Apache Kafka](#) から、Amazon Redshift でプロビジョニングされたビューや Amazon Redshift Serverless データベースへの、低レイテンシーかつ高速のデータインジェストを行います。データは、目的に合わせて設定された Redshift マテリアライズドビューに到達します。このため、外部データへのアクセスが高速化されます。ストリーミング取り込みは、データアクセス時間を短縮し、

ストレージコストを削減します。ストリーミング取り込みは、SQL コマンドの小さなコレクションを使用して、Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループ用に設定できます。設定後は、マテリアライズドビューの更新ごとに、毎秒数百メガバイトのデータを取り込むことができます。

ストリーミングサービスから Redshift へのデータフロー

これにより、ストリーミング取り込みの仕組みやプロセスで使用されるデータベースオブジェクトを理解しやすくなります。データは、データストリームプロバイダーから Amazon Redshift でプロビジョニングされたクラスターまたは Amazon Redshift Serverless ワークグループに直接流れます。Amazon S3 バケットなどの一時的な到着エリアはありません。プロビジョニングされたクラスターやワークグループは、ストリームコンシューマーです。Redshift データベースの場合、ストリームから読み取られたデータはマテリアライズドビューに到着します。データは到着時に処理されます。例えば、SQL を使用して JSON 値を消費し、マテリアライズドビューのデータ列にマッピングできます。マテリアライズドビューを更新すると、Redshift は、ストリームに伴ってビューが最新の状態になるまで、割り当てられた Kinesis データシャードまたは Kafka パーティションのデータを消費します。

Amazon Redshift ストリーミング取り込みのユースケースでは、データの継続的な生成と、生成時点からの短期間 (低レイテンシー) での処理が伴います。これは、一般的に、ほぼリアルタイムの分析と呼ばれます。ソースには、IT デバイス、システムテレメトリデバイス、ビジー状態のウェブサイトやアプリケーションからのクリックストリームデータなどが含まれます。

パフォーマンス改善に向けたデータ解析のベストプラクティス

ストリーミング取り込みを設定する場合、受信データを解析する方法について、いくつかのオプションがあります。プラクティスには、データの到着時にビジネスロジックやフォーマットを実行することが含まれます。エラーやデータ損失を避けるため、以下のベストプラクティスをお勧めします。ベストプラクティスは、内部テストに基づくものであり、設定や解析の問題のトラブルシューティングを行う際に役立ちます。

- ストリーミングされたデータから値を抽出する — マテリアライズドビュー定義で JSON_EXTRACT_PATH_TEXT 関数を使用してストリーミング JSON を解析または細分化すると、パフォーマンスとレイテンシーに大きな影響を及ぼす可能性があります。具体的には、JSON_EXTRACT_PATH_TEXT を使用して抽出した列ごとに、着信 JSON が再解析されます。これに続けて、データ型の変換、フィルタリング、ビジネスロジックの計算が行われます。例えば、JSON データから 10 列を抽出すると、各 JSON レコードは 10 回解析されます (これには追加のロジックが含まれます)。その結果、取り込みのレイテンシーが長くなります。代

わりに、JSON_PARSE 関数を使用して JSON レコードを Redshift の SUPER データ型に変換することをお勧めします。ストリーミングされたデータがマテリアライズドビューに到着したら、PartiQL を使用して JSON データの SUPER 表現から個々の文字列を抽出します。詳細については、「半構造化データのクエリ」を参照してください。

さらに、JSON_EXTRACT_PATH_TEXT のデータサイズは最大 64 KB であることに注意してください。JSON レコードのサイズが 64 KB を超えると、JSON_EXTRACT_PATH_TEXT での処理エラーになります。

- Amazon Kinesis Data Streams ストリームまたは Amazon MSK トピックを複数のマテリアライズドビューにマッピングする — 単一のストリームやトピックからデータを取り込むために、複数のマテリアライズドビューを作成することはお勧めしません。その理由として、各マテリアライズドビューは Kafka トピックの Kinesis Data Streams ストリームやパーティション内のシャードごとにコンシューマーを作成するためです。これにより、スロットリングや、ストリームまたはトピックのスループット超過が生じる場合があります。また、同じデータを複数回取り込むことになるため、コストが高くなる可能性もあります。ストリーミング取り込みを設定する場合、ストリームやトピックごとに 1 つのマテリアライズドビューを作成することをお勧めします。

  ユースケースで 1 つの KDS ストリームや MSK トピックから複数のマテリアライズドビューにデータを取り込む必要がある場合は、事前に AWS Big Data Blog の「Amazon MSK で Amazon Redshift ストリーミング取り込みを使用して、ほぼリアルタイムの分析を実装するためのベストプラクティス」を参照してください。

ストリーミング取り込みの動作とデータタイプ

次の表では、データタイプ別の技術的な動作の詳細とサイズ制限について説明します。ストリーミング取り込み用にマテリアライズドビューを設定する前に、以下を理解しておくことをお勧めします。

機能または動作	説明
Kafka トピックの長さ制限	Kafka トピックの名前は 128 文字 (引用符は含まない) を超えることはできません。詳細については、「名前と識別子」を参照してください。
マテリアライズドビューでの増分の更新と JOIN	マテリアライズドビューは、増分的な保守が可能である必要があります。Kinesis や Amazon MSK では、24 時間または 7 日前のストリームやトピックの履歴は保持されないため、完全な再計算は不可能です。Kinesis または Amazon MSK では、よ

機能または動作	説明
	り長いデータ保持期間を設定することができます。ただし、これによりメンテナンスとコストが増える可能性があります。また、現在、Kinesis Streams や Amazon MSK トピックで作成されたマテリアライズドビューでは、JOIN の使用はサポートされていません。ストリームやトピックでマテリアライズドビューを作成した後、別のマテリアライズドビューを作成して、ストリーミングのマテリアライズドビューと他のマテリアライズドビュー、テーブル、またはビューとの結合のために使用できます。 詳細については、「REFRESH MATERIALIZED VIEW」を参照してください。
レコード解析	Amazon Redshift のストリーミング取り込みでは、Kinesis プロデューサーライブラリ (KPL の重要なコンセプト) によって集計されたレコードの解析をサポートしていません。集計されたレコードは取り込まれますが、バイナリプロトコルのバッファデータとして格納されます。(詳細については「Protocol buffers」(プロトコルバッファ) を参照してください。) Kinesis へのデータのプッシュ方法によっては、この機能の無効化が必要となる場合があります。
解凍	VARBYTE は解凍をサポートしていません。このため、圧縮データを含むレコードを Redshift でクエリすることはできません。データは、Kinesis ストリームや Amazon MSK トピックに追加する前に解凍してください。

機能または動作	説明
レコードの最大サイズ	Amazon Redshift が Kinesis または Amazon MSK から取り込むことができるレコードの最大サイズは 16,777,216 バイト (16 MiB) で、これは Amazon Redshift の VARBYTE データ型でサポートされる最大サイズです。デフォルトでは、Kinesis データストリームまたは Amazon MSK トピックで作成された Amazon Redshift ストリーミングマテリアライズドビューは、データ列のサイズをそれぞれ 1,048,576 バイト (1 MiB) と 16,777,216 バイト (16 MiB) に設定します。 Note 1 MiB は、現在 Kinesis データストリームに取り込むことができるレコードの最大サイズです。Kinesis のサイズ制限の詳細については、「Amazon Kinesis Data Streams 開発者ガイド」の「Quotas and limits」を参照してください。
エラーレコード	データが最大サイズを超えているためにレコードを Redshift に取り込めない場合、そのレコードはスキップされます。この場合でも、マテリアライズドビューの更新は成功し、各エラーレコードのセグメントが SYS_STREAM_SCAN_ERRORS システムテーブルに書き込まれます。計算のエラーやタイプ変換によるエラーなど、ビジネスロジックに起因するエラーはスキップされません。ロジックは、マテリアライズドビュー定義に追加する前に、入念にテストしてください。
Amazon MSK マルチ VPC プライベート接続	Amazon MSK マルチ VPC プライベート接続は、現在 Redshift ストリーミングの取り込みではサポートされていません。代わりに、VPC ピアリングを使用して VPC に接続するか、AWS Transit Gateway を使用しセントラルハブを介して VPC およびオンプレミスネットワークに接続することができます。これらのいずれかにより、Redshift は Amazon MSK クラスターまたは別の VPC にある Amazon MSK サーバーレスと通信できるようになります。

機能または動作	説明
自動更新の使用とアクティブ化	マテリアライズドビューに対する自動更新クエリは、他のユーザーワークロードと同様に扱われます。自動更新は、ストリームが到達するとデータをロードします。 ストリーミング取り込み用に作成されたマテリアライズドビューでは、自動更新を明示的にオンにできます。これを行うには、マテリアライズドビュー定義で `AUTO REFRESH` を指定します。手動更新がデフォルトです。ストリーミング取り込み用の既存のマテリアライズドビューに自動更新を指定するには、`ALTER MATERIALIZED VIEW` を実行してオンにします。詳細については、「CREATE MATERIALIZED VIEW」または「ALTER MATERIALIZED VIEW」を参照してください。
ストリーミング取り込みと Amazon Redshift Serverless	プロビジョニングされたクラスターで Amazon Redshift ストリーミング取り込みに適用されるセットアップと設定の手順は、Amazon Redshift Serverless でのストリーミング取り込みにも適用されます。自動更新やその他のワークロードでストリーミング取り込みをサポートするには、必要なレベルの RPU を指定することが重要です。詳細については、「Amazon Redshift Serverless の料金」を参照してください。
Amazon MSK クラスターとは異なるアベイラビリティーゾーンの Amazon Redshift ノード	ストリーミング取り込みを設定すると、Amazon MSK のラック認識が有効になっている場合、Amazon Redshift は同じアベイラビリティーゾーンの Amazon MSK クラスターへの接続を試みます。すべてのノードが Amazon Redshift クラスターとは異なるアベイラビリティーゾーンにある場合、アベイラビリティーゾーン間のデータ転送コストが発生する可能性があります。これを回避するには、Redshift のプロビジョニングされたクラスターまたはワークグループと同じ AZ に少なくとも 1 つの Amazon MSK ブローカークラスターノードを保持します。
更新の開始場所	マテリアライズドビューを作成すると、最初の更新は Kinesis ストリームの `TRIM_HORIZON` または Amazon MSK トピックのオフセット 0 から始まります。

機能または動作	説明
データ形式	サポートされるデータ形式は、VARBYTE からの変換が可能なデータ形式に限られます。詳細については、VARBYTE 型およびVARBYTE 演算子を参照してください。
テーブルへのレコードの追加	既存のソースマテリアライズドビューからターゲットテーブルに行を追加するには、ALTER TABLE APPEND を実行できます。これは、マテリアライズドビューがストリーミング取り込み用に設定されている場合にのみ機能します。詳細については、「ALTER TABLE APPEND」を参照してください。
TRUNCATE または DELETE の実行	ストリーミング取り込みに使用するマテリアライズドビューからレコードを削除するには、以下を使用します。 • TRUNCATE — ストリーミング取り込み用に設定されたマテリアライズドビューからすべての行を削除します。テーブルスキャンは行われません。詳細については、「TRUNCATE」を参照してください。 • DELETE — ストリーミング取り込み用に設定されたマテリアライズドビューからすべての行を削除します。詳細については、「DELETE」を参照してください。

Amazon Kinesis Data Streams からストリーミング取り込みを開始する方法

このトピックでは、マテリアライズドビューを使用して Kinesis Data Streams からストリーミングデータを使用する方法について説明します。

Amazon Redshift ストリーミング取り込みを設定する際は、外部スキーマを作成しストリーミングのデータソースにマッピングした上で、その外部スキーマを参照するマテリアライズドビューを作成します。Amazon Redshift のストリーミング取り込みでは、データソースとして Kinesis Data Streams をサポートします。そのため、ストリーミング取り込みを設定する前に、Kinesis Data Streams ソースを使用可能な状態にしておく必要があります。ソースがない場合は、Kinesis ドキュメント「Amazon Kinesis Data Streams の開始方法」の手順に従います。または「AWS マネジメントコンソールを介してのストリームの作成」の手順に従って、コンソールからソースを作成します。

Amazon Redshift のストリーミング取り込みではマテリアライズドビューが使用されます。マテリアライズドビューは、REFRESH実行時にストリームにより直接更新されます。マテリアライズドビューは、ストリームのデータソースにマッピングされています。マテリアライズドビューの定義を行う際には、ストリームデータをフィルタリングしたり集計したりできます。ストリーミング取り込みのマテリアライズドビュー (基盤のマテリアライズドビュー) は 1 つのストリームのみを参照します。ただし、追加のマテリアライズドビューを作成して、それを基盤または別のマテリアライズドビュー、あるいはテーブルと結合させることができます。

> **Note**
>
> ストリーミング取り込みと Amazon Redshift Serverless - このトピックの設定手順は Amazon Redshift クラスターおよび Amazon Redshift Serverless の両方に適用されます。詳細については、「ストリーミング取り込みの動作とデータタイプ」を参照してください。

Kinesis Data Streams のストリームが利用可能な場合の最初のステップは、CREATE EXTERNAL SCHEMA を使用して Amazon Redshift 内にスキーマを定義することです。その上で、Kinesis Data Streams リソースを参照します。その後、ストリーム内のデータにアクセスするために、マテリアライズドビュー内で STREAM を定義します。ストリームレコードは半構造的な SUPER 形式で保存できます。あるいは、Redshift データ型に変換されたデータを出力するスキーマを定義することができます。マテリアライズドビューをクエリすると、返されるレコードにはその時点のストリームが反映されます。

1. Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループがロールを引き受けることを許可する信頼ポリシーを持つ IAM ロールを作成します。IAM ロール向けに信頼ポリシーを設定する方法については、「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを許可する」を参照してください。作成されたロールには次の IAM ポリシーが設定されており、これにより、Amazon Kinesis のデータストリームとの通信に関するアクセス許可が提供されます。

   Kinesis データストリームからの暗号化されていないストリームの IAM ポリシー

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ReadStream",
               "Effect": "Allow",
               "Action": [
   ```

```
                "kinesis:DescribeStreamSummary",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
        },
        {
            "Sid": "ListStream",
            "Effect": "Allow",
            "Action": "kinesis:ListStreams",
            "Resource": "*"
        }
    ]
}
```

Kinesis データストリームからの暗号化されたストリームの IAM ポリシー

```
{
 "Version": "2012-10-17",
 "Statement": [{
   "Sid": "ReadStream",
   "Effect": "Allow",
   "Action": [
    "kinesis:DescribeStreamSummary",
    "kinesis:GetShardIterator",
    "kinesis:GetRecords",
    "kinesis:ListShards",
    "kinesis:DescribeStream"
   ],
   "Resource": "arn:aws:kinesis:*:0123456789:stream/*"
  },
  {
   "Sid": "DecryptStream",
   "Effect": "Allow",
   "Action": [
    "kms:Decrypt"
   ],
   "Resource": "arn:aws:kms:us-
east-1:0123456789:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
        {
```

```
    "Sid": "ListStream",
    "Effect": "Allow",
    "Action": "kinesis:ListStreams",
    "Resource": "*"
   }
  ]
}
```

2. VPC をチェックして、Amazon Redshift クラスターまたは Amazon Redshift Serverless に NAT ゲートウェイまたはインターネットゲートウェイを使用して、インターネット経由で Kinesis Data Streams エンドポイントに到達するルートがあることを確認します。Redshift と Kinesis Data Streams 間のトラフィックを AWS ネットワーク内に残しておきたい場合は、Kinesis インターフェイス VPC エンドポイントの使用を検討してください。詳細については、「Using Amazon Kinesis Data Streams Kinesis Data Streams with Interface VPC Endpoints」(Amazon Kinesis Data Streamsでのインターフェイス VPC エンドポイントの使用) を参照してください。
3. Amazon Redshift で、Kinesis からのデータをスキーマにマッピングするために、外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
IAM_ROLE { default | 'iam-role-arn' };
```

Kinesis Data Streams のストリーミング取り込みには認証タイプは必要ありません。`CREATE EXTERNAL SCHEMA` ステートメントで定義されている IAM ロールを使用して Kinesis Data Streams リクエストを行います。

オプション: REGION キーワードを使用して、Amazon Kinesis Data Streams または Amazon MSK ストリームが存在するリージョンを指定します。

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
REGION 'us-west-2'
IAM_ROLE { default | 'iam-role-arn' };
```

このサンプルでは、リージョンがソースストリームの場所を指定します。IAM_ROLE はサンプルです。

4. ストリームデータを利用するためのマテリアライズドビューを作成します。次のようなステートメントの場合、レコードを解析できないと、エラーが発生します。エラーレコードをスキップしない場合は、このようなコマンドを使用します。

```
CREATE MATERIALIZED VIEW my_view AUTO REFRESH YES AS
SELECT *
FROM kds.my_stream_name;
```

Kinesis のストリームの名前では大文字と小文字が区別され、その両方を使用することができます。名前が大文字のストリームからインジェストするには、データベースレベルで設定 `enable_case_sensitive_identifier` を `true` に指定できます。詳細については、「名前と識別子」ならびに「enable_case_sensitive_identifier」を参照してください。

自動更新をオンにするには、`AUTO REFRESH YES` を使用してください。デフォルトの動作は手動更新です。CAN_JSON_PARSE を使用する場合、解析できないレコードはスキップされる可能性があることに注意してください。

メタデータ列には次の列が含まれます。

メタデータ列	データ型	説明
approximate_arrival_timestamp	タイムゾーンなしのタイムスタンプ	レコードが Kinesis Streams に挿入された、おおよその時間
partition_key	varCHAR(256)	レコードをシャードに割り当てるために Kinesis によって使用されるキー
shard_id	char(20)	レコードを取得したストリーム内のシャードの一意識別子
sequence_number	varCHAR(128)	Kinesis シャードのレコードの一意識別子
refresh_time	タイムゾーンなしのタイムスタンプ	更新の開始時間
kinesis_data	varbyte	Kinesis ストリームからのレコード

マテリアライズドビュー定義内のビジネスロジックによっては、ビジネスロジックのエラーに伴ってストリーミング取り込みがブロックされることに注意してください。場合によっては、マテリアライズドビューを削除して再作成しなければならないことがあります。これを回避するには、ロジックをできるだけシンプルにし、ビジネスロジックのチェックのほとんどをインジェスト後のデータに実行することをお勧めします。

5. ビューを更新します。これにより、Redshift を呼び出してストリームから読み取り、データをマテリアライズドビューにロードします。

```
REFRESH MATERIALIZED VIEW my_view;
```

6. マテリアライズドビュー内でデータをクエリします。

```
select * from my_view;
```

Amazon Managed Streaming for Apache Kafka (Amazon MSK) からのストリーミング取り込みを開始する

このトピックでは、マテリアライズドビューを使用して Amazon MSK からストリーミングデータを使用する方法について説明します。

Amazon Redshift ストリーミング取り込みの目的は、ストリーミングサービスから Amazon Redshift または Amazon Redshift Serverless にストリームデータを直接取り込むプロセスを簡略化することです。これは Amazon MSK Provisioned と Amazon MSK Severless、および Kinesis データストリームで動作します。Amazon Redshift ストリーミング取り込みにより、ストリームデータを Redshift に取り込む前に、Kinesis Data Streams ストリームまたは Amazon MSK トピックを Amazon S3 にステージングする必要がなくなります。

技術的なレベルでは、Amazon Kinesis Data Streams と Amazon MSK の両方からのストリーミング取り込みは、Amazon Redshift のマテリアライズドビューへのストリームまたはトピックデータの低レイテンシーで高速な取り込みを提供します。セットアップ後、マテリアライズドビューの更新を使用すると、大量のデータを取り込むことができます。

以下の手順を実行して、Amazon MSK 用の Amazon Redshift ストリーミング取り込みをセットアップします。

1. ストリーミングデータソースにマッピングする外部スキーマを作成します。

2. 外部スキーマを参照するマテリアライズドビューを作成します。

Amazon Redshift ストリーミング取り込みを設定する前に、Amazon MSK ソースが利用可能になっている必要があります。ソースがない場合は、「Amazon MSK の使用を開始する」の指示に従ってください。

> **Note**
>
> ストリーミング取り込みと Amazon Redshift Serverless - このトピックの設定手順は Amazon Redshift クラスターおよび Amazon Redshift Serverless の両方に適用されます。詳細については、「ストリーミング取り込みの動作とデータタイプ」を参照してください。

IAM アクセス許可の設定と Kafka からのストリーミング取り込みの実行

Amazon MSK クラスターが利用可能な場合の最初のステップは、`CREATE EXTERNAL SCHEMA` を使用して Redshift でスキーマを定義し、Amazon MSK クラスターをデータソースとして参照することです。その後、トピック内のデータにアクセスするために、マテリアライズドビュー内で `STREAM` を定義します。トピックからのレコードは、デフォルトの Amazon Redshift VARBYTE データ型を使用して保存できます。または、半構造的な `SUPER` 形式にデータを変換するスキーマを定義することができます。マテリアライズドビューをクエリすると、返されるレコードにはその時点のトピックが反映されます。

1. AUTHENTICATION NONE を使用して MSK に接続する場合、IAM ロールは必要ありません。ただし、AUTHENTICATION IAM または MTLS を使用して Amazon MSK クラスターで認証する場合、Amazon Redshift クラスターまたは Amazon Redshift Serverless 名前空間には、適切なアクセス許可を持つ IAM ロールがアタッチされている必要があります。Amazon Redshift クラスターまたは Amazon Redshift Serverless 名前空間がロールを引き受けることを許可する信頼ポリシーを持つ IAM ロールを作成します。ロールを作成したら、IAM または MTLS をサポートするために、次のいずれかのアクセス許可を追加します。mTLS 認証では、Amazon Redshift が使用する証明書を AWS Certificate Manager または AWS Secrets Manager に保存できるため、証明書が保存されている場所に一致するポリシーを選択する必要があります。プロビジョニングされた Amazon Redshift クラスターまたは Redshift Serverless 名前空間にロールをアタッチします。IAM ロール向けに信頼ポリシーを設定する方法については、「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを許可する」を参照してください。

   AUTHENTICATION IAM:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MSKIAMpolicy",
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:Connect"
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:cluster/MyTestCluster/*",
                "arn:aws:kafka:*:0123456789:topic/MyTestCluster/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeGroup"
            ],
            "Resource": [
                "arn:aws:kafka:*:0123456789:group/MyTestCluster/*"
            ]
        }
    ]
}
```

AUTHENTICATION MTLS: AWS Certificate Manager に保存された証明書を使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MSKmTLSACMpolicy",
            "Effect": "Allow",
            "Action": [
                "acm:ExportCertificate"
            ],
            "Resource": [
                "arn:aws:acm:us-east-1:444455556666:certificate/certificate_ID"
```

```
            ]
        }
    ]
}
```

AUTHENTICATION MTLS: AWS Secrets Manager に保存された証明書を使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MSKmTLSSecretsManagerpolicy",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-east-1:444455556666:secret:secret_ID"
            ]
        }
    ]
}
```

2. VPC を確認して、Amazon Redshift クラスターまたは Amazon Redshift Serverless に Amazon MSK クラスターに到達するためのルートがあることを確認します。Amazon MSK クラスターのインバウンドセキュリティグループルールでは、Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループのセキュリティグループを許可する必要があります。指定するポートは、Amazon MSK クラスターで設定されている認証方法によって異なります。詳細については、「ポート情報」と「AWS 内かつ VPC 外からのアクセス」を参照してください。

   次の表は、Amazon MSK からのストリーミング取り込みに設定する、補足の設定オプションを示しています。

Amazon Redshift の設定	Amazon MSK 設定	Redshift と Amazon MSK の間で開くポート
AUTHENTICATION NONE	TLS トランスポート無効	9092
AUTHENTICATION NONE	TLS トランスポート有効	9094

Amazon Redshift の設定	Amazon MSK 設定	Redshift と Amazon MSK の間で開くポート
AUTHENTICATION IAM	IAM	9098/9198
AUTHENTICATION MTLS	TLS トランスポート有効	9094

Amazon Redshift 認証は、CREATE EXTERNAL SCHEMA ステートメントで設定されます。

> **Note**
>
> Amazon MSK クラスターで相互 Transport Layer Security (mTLS) 認証が有効になっている場合は、AUTHENTICATION NONE を使用するよう Amazon Redshift を設定すると、認証されていないアクセスにはポート 9094 を使用するように指示されます。ただし、ポートは mTLS 認証で使用されているため、これは失敗します。このため、mTLS を使用する場合は AUTHENTICATION mtls に切り替えることをお勧めします。

3. Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループで拡張 VPC ルーティングを有効にします。詳細については、「拡張された VPC ルーティングの有効化」を参照してください。
4. Amazon Redshift で、Amazon MSK クラスターにマッピングする外部スキーマを作成します。構文は次のとおりです。

```
CREATE EXTERNAL SCHEMA MySchema
FROM MSK
[ IAM_ROLE [ default | 'iam-role-arn' ] ]
AUTHENTICATION [ none | iam | mtls ]
[AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret-arn' ];
```

`FROM` 句の `MSK` は、スキーマが Managed Kafka Services のデータをマッピングしていることを示します。

`AUTHENTICATION` は、Amazon MSK でのストリーミング取り込みの認証タイプを示します。3 つのタイプを使用できます。

- none — 必要な認証がないことを指定します。これは、MSK での認証されていないアクセスに対応します。

- iam — IAM 認証を指定します。これを選択するときは、IAM ロールに IAM 認証のアクセス許可があることを確認します。必要な IAM ポリシーの設定の詳細については、「IAM アクセス許可の設定と Kafka からのストリーミング取り込みの実行」を参照してください。
- mtls – クライアントとサーバー間の認証を行うことで、相互 Transport Layer Security が安全な通信を提供することを指定します。この場合、クライアントは Redshift で、サーバーは Amazon MSK です。mTLS によるストリーミング取り込みの設定の詳細については、「Amazon MSK からの Redshift ストリーミング取り込みにおける mTLS による認証」を参照してください。

ユーザー名とパスワードを使用した Amazon MSK 認証は、ストリーミング取り込みではサポートされていないことに注意してください。

AUTHENTICATION_ARN は、暗号化された接続を確立するために使用する ACM 相互 Transport Layer Security (mTLS) 証明書の ARN を指定します。

SECRET_ARN は、Amazon Redshift が mTLS に使用する証明書を含む AWS Secrets Manager シークレットの ARN を指定します。

以下のサンプルは、外部スキーマを作成するときに Amazon MSK クラスターに対してブローカー URI を設定する方法を示しています。

```
CREATE EXTERNAL SCHEMA my_schema
FROM MSK
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION IAM
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9098,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9098'
```

認証を使用しない場合:

```
CREATE EXTERNAL SCHEMA my_schema
FROM MSK
AUTHENTICATION none
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9092,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9092'
```

mTLS の使用

```
CREATE EXTERNAL SCHEMA my_schema
FROM MSK
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION MTLS
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9094,b-
 2.myTestCluster.123z8u.c2.kafka.us-west-1.amazonaws.com:9094'
AUTHENTICATION_ARN 'acm-certificate-arn' | [ SECRET_ARN 'ssm-secret-arn' ];
```

外部スキーマを作成する方法の詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

5. トピックからのデータを利用するためのマテリアライズドビューを作成します。次の例のような SQL コマンドを使用します。

```
CREATE MATERIALIZED VIEW MyView AUTO REFRESH YES AS
SELECT *
FROM MySchema."mytopic";
```

Kafka トピックの名前では大文字と小文字が区別され、その両方を使用することができます。名前が大文字のトピックから取り込むには、セッションまたはデータベースレベルで設定 `enable_case_sensitive_identifier` を `true` に指定できます。詳細については、「名前と識別子」ならびに「enable_case_sensitive_identifier」を参照してください。

自動更新をオンにするには、`AUTO REFRESH YES` を使用してください。デフォルトの動作は手動更新です。

メタデータ列には次の列が含まれます。

メタデータ列	データ型	説明
kafka_partition	bigint	Kafka トピックのレコードのパーティション ID
kafka_offset	bigint	指定されたパーティションの Kafka トピック内のレコードのオフセット
kafka_timestamp_type	char(1)	Kafka レコードで使用されるタイムスタンプのタイプ:

メタデータ列	データ型	説明
		• C — クライアント側でのレコード作成時間 (CREATE_TIME) • L — Kafka サーバー側のレコード追加時間 (LOG_APPEND_TIME) • U — レコードの作成時刻が利用不可 (NO_TIMESTAMP_TYPE)
kafka_timestamp	タイムゾーンなしのタイムスタンプ	レコードの timestamp 値
kafka_key	varbyte	Kafka レコードのキー
kafka_value	varbyte	Kafka から受け取ったレコード
kafka_headers	super	Kafka から受け取ったレコードのヘッダー
refresh_time	タイムゾーンなしのタイムスタンプ	更新の開始時間

マテリアライズドビュー定義内のビジネスロジックによってビジネスロジックエラーが発生する場合、一部のケースではストリーミング取り込みが失敗する可能性があることに注意してください。場合によっては、マテリアライズドビューを削除して再作成しなければならないことがあります。これを回避するには、ビジネスロジックをシンプルにし、インジェスト後のデータに追加のロジックを実行することをお勧めします。

6. ビューを更新します。これにより、Amazon Redshift を呼び出してトピックから読み取り、データがマテリアライズドビューにロードされます。

```
REFRESH MATERIALIZED VIEW MyView;
```

7. マテリアライズドビュー内でデータをクエリします。

```
select * from MyView;
```

マテリアライズドビューは、REFRESH の実行時にトピックから直接更新されます。Kafka トピックのデータソースにマッピングするマテリアライズドビューを作成します。マテリアライズドビューの定義を行う際には、データをフィルタリングしたり集計したりできます。ストリーミング取り込みのマテリアライズドビュー (基盤のマテリアライズドビュー) は 1 つの Kafka トピックのみを参照します。ただし、追加のマテリアライズドビューを作成して基盤のマテリアライズドビューと結合したり、は別のマテリアライズドビューやテーブルと結合したりできます。

ストリーミング取り込みの制限の詳細については、「ストリーミング取り込みの動作とデータタイプ」を参照してください。

Amazon MSK からの Redshift ストリーミング取り込みにおける mTLS による認証

相互 Transport Layer Security (mTLS) は、情報の送信先のクライアントをサーバーが認証し、クライアントがサーバーを認証する手段を提供します。mTLS を使用する利点は、業界特有のいくつかのアプリケーションのさまざまなユースケースに対して、信頼できる認証を提供することです。これには、金融、小売、政府、医療業界のユースケースが含まれます。Redshift にストリーミングを取り込む場合、認証はサーバー (この場合は Amazon MSK) と、プロビジョニングされた Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループとの間で行われます。

このトピックでは、Redshift クライアントと Amazon MSK サーバー間で mTLS を使用して認証する外部スキーマを作成する方法を示す手順と SQL コマンドの例を示します。このトピックで説明する手順は、Amazon MSK からのストリーミング取り込みを設定するための一連の手順を補完します。詳細については、「Amazon Managed Streaming for Apache Kafka (Amazon MSK) からのストリーミング取り込みを開始する」を参照してください。

ストリーミング取り込みに mTLS を使用するための前提条件

このセクションでは、AWS Certificate Manager または Amazon SageMaker でのストリーミング取り込みに mTLS を使用するための前提となる手順について説明します。

1. 事前の手順として、プライベート CA (PCA) を所有または作成する必要があります。PCA を使用すると、他の機能の中でも特に、安全な通信チャネルを介した安全な通信を可能にする証明書を発行できます。 AWS Private Certificate Authority(Private CA) は、この機能を実行する利用可能なサービスです。詳細については、「AWS Private Certificate Authority ユーザーガイド」の「Creating a private CA」を参照してください。後の手順では、証明書を発行して Amazon MSK クラスターにアタッチし、Redshift への暗号化通信を有効にします。

2. mtls クライアント認証をサポートする Amazon MSK クラスターを作成します。Amazon MSK クラスターの設定の詳細については、「Amazon Managed Streaming for Apache Kafka 開発者ガイド」の「To create a cluster that supports client authentication」を参照してください。
3. Amazon MSK クラスターのセキュリティ設定を編集し、AWS Certificate Manager (ACM) を使用して TLS クライアント認証を有効にして、以前に作成した AWS Private CA (PCA) を選択します。詳細については、「Amazon Managed Streaming for Apache Kafka 開発者ガイド」の「Updating security settings of a cluster」を参照してください。

AWS Certificate Manager でのストリーミング取り込みに mTLS を使用する

次の手順は、証明書のストレージと管理に AWS Certificate Manager (ACM) を活用して Redshift ストリーミングの取り込みに mTLS を設定する方法を示します。

1. ACM からプライベート証明書をリクエストします。これを実行するには、前提条件セクションで作成した PCA を認証機関として選択します。ACM は、署名付き証明書とアタッチされたプライベートキーを安全な通信のために保存します。詳細については、「AWS Certificate Manager ユーザーガイド」の「Issuing and managing certificates」を参照してください。
2. Redshift クラスターまたは Amazon Redshift Serverless ワークグループの管理に使用する IAM ロールには、証明書をエクスポートするアクセス許可をアタッチします。これは acm:ExportCertificate です。Amazon MSK でのストリーミング取り込みに必要な IAM リソースの設定の詳細については、「IAM アクセス許可の設定と Kafka からのストリーミング取り込みの実行」を参照してください。次の手順で同じ IAM ロールを指定して、外部スキーマを作成します。

> **Note**
>
> AWS Certificate Manager へのリクエストには、VPC にインターネットゲートウェイ (IGW) または NAT ゲートウェイ (NGW) が必要です。VPC に IGW または NGW のいずれも存在しない場合は、以下を実行します。
>
> - ACM の代わりに Secrets Manager を使用して証明書を保存します。
> - Secrets Manager VPC エンドポイントを VPC にアタッチします。
>
> ストリーミング取り込みに Secrets Manager を mTLS で使用する方法については、次の AWS Secrets Manager でのストリーミング取り込みに mTLS を使用する を参照してください。

3. Amazon MSK クラスター用のブートストラップブローカーを取得します。ブートストラップブローカー URI の取得の詳細については、「Amazon Managed Streaming for Apache Kafka 開発者ガイド」の「Getting the bootstrap brokers for an Amazon MSK cluster」を参照してください。

4. 次のサンプルのような SQL コマンドを実行して、mtls を使用して Amazon MSK クラスターから Redshift 外部スキーマにストリームをマッピングする外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA my_schema
FROM MSK
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION mtls
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094'
AUTHENTICATION_ARN 'arn:aws:acm:Region:444455556666:certificate/certificate_ID';
```

重要なパラメータ:

- IAM_ROLE – クラスターに関連付けられたストリーミング取り込み用の IAM ロール。
- URI – Amazon MSK クラスターのブートストラップブローカー URI。ポート 9094 は、TLS 暗号化のブローカーとの通信用に指定されていることに注意してください。
- AUTHENTICATION_ARN – ACM 証明書の ARN。ARN は、発行された証明書を選択する際に ACM コンソールで利用できます。

これらの設定手順を実行したら、サンプルで定義されたスキーマを参照する Redshift マテリアライズドビューを作成し、REFRESH MATERIALIZED VIEW を使用してデータをストリーミングできます。詳細については、Amazon MSK からのストリーミングの開始 (Amazon Managed Streaming for Apache Kafka からのストリーミング取り込みを開始する) を参照してください。

AWS Secrets Manager でのストリーミング取り込みに mTLS を使用する

ACM の証明書を参照しない場合は、AWS Secrets Manager を使用して証明書管理を行うことで、Redshift ストリーミング取り込みの mTLS を設定できます。以下の手順では、その設定方法について説明します。

1. 任意のツールを使用して、証明書署名リクエストとプライベートキーを作成します。次に、署名リクエストを使用して、Amazon MSK クラスターの証明書の生成に使用したのと同じ AWS Private CA (PCA) を使用して、署名付き証明書を生成します。証明書の発行の詳細について

は、「AWS Private Certificate Authority API リファレンス」の「IssueCertificate」を参照してください。

2. AWS Private Certificate Authority を使用して証明書を抽出します。詳細については、「AWS Private Certificate Authority ユーザーガイド」の「Retrieve a private certificate」を参照してください。

3. AWS Secrets Manager の前の手順で生成した証明書とプライベートキーを保存します。`Other type of secret` を選択して、プレーンテキスト形式を使用します。key-value ペアは、次の例のように `{"certificate":"<cert value>","privateKey":"<pkey value>"}` 形式である必要があります。AWS Secrets Manager でのシークレットの作成と管理の詳細については、「AWS Secrets Manager ユーザーガイド」の「Create and manage secrets with AWS Secrets Manager」を参照してください。

```
{"certificate":"-----BEGIN CERTIFICATE-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
H4hAX8/eE96qCcjkpfT84EdvHzp6fC+/WwM0oXlwUEWlvfMCXNaG5D8SqRq3qA==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
-----END CERTIFICATE-----",
"privateKey":"-----BEGIN PRIVATE KEY-----
klhdslkfjahksgdfkgioeuyihbflahabhbdslv6akybeoiwv1hoaiusdhbahsbdi
7OD4m1dBEs3Fj++hDMH9rYRp99RqtCOf0EWOUe139KOilOsW+cyhAoc9Ci2+Jo/k
17u2N1iGILMQEZuCRtnJOkFYkw==
-----END PRIVATE KEY-----"}
```

4. Redshift で、外部スキーマを作成する SQL コマンドを実行します。AUTHENTICATION タイプに `mtls` を使用します。また、Amazon MSK クラスターの URI と AWS Secrets Manager のシークレット ARN も指定します。

```
CREATE EXTERNAL SCHEMA my_schema
FROM MSK
IAM_ROLE 'arn:aws:iam::012345678901:role/my_role'
AUTHENTICATION mtls
URI 'b-1.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094,b-2.myTestCluster.123z8u.c2.kafka.us-
west-1.amazonaws.com:9094'
SECRET_ARN 'arn:aws:secretsmanager:us-east-1:012345678910:secret:myMTLSSecret';
```

重要なパラメータ:

- IAM_ROLE – クラスターに関連付けられたストリーミング取り込み用の IAM ロール。
- URI – Amazon MSK クラスターのブートストラップブローカー URI。ポート 9094 は、TLS 暗号化のブローカーとの通信用に指定されていることに注意してください。
- SECRET_ARN – Secrets Manager からのシークレットの ARN。mTLS に使用する証明書が含まれています。

既存の外部スキーマの mTLS 認証を有効にする

ストリーミングの取り込みに使用する既存の外部スキーマがあり、認証に相互 TLS を実装する場合は、次のようなコマンドを実行できます。このコマンドは、mTLS 認証と ACM の ACM 証明書 ARN を指定します。

```
ALTER EXTERNAL SCHEMA schema_name
AUTHENTICATION mtls
AUTHENTICATION_ARN 'arn:aws:acm:Region:444455556666:certificate/certificate_ID';
```

または、AWS Secrets Manager のシークレット ARN を参照して、mTLS 認証を指定できます。

```
ALTER EXTERNAL SCHEMA schema_name
AUTHENTICATION mtls
SECRET_ARN 'arn:aws:secretsmanager:us-east-1:012345678910:secret:myMTLSSecret';
```

Kinesis を使用したストリーミングデータの取り込み

この手順では、ev_station_data という名前の Kinesis のストリームからデータを取り込む方法を説明します。このデータには、さまざまな EV 充電ステーションからの消費データが JSON 形式で含まれています。スキーマの定義は詳細です。この例では、データを未加工のまま JSON として保存する方法と、取り込み時に JSON データ を Amazon Redshift データ型に変換する方法を示します。

プロデューサーのセットアップ

1. Amazon Kinesis Data Streams を使用して、手順に従い `ev_station_data` という名前のストリームを作成します。[Capacity mode] (キャパシティーモード) で、[On-demand] (オンデマンド) を選択します。詳細については、「AWS マネジメントコンソールを介してのストリームの作成」を参照してください。

2. Amazon Kinesis Data Generator は、ストリームで使用するテストデータを生成するのに役立ちます。このツールでは、使用を開始するための手順がガイドされ、データを生成するためには、次のようなデータテンプレートが使用できます。

```
{

   "_id" : "{{random.uuid}}",
   "clusterID": "{{random.number(
        {   "min":1,
            "max":50
        }
    )}}",
    "connectionTime": "{{date.now("YYYY-MM-DD HH:mm:ss")}}",
    "kWhDelivered": "{{commerce.price}}",
    "stationID": "{{random.number(
        {   "min":1,
            "max":467
        }
    )}}",
      "spaceID": "{{random.word}}-{{random.number(
        {   "min":1,
            "max":20
        }
    )}}",

   "timezone": "America/Los_Angeles",
   "userID": "{{random.number(
        {   "min":1000,
            "max":500000
        }
    )}}"
}
```

ストリームデータ内の各 JSON オブジェクトには、以下のプロパティがあります。

```
{
    "_id": "12084f2f-fc41-41fb-a218-8cc1ac6146eb",
    "clusterID": "49",
    "connectionTime": "2022-01-31 13:17:15",
    "kWhDelivered": "74.00",
    "stationID": "421",
    "spaceID": "technologies-2",
```

```
    "timezone": "America/Los_Angeles",
    "userID": "482329"
}
```

Amazon Redshift のセットアップ

次の手順では、データを取り込むための、マテリアライズドビューの設定方法を示します。

1. データを Kinesis から Redshift オブジェクトにマッピングするために、外部スキーマを作成します。

   ```
   CREATE EXTERNAL SCHEMA evdata FROM KINESIS
   IAM_ROLE 'arn:aws:iam::0123456789:role/redshift-streaming-role';
   ```

   IAM ロールの設定方法については、「Amazon Kinesis Data Streams からストリーミング取り込みを開始する方法」を参照してください。

2. ストリームデータを利用するためのマテリアライズドビューを作成します。次の例に、JSON ソースデータの取り込のためにマテリアライズドビューを定義する 2 つの方法を示します。

   まず、ストリームレコードを半構造化された SUPER 形式で保存します。この例では、JSON ソースを Redshift 形に変換せずに Redshift に格納します。

   ```
   CREATE MATERIALIZED VIEW ev_station_data AS
       SELECT approximate_arrival_timestamp,
       partition_key,
       shard_id,
       sequence_number,
       case when can_json_parse(kinesis_data) then json_parse(kinesis_data) else null
    end as payload,
       case when not can_json_parse(kinesis_data) then kinesis_data else null end as
    failed_payload
       FROM evdata."ev_station_data" ;
   ```

   対照的に次の定義例では、マテリアライズドビューは Redshift で定義されたスキーマを持っています。マテリアライズドビューは、ストリームからの UUID 値に分散された上で、`approximatearrivaltimestamp`値により保存されます。

   ```
   CREATE MATERIALIZED VIEW ev_station_data_extract DISTKEY(6) sortkey(1) AUTO REFRESH
    YES AS
   ```

```
    SELECT refresh_time,
    approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'_id',true)::character(36)
 as ID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'clusterID',true)::varchar(30)
 as clusterID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'connectionTime',true)::varchar(
 as connectionTime,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'kWhDelivered',true)::DECIMAL(10
 as kWhDelivered,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'stationID',true)::DECIMAL(10,2)
 as stationID,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'spaceID',true)::varchar(100)
 as spaceID,
    json_extract_path_text(from_varbyte(kinesis_data,
 'utf-8'),'timezone',true)::varchar(30)as timezone,

 json_extract_path_text(from_varbyte(kinesis_data,'utf-8'),'userID',true)::varchar(30)
 as userID
    FROM evdata."ev_station_data"
    WHERE LENGTH(kinesis_data) < 65355;
```

ストリームをクエリする

1. リフレッシュしたマテリアライズドビューをクエリし、使用状況に関する統計を取得します。

```
SELECT to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS') as connectiontime
,SUM(kWhDelivered) AS Energy_Consumed
,count(distinct userID) AS #Users
from ev_station_data_extract
group by  to_timestamp(connectionTime, 'YYYY-MM-DD HH24:MI:SS')
order by 1 desc;
```

2. 結果を表示します。

```
connectiontime         energy_consumed #users
2022-02-08 16:07:21+00   4139         10
2022-02-08 16:07:20+00   5571         10
2022-02-08 16:07:19+00   8697         20
2022-02-08 16:07:18+00   4408         10
2022-02-08 16:07:17+00   4257         10
2022-02-08 16:07:16+00   6861         10
2022-02-08 16:07:15+00   5643         10
2022-02-08 16:07:14+00   3677         10
2022-02-08 16:07:13+00   4673         10
2022-02-08 16:07:11+00   9689         20
```

AWS Glue Data Catalog のビュー

このトピックでは、AWS Glue Data Catalog でのビューの作成方法について説明します。Data Catalog のビューを使用すると、同じスキーマを使用して異なるデータソースのデータにアクセスできます。

Data Catalog のビューを作成することで、Amazon Athena や Amazon EMR Spark などのエンジン間で使用できる単一の共通ビュースキーマとメタデータオブジェクトを作成できます。そうすることで、ユースケースに合わせてデータレイクとデータウェアハウスの全体で同じビューを使用できます。データカタログのビューは、定義者ビューとして分類されている点で特別です。定義者ビューでは、アクセス許可はビューをクエリするユーザーではなく、ビューを作成したユーザーによって定義されます。データカタログのビューを作成するユースケースとメリットをいくつか紹介します。

- ユーザーが必要とするアクセス許可に基づいてデータアクセスを制限するビューを作成します。例えば、データカタログのビューを使用して、人事 (HR) 部門に属さない従業員が個人を特定できる情報 (PII) を表示できないようにすることができます。
- ユーザーが不完全なレコードにアクセスできないようにします。データカタログのビューに特定のフィルターを適用することで、データカタログビュー内のデータレコードを常に完全な状態にすることができます。
- データカタログビューには、ビューの作成に使用するクエリ定義がビューの作成のための完全性を必ず備える必要であるというセキュリティ上のメリットが含まれます。このセキュリティ上のメリットにより、データカタログのビューは悪意のあるプレイヤーからの SQL コマンドの影響を受けにくくなります。
- データカタログのビューには、基になるテーブルをユーザーが利用できなくてもユーザーがビューにアクセスできるなどの、通常のビューと同じメリットをサポートしています。

データカタログ内にビューを作成するには、Spectrum 外部テーブル、Lake Formation 管理のデータ共有に含まれるオブジェクト、または Apache Iceberg テーブルが必要です。

データカタログビュー定義は AWS Glue Data Catalog に保存されます。AWS Lake Formation を使用して、リソース許可、列付与、またはタグベースのアクセスコントロールを通じてアクセスを許可します。Lake Formation でのアクセス権の付与と取り消しについての詳細は、「データカタログリソースに対するアクセス許可の付与と取り消し」を参照してください。

前提条件

データカタログ内にビューを作成する前に、以下の前提条件が完了していることを確認します。

- 次の信頼ポリシーが IAM ロールに定義されていることを確認します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
               "Service": [
                    "glue.amazonaws.com",
                    "lakeformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

- また、以下のパスロールポリシーも必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1",
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "glue.amazonaws.com",
                        "lakeformation.amazonaws.com"
                    ]
                }
            }
        }
```

```
    ]
}
```

- 最後に、次のアクセス許可も必要です。
  - `Glue:GetDatabase`
  - `Glue:GetDatabases`
  - `Glue:CreateTable`
  - `Glue:GetTable`
  - `Glue:UpdateTable`
  - `Glue:DeleteTable`
  - `Glue:GetTables`
  - `Glue:SearchTables`
  - `Glue:BatchGetPartition`
  - `Glue:GetPartitions`
  - `Glue:GetPartition`
  - `Glue:GetTableVersion`
  - `Glue:GetTableVersions`

エンドツーエンドの例

まず、データカタログデータベースに基づいて外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA IF NOT EXISTS external_schema FROM DATA CATALOG DATABASE
 'external_data_catalog_db'
IAM_ROLE 'arn:aws:iam::123456789012:role/sample-role';
```

データカタログビューを作成できるようになります。

```
CREATE EXTERNAL PROTECTED VIEW external_schema.remote_view
AS SELECT * FROM external_schema.remote_table;
```

これで、ビューのクエリを開始できます。

```
SELECT * FROM external_schema.remote_view;
```

データカタログ内のビューに関連する SQL コマンドの詳細については、「外部ビューの作成」、「外部ビューの変更」、および「外部ビューの削除」を参照してください。

セキュアなログ記録

Redshift は、クエリがマルチダイアレクトのグルービューを参照するときに、Redshift システムログに記録されたメタデータをマスクします。マルチダイアレクトとは、ビューが Redshift や Amazon EMR などのさまざまなクエリエンジンの SQL ダイアレクトをサポートしていることを意味します。次の表のデータは、同じクエリ ID を持つすべてのクエリに対してマスクされます。次の表は、セキュアなログ記録が適用されたシステムビューと列を示しています。

システムテーブル	機密列
SYS_EXTERNAL_QUERY_DETAIL	列: source_type、total_partitions、qualified_partitions、scanned_files、returned_rows、returned_bytes、file_format、file_location、external_query_text、warning_message。詳細については、「SYS_EXTERNAL_QUERY_DETAIL」を参照してください。
SYS_EXTERNAL_QUERY_ERROR	列: file_location、rowid、column_name、original_value、modified_value、trigger、action、action_value、error_code。詳細については、「SYS_EXTERNAL_QUERY_ERROR」を参照してください。
SYS_QUERY_DETAIL	列: step_name、table_id、table_name、input_bytes、input_rows、output_bytes、output_rows、blocks_read、blocks_write、local_read_IO、remote_read_IO、spilled_block_local_disk、spilled_block_remote_disk。詳細については、「SYS_QUERY_DETAIL」を参照してください。
SYS_QUERY_HISTORY	列: returned_rows、returned_bytes。詳細については、「SYS_QUERY_HISTORY」を参照してください。
STL_AGGR	列: rows、bytes、tbl、type。詳細については、「STL_AGGR」を参照してください。
STL_BCAST	列: rows、bytes、packets。詳細については、「STL_BCAST」を参照してください。
STL_DDLTEXT	列: text。詳細については、「STL_DDLTEXT」を参照してください。

システムテーブル	機密列
STL_DELETE	列: rows、tbl。詳細については、「STL_DELETE」を参照してください。
STL_DIST	列: rows、bytes、packets。詳細については、「STL_DIST」を参照してください。
STL_EXPLAIN	列: plannode、info。詳細については、「STL_EXPLAIN」を参照してください。
STL_HASH	列: rows、bytes、tbl、est_rows。詳細については、「STL_HASH」を参照してください。
STL_HASHJOIN	列: rows、tbl、num_parts、join_type。詳細については、「STL_HASHJOIN」を参照してください。
STL_INSERT	列: rows、tbl。詳細については、「STL_INSERT」を参照してください。
STL_LIMIT	列: rows。詳細については、「STL_LIMIT」を参照してください。
STL_MERGE	列: rows。詳細については、「STL_MERGE」を参照してください。
STL_MERGEJOIN	列: rows、tbl。詳細については、「STL_MERGEJOIN」を参照してください。
STL_NESTLOOP	列: rows、tbl。詳細については、「STL_NESTLOOP」を参照してください。
STL_PARSE	列: rows。詳細については、「STL_PARSE」を参照してください。
STL_PLAN_INFO	列: rows、bytes。詳細については、「STL_PLAN_INFO」を参照してください。
STL_PROJECT	列: rows、tbl。詳細については、「STL_PROJECT」を参照してください。
STL_QUERY	列: querytxt。詳細については、「STL_QUERY」を参照してください。

システムテーブル	機密列
STL_QUERY_METRICS	列: max_rows、rows、max_blocks_read、blocks_read、max_blocks_to_disk、blocks_to_disk、max_query_scan_size、query_scan_size。詳細については、「STL_QUERY_METRICS」を参照してください。
STL_QUERYTEXT	列: text。詳細については、「STL_QUERYTEXT」を参照してください。
STL_RETURN	列: rows、bytes。詳細については、「STL_RETURN」を参照してください。
STL_SAVE	列: rows、bytes、tbl。詳細については、「STL_SAVE」を参照してください。
STL_SCAN	列: rows、bytes、fetches、type、tbl、rows_pre_filter、perm_table_name、scanned_mega_value。詳細については、「STL_SCAN」を参照してください。
STL_SORT	列: rows、bytes、tbl。詳細については、「STL_SORT」を参照してください。
STL_TR_CONFLICT	列: table_id。詳細については、「STL_TR_CONFLICT」を参照してください。
STL_UNDONE	列: table_id。詳細については、「STL_UNDONE」を参照してください。
STL_UNIQUE	列: rows、type、bytes。詳細については、「STL_UNIQUE」を参照してください。
STL_UTILITYTEXT	列: text。詳細については、「STL_UTILITYTEXT」を参照してください。
STL_WINDOW	列: rows。詳細については、「STL_WINDOW」を参照してください。
STV_BLOCKLIST	列: col、tbl、num_values、minvalue、maxvalue。詳細については、「STV_BLOCKLIST」を参照してください。

システムテーブル	機密列
STV_EXEC_STATE	列: rows、bytes、label。詳細については、「STV_EXEC_STATE」を参照してください。
STV_LOCKS	列: table_id。詳細については、「STV_LOCKS」を参照してください。
STV_QUERY_METRICS	列: rows、rowsmax_rows、blocks_read、max_blocks_read、max_blocks_to_disk、blocks_to_disk、max_query_scan_size、query_scan_size。詳細については、「STV_QUERY_METRICS」を参照してください。
STV_STARTUP_RECOVERY_STATE	列: table_id、table_name。詳細については、「STV_STARTUP_RECOVERY_STATE」を参照してください。
STV_TBL_PERM	列: ID、rows、sorted_rows、temp、block_count、query_scan_size。詳細については、「STV_TBL_PERM」を参照してください。
STV_TBL_TRANS	列: id、rows、size。詳細については、「STV_TBL_TRANS」を参照してください。
SVCS_EXPLAIN	列: plannode、info。詳細については、「SVCS_EXPLAIN」を参照してください。
SVCS_PLAN_INFO	列: rows、bytes。詳細については、「SVCS_PLAN_INFO」を参照してください。
SVCS_QUERY_SUMMARY	列: step、rows、bytes、rate_row、rate_byte、label、rows_pre_filter。詳細については、「SVCS_QUERY_SUMMARY」を参照してください。
SVCS_S3LIST	列: bucket、prefix、max_file_size、avg_file_size。詳細については、「SVCS_QUERY_SUMMARY」を参照してください。
SVCS_S3LOG	列: message。詳細については、「SVCS_QUERY_SUMMARY」を参照してください。

システムテーブル	機密列
SVCS_S3PARTITION_SUMMARY	列: total_partitions、qualified_partitions、min_assigned_partitions、max_assigned_partitions、avg_assigned_partitions。詳細については、「SVCS_S3PARTITION_SUMMARY」を参照してください。
SVCS_S3QUERY_SUMMARY	列: external_table_name、file_format、s3_scanned_rows、s3_scanned_bytes、s3query_returned_rows、s3query_returned_bytes。詳細については、「SVCS_S3QUERY_SUMMARY」を参照してください。
SVL_QUERY_METRICS	列: step_label、Scan_row_count、join_row_count、nested_loop_join_row_count、return_row_count、spectal_scan_row_count、spectal_scan_size_mb。詳細については、「SVL_QUERY_METRICS」を参照してください。
SVL_QUERY_METRICS_SUMMARY	列: step_label、Scan_row_count、join_row_count、nested_loop_join_row_count、return_row_count、spectal_scan_row_count、spectal_scan_size_mb。詳細については、「SVL_QUERY_METRICS_SUMMARY」を参照してください。
SVL_QUERY_REPORT	列: rows、bytes、label、rows_pre_filter。詳細については、「SVL_QUERY_REPORT」を参照してください。
SVL_QUERY_SUMMARY	列: rows、bytes、rows_pre_filter。詳細については、「SVL_QUERY_SUMMARY」を参照してください。
SVL_S3LIST	列: bucket、prefix、max_file_size、avg_file_size。詳細については、「SVL_S3LIST」を参照してください。
SVL_S3LOG	列: message。詳細については、「SVL_S3LOG」を参照してください。
SVL_S3PARTITION	列: rows、bytes、label、rows_pre_filter。詳細については、「SVL_S3PARTITION」を参照してください。
SVL_S3PARTITION_SUMMARY	列: total_partitions、qualified_partitions、min_assigned_partitions、max_assigned_partitions、avg_assigned_partitions。詳細については、「SVL_S3PARTITION_SUMMARY」を参照してください。

システムテーブル	機密列
SVL_S3QUERY	列: external_table_name、file_format、s3_scanned_rows、s3_scanned_bytes、s3query_returned_rows、s3query_returned_bytes。詳細については、「SVL_S3QUERY」を参照してください。
SVL_S3QUERY_SUMMARY	列: external_table_name、file_format、s3_scanned_rows、s3_scanned_bytes、s3query_returned_rows、s3query_returned_bytes。詳細については、「SVL_S3QUERY_SUMMARY」を参照してください。
SVL_S3RETRIES	列: file_size、location、message。詳細については、「SVL_S3RETRIES」を参照してください。
SVL_SPECTRUM_SCAN_ERROR	列: location、rowid、colname、 original_value、modified_value。詳細については、「SVL_SPECTRUM_SCAN_ERROR」を参照してください。
SVL_STATEMENTTEXT	列: type、text。詳細については、「SVL_STATEMENTTEXT」を参照してください。
SVL_STORED_PROC_CALL	列: querytxt。詳細については、「SVL_STORED_PROC_CALL」を参照してください。
SVL_STORED_PROC_MESSAGES	列: querytext。詳細については、「SVL_STORED_PROC_MESSAGES」を参照してください。
SVL_UDF_LOG	列: funcname。詳細については、「SVL_UDF_LOG」を参照してください。
SVV_DISKUSAGE	列: name、col、tbl、blocknum、num_values、minvalue、maxvalue。詳細については、「SVV_DISKUSAGE」を参照してください。
SVV_QUERY_STATE	列: rows、bytes、label。詳細については、「SVV_QUERY_STATE」を参照してください。
SVV_TABLE_INFO	列: table_id、table。詳細については、「SVV_TABLE_INFO」を参照してください。

システムテーブル	機密列
SVV_TRANSACTIONS	列: relation。詳細については、「SVV_TRANSACTIONS」を参照してください。

考慮事項と制限事項

データカタログ内に作成されたビューに適用される考慮事項と制限は、以下のとおりです。

- 別のビューを基にしたデータカタログビューは作成できません。
- データカタログビューに作成できるベーステーブルは 10 個のみです。
- ビューの定義者は、ベーステーブルに対する完全な `SELECT GRANTABLE` アクセス許可を持っている必要があります。
- ビューには、Lake Formation オブジェクトとビルトインのみを含めることができます。以下のオブジェクトはビュー内には入れることができません。
  - システムテーブル
  - ユーザー定義関数 (UDF)
  - Redshift のテーブル、ビュー、マテリアライズドビュー、遅延バインディングビューであって、Lake Formation 管理のデータ共有内にないもの。
- ビューにはネストされた Redshift Spectrum テーブルを含めることはできません。
- ビューの基本オブジェクトについての AWS Glue の表現は、ビューと同じ AWS アカウント とリージョンになければなりません。

Amazon Redshift での空間データのクエリ

空間データは、定義された空間 (空間参照系) におけるジオメトリの位置と形状を説明するものです。Amazon Redshift は、GEOMETRY および GEOGRAPHY データ型の空間データをサポートします。これには空間データが含まれており、さらにオプションでデータの空間参照系識別子 (SRID) を使用できます。

空間データには、ジオメトリの特徴を表すために使用できるジオメトリデータが含まれます。このタイプのデータの例としては、天気予報、地図の案内、位置情報を含むツイート、店舗の場所、航空路線などがあります。空間データは、ビジネス分析、レポート、および予測で重要な役割を果たします。

空間データは、Amazon Redshift SQL 関数を使ってクエリできます。空間データには、オブジェクトのジオメトリ値が含まれます。

GEOMETRY データ型の演算は、デカルト平面上で動作します。空間参照系識別子 (SRID) はオブジェクト内に格納されますが、この SRID は座標系の識別子に過ぎず、GEOMETRY オブジェクトの処理で使用するアルゴリズム内の処理とは特に関係がありません。一方、GEOGRAPHY データ型の演算では、オブジェクト内の座標は回転楕円体上の球座標として扱われます。この回転楕円体は、地理空間参照系を参照する SRID によって定義されます。デフォルトで GEOGRAPHY データ型は、世界測地系 (WGS) 84 を参照しながら、空間参照 (SRID) 4326 を使用して作成されます。SRA の詳細については、Wikipedia で「Spatial reference system」を参照してください。

ST_Transform 関数を使用すると、さまざまな空間参照系からの座標を変換できます。座標変換の完了後は、これら 2 つの座標間で単純なキャストを使用できるようになります (ただし、入力された GEOMETRY が地理的 SRID でエンコードされている場合に限ります)。このキャストでは追加的な変換は行わず、単に座標をコピーします。例:

```
SELECT ST_AsEWKT(ST_GeomFromEWKT('SRID=4326;POINT(10 20)')::geography);
```

```
st_asewkt
------------------------
 SRID=4326;POINT(10 20)
```

GEOMETRY と GEOGRAPHY データ型間での違いをよりよく理解するためには、世界測地系 (WGS) 84 を使用して、ベルリン空港 (BER) とサンフランシスコ空港 (SFO) の間の距離を計算してみるということもできます。GEOGRAPHY データ型を使用した場合、結果はメートル単位で返されます。SRID

4326 による GEOMETRY データ型を使用する場合、結果は度数で返されます。1 度に対応する距離が地球上でのジオメトリの位置によって異なるため、度数をメートルに変換することはできません。

GEOGRAPHY データ型での計算は、国の正確な面積をゆがみなく計算する場合など、地球面上での現実に沿った計算に主に使用されます。ただし、これらの処理ではコストが高くなります。したがって、ST_Transform により、ローカルに投影された適切な座標系に座標を変換することで、GEOMETRY データ型を使用して計算を高速化できるようになります。

空間データを使って、以下を実行するクエリを実行できます。

- 2 点間の距離を確認する。
- あるエリア (ポリゴン) 内に別のエリアが含まれているかどうかを確認する。
- あるラインストリングが別のラインストリングまたはポリゴンと交差するかどうかを確認する。

空間データの値を保持するには、GEOMETRY データ型を使います。Amazon Redshift のGEOMETRY 値では、2 次元 (2D)、3 次元 (3DZ)、メジャー付 2 次元 (3DM)、および 4 次元 (4D) ジオメトリの、プリミティブデータ型を定義できます。

- 2 次元 (2D) ジオメトリは、平面上の 2 つのデカルト座標 (x, y) によって表現されます。
- 3 次元 (3DZ) ジオメトリは、空間内の 3 つのデカルト座標 (x, y, z) によって表現されます。
- メジャー付き 2次元 (3DM) ジオメトリは、3 つの座標 (x、y、m) によって表現されます。最初の 2 つは平面内のデカルト座標であり、3 つ目は測定尺度です。
- 4 次元 (4D) ジオメトリは、4 つの座標 (x, y, z, m) によって表現されます。最初の 3 つは空間内のデカルト座標、4 つ目は測定尺度です。

ジオメトリプリミティブのデータ型に関する詳細は、ウィキペディアの Well-known text を参照してください。

空間データの値を保持するには、GEOGRAPHY データ型を使います。Amazon Redshift のGEOGRAPHY 値では、2 次元 (2D)、3 次元 (3DZ)、メジャー付 2 次元 (3DM)、および 4 次元 (4D) ジオメトリの、プリミティブデータ型を定義できます。

- 2 次元 (2D) ジオメトリは、回転楕円体上の経度と緯度の座標によって表現されます。
- 3 次元 (3DZ) ジオメトリは、回転楕円体上の経度、緯度、および標高によって表現されます。
- メジャー付き 2 次元 (3DM) ジオメトリは、3 つの座標 (経度、緯度、メジャー) によって表現されます。最初の 2 つは球体面上の角座標であり、3 つ目は測定尺度です。

- 4 次元 (4D) ジオメトリは、4 つの座標 (経度、緯度、標高、メジャー) によって表現されます。最初の 3 つが経度、緯度、高度で、4 つ目は測定尺度です。

地理座標系の詳細については、ウィキペディアで「地理座標系」および「球形座標系」を参照してください。

GEOMETRY および GEOGRAPHY データ型には、以下のサブタイプがあります。

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

ジオメトリデータの次の表記をサポートする Amazon Redshift SQL 関数があります。

- GeoJSON
- Well-known text (WKT)
- Extended well-known text (EWKT)
- Well-known binary (WKB) 表記
- Extended well-known binary (EWKB)

GEOMETRY データ型と GEOGRAPHY データ型の間はキャストすることが可能です。

次の SQL は、GEOMETRY から GEOGRAPHY にラインストリングをキャストします。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geography);
```

```

 st_asewkt
----------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

次の SQL は、GEOGRAPHY から GEOMETRY にラインストリングをキャストします。

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)')::geometry);
```

```
 st_asewkt
----------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

Amazon Redshift は、空間データをクエリするための多くの SQL 関数が用意されています。`ST_IsValid` 関数を除いて、引数として GEOMETRY オブジェクトを受け入れる空間関数は、この GEOMETRY オブジェクトが有効なジオメトリであることを期待します。GEOMETRY あるいは GEOGRAPHY オブジェクトが有効でない場合、空間関数の動作が定義されていません。検証についての詳細は、幾何学的妥当性 を参照してください。

空間データをクエリする SQL 関数の詳細については、「空間関数」を参照してください。

空間データのロードの詳細については、「GEOMETRY もしくは GEOGRAPHY データ型の列のロード」を参照してください。

トピック

- チュートリアル: Amazon Redshift での空間 SQL 関数の使用
- シェープファイルを Amazon Redshift にロードする
- Amazon Redshift 空間データの用語
- Amazon Redshift で空間データを使用する際の考慮事項

チュートリアル: Amazon Redshift での空間 SQL 関数の使用

このチュートリアルでは、Amazon Redshift でいくつかの空間 SQL 関数を使用する方法を説明します。

これを行うには、空間 SQL 関数を使用して 2 つのテーブルをクエリします。このチュートリアルでは、ドイツのベルリンに位置する賃貸宿泊施設の位置データと郵便番号を関連付ける公開データセットのデータを使用します。

トピック

- 前提条件
- ステップ 1: テーブルを作成し、テストデータをロードする

- ステップ 2: 空間データのクエリを実行する
- ステップ 3: リソースをクリーンアップする

前提条件

このチュートリアルでは、以下のリソースが必要になります。

- アクセスおよび更新できる既存の Amazon Redshift クラスターおよびデータベース。既存のクラスターでは、テーブルを作成し、サンプルデータをロードし、SQL クエリを実行して空間関数を実証します。クラスターには 2 つ以上のノードが必要です。クラスターの作成方法を学ぶには、Amazon Redshift 入門ガイドの手順に従ってください。
- Amazon Redshift クエリエディタを使用するには、クエリエディタをサポートする AWS リージョンにあるクラスターを使用する必要があります。詳細については、「Amazon Redshift 管理ガイド」の「クエリエディタを使用してデータベースのクエリを実行する」を参照してください。
- Amazon S3 からのテストデータのロードを Amazon Redshift クラスターに許可する AWS 認証情報。AWS の他のサービス (Amazon S3 など) へのアクセス方法については、「Amazon Redshift が AWS サービスにアクセスすることを許可する」を参照してください。
- `mySpatialDemoRole` という名前の AWS Identity and Access Management (IAM) ロール。Amazon S3 データを読み取るためのマネージドポリシー `AmazonS3ReadOnlyAccess` がアタッチ済み。Amazon S3 バケットからデータをロードするアクセス許可を持つロールを作成するには、「Amazon Redshift 管理ガイド」の「IAM ロールを使用して COPY、UNLOAD、CREATE EXTERNAL SCHEMA オペレーションを認可する」を参照してください。
- IAM ロール `mySpatialDemoRole` を作成した後、そのロールには Amazon Redshift クラスターとの関連付けが必要です。その関連付けを作成する方法の詳細については、「Amazon Redshift 管理ガイド」の「IAM ロールを使用して COPY、UNLOAD、CREATE EXTERNAL SCHEMA オペレーションを認可する」を参照してください。

ステップ 1: テーブルを作成し、テストデータをロードする

このチュートリアルで使用されているソースデータは、`accommodations.csv` および `zipcodes.csv` という名前のファイルです。

`accommodations.csv` ファイルは、insideairbnb.com からのオープンソースデータです。`zipcodes.csv` ファイルは、ドイツのベルリンブランデンブルク国立統計研究所 (Amt für Statistik Berlin-Brandenburg) からのオープンソースデータである郵便番号を提供します。どちらのデータソースも、クリエイティブ· コモンズ· ライセンスで提供されています。データは、ドイ

ツ、ベルリンリージョンに限定されています。これらのファイルは、このチュートリアルで使用する Amazon S3 公開バケットにあります。

必要に応じて、次の Amazon S3 リンクからソースデータをダウンロードできます。

- accommodations テーブルのソースデータ。
- zipcode テーブルのソースデータ。

次の手順を使用して、テーブルを作成し、テストデータをロードします。

テーブルを作成してテストデータをロードするには

1. Amazon Redshift クエリエディタを開きます。クエリエディタ操作の詳細については、「Amazon Redshift 管理ガイド」の「クエリエディタを使用するデータベースのクエリ」を参照してください。
2. このチュートリアルで使用するテーブルがデータベースにすでに存在する場合は、削除してください。詳細については、「ステップ 3: リソースをクリーンアップする」を参照してください。
3. accommodations テーブルを作成して、各宿泊施設の地理的位置 (経度と緯度)、リストの名前、およびその他のビジネスデータを保存します。

   このチュートリアルでは、ドイツ、ベルリンの部屋レンタルについて説明します。shape 列には、宿泊施設の場所に関する地理的ポイントが保存されます。その他の列には、レンタルに関する情報が含まれています。

   accommodations テーブルを作成するには、Amazon Redshift クエリエディタで次の SQL ステートメントを実行します。

```
CREATE TABLE public.accommodations (
  id INTEGER PRIMARY KEY,
  shape GEOMETRY,
  name VARCHAR(100),
  host_name VARCHAR(100),
  neighbourhood_group VARCHAR(100),
  neighbourhood VARCHAR(100),
  room_type VARCHAR(100),
  price SMALLINT,
  minimum_nights SMALLINT,
  number_of_reviews SMALLINT,
  last_review DATE,
  reviews_per_month NUMERIC(8,2),
```

```
  calculated_host_listings_count SMALLINT,
  availability_365 SMALLINT
);
```

4. ベルリンの郵便番号を保存するために、クエリエディタで `zipcode` テーブルを作成します。

   郵便番号は、`wkb_geometry` 列のポリゴンとして定義されています。残りの列では、郵便番号に関する追加の空間メタデータについて説明しています。

   `zipcode` テーブルを作成するには、Amazon Redshift クエリエディタで次の SQL ステートメントを実行します。

```
CREATE TABLE public.zipcode (
  ogc_field INTEGER PRIMARY KEY NOT NULL,
  wkb_geometry GEOMETRY,
  gml_id VARCHAR(256),
  spatial_name VARCHAR(256),
  spatial_alias VARCHAR(256),
  spatial_type VARCHAR(256)
 );
```

5. サンプルデータをテーブルにロードします。

   このチュートリアルのサンプルデータは、認証済みのすべての AWS ユーザーに読み取りアクセスを許可する、Amazon S3 バケット内に用意されています。Amazon S3 へのアクセスを許可する、有効な AWS 認証情報を使用する必要があります。

   テストデータをテーブルにロードするには、次の COPY コマンドを実行します。*`account-number`* は自分の AWS アカウント番号に置き換えます。一重引用符で囲まれた認証情報文字列に、スペースまたは改行を含めることはできません。

```
COPY public.accommodations
FROM 's3://redshift-downloads/spatial-data/accommodations.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

```
COPY public.zipcode
FROM 's3://redshift-downloads/spatial-data/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
```

```
CREDENTIALS 'aws_iam_role=arn:aws:iam::account-number:role/mySpatialDemoRole';
```

6. 以下のコマンドを実行して、各テーブルが正しくロードされていることを確認します。

```
select count(*) from accommodations;
```

```
select count(*) from zipcode;
```

以下の結果の表に示しているのは、テストテーブルの各テーブルでの行数です。

テーブル名	Rows
宿泊施設	22,248
zipcode	190

ステップ 2: 空間データのクエリを実行する

テーブルを作成してロードした後、SQL SELECT ステートメントを使用してテーブルにクエリを実行できます。次のクエリは、取得できる情報の一部を示しています。ニーズを満たすために空間関数を使用する他の多くのクエリを書き込むことができます。

空間データをクエリするには

1. 次のように、クエリを実行して、accommodations テーブルに保存されているリストの総数を取得します。空間参照系は世界測地系 (WGS) 84 であり、一意の空間参照識別子 4326 を持ちます。

```
SELECT count(*) FROM public.accommodations WHERE ST_SRID(shape) = 4326;
```

```

 count
-------
 22248
```

2. いくつかの追加属性を使用して、ジオメトリオブジェクトを well-known text (WKT) 形式で取得します。さらに、この郵便番号データが空間参照 ID (SRID) 4326 を使用する世界測地システム

(WGS) 84 にも保存されているかどうかを検証できます。空間データを相互運用するには、空間データを同じ空間参照系に保存する必要があります。

```
SELECT ogc_field, spatial_name, spatial_type, ST_SRID(wkb_geometry),
 ST_AsText(wkb_geometry)
FROM public.zipcode
ORDER BY spatial_name;
```

```
ogc_field  spatial_name  spatial_type  st_srid  st_astext
---------------------------------------------------------------
0           10115        Polygon        4326     POLYGON((...))
4           10117        Polygon        4326     POLYGON((...))
8           10119        Polygon        4326     POLYGON((...))
...
(190 rows returned)
```

3. ベルリンの自治区であるベルリンミッテ (10117) のポリゴンを GeoJSON 形式で選択し、そのディメンション、およびこのポリゴンのポイント数を選択します。

```
SELECT ogc_field, spatial_name, ST_AsGeoJSON(wkb_geometry),
 ST_Dimension(wkb_geometry), ST_NPoints(wkb_geometry)
FROM public.zipcode
WHERE spatial_name='10117';
```

```
ogc_field  spatial_name  spatial_type
 st_dimension  st_npoint
-----------------------------------------------------------------------------------------------
4           10117         {"type":"Polygon", "coordinates":[[[...]]]}    2
    331
```

4. 次の SQL コマンドを実行して、ブランデンブルク門から 500 メートル以内にある宿泊施設の数を確認します。

```
SELECT count(*)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500;
```

```
count
------
  29
```

5. 次のクエリを実行して、近いと表示されている宿泊施設に保存されているデータから、ブランデンブルク門の大まかな位置を取得します。

   このクエリには、複数の選択が必要です。リクエストされた場所が宿泊施設に近いため、以前のクエリと同じではないため、カウントが異なります。

```
WITH poi(loc) as (
  SELECT st_astext(shape) FROM accommodations WHERE name LIKE '%brandenburg gate%'
)
SELECT count(*)
FROM accommodations a, poi p
WHERE ST_DistanceSphere(a.shape, ST_GeomFromText(p.loc, 4326)) < 500;
```

```
count
------
  60
```

6. 次のクエリを実行して、ブランデンブルク門周辺のすべての宿泊施設の詳細を料金の降順に表示します。

```
SELECT name, price, ST_AsText(shape)
FROM public.accommodations
WHERE ST_DistanceSphere(shape, ST_GeomFromText('POINT(13.377704 52.516431)', 4326))
 < 500
ORDER BY price DESC;
```

```
name                                                    price  st_astext
-------------------------------------------------------------------------------------------
DUPLEX APARTMENT/PENTHOUSE in 5* LOCATION! 7583         300
 POINT(13.3826510209548 52.5159819722552)
DUPLEX-PENTHOUSE IN FIRST LOCATION! 7582                300
 POINT(13.3799997083855 52.5135918444834)
...
```

```
(29 rows returned)
```

7. 次のクエリを実行して、最も高価な宿泊施設を郵便番号で取得します。

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE price = 9000 AND ST_Within(a.shape, z.wkb_geometry);
```

```
price   name                                 st_astext
     spatial_name      st_astext
---------------------------------------------------------------------------------------------------
9000    Ueber den Dächern Berlins Zentrum    POINT(13.334436985013
 52.4979779501538)    10777             POLYGON((13.3318284987227
 52.4956021172799,...
```

8. サブクエリを使用して、宿泊施設の最大料金、最小料金、または中央値を計算します。

次のクエリは、宿泊施設の料金の中央値を郵便番号別に示しています。

```
SELECT
  a.price, a.name, ST_AsText(a.shape),
  z.spatial_name, ST_AsText(z.wkb_geometry)
FROM accommodations a, zipcode z
WHERE
  ST_Within(a.shape, z.wkb_geometry) AND
  price = (SELECT median(price) FROM accommodations)
ORDER BY a.price;
```

```
price name                            st_astext
 spatial_name   st_astext
---------------------------------------------------------------------------------------------------


45    "Cozy room Berlin-Mitte"        POINT(13.3864349535358 52.5292016386514)
 10115          POLYGON((13.3658598465795 52.535659581048,...
...
(723 rows returned)
```

9. 次のクエリを実行して、ベルリンにリストされている宿泊施設の数を取得します。ホットスポットを見つけるために、これらは郵便番号でグループ化され、供給量によってソートされます。

```
SELECT z.spatial_name as zip, count(*) as numAccommodations
FROM public.accommodations a, public.zipcode z
WHERE ST_Within(a.shape, z.wkb_geometry)
GROUP BY zip
ORDER BY numAccommodations DESC;
```

```
zip   numaccommodations
----------------------------
10245 872
10247 832
10437 733
10115 664
...
(187 rows returned)
```

ステップ 3: リソースをクリーンアップする

クラスターが実行されている限り料金が発生し続けます。このチュートリアルを完了したら、サンプルクラスターを削除できます。

クラスターを維持したいが、テストデータテーブルで使用されるストレージを復元するには、以下のコマンドを実行してテーブルを削除します。

```
drop table public.accommodations cascade;
```

```
drop table public.zipcode cascade;
```

シェープファイルを Amazon Redshift にロードする

COPY コマンドを使用して、Amazon S3 に保存されている Esri シェープファイルを Amazon Redshift テーブルに取り込むことができます。シェープファイルは、地理的特徴の幾何学的位置と属性情報をベクトル形式で保存します。シェープファイル形式は、ポイント、ライン、ポリゴンなどの空間オブジェクトを空間的に記述できます。シェープファイルの詳細については、ウィキペディアで[シェープファイル](#)を参照してください。

COPY コマンドは、データ形式パラメータ SHAPEFILE をサポートします。デフォルトでは、シェープファイルの最初の列は GEOMETRY 列または IDENTITY 列のいずれかです。後続のすべての列は、シェープファイルで指定された順序に従います。ただし、COPY 列マッピングを使用して順序を定義できるため、ターゲットテーブルはこの正確なレイアウトである必要はありません。COPY コマンドのシェープファイルのサポートについては、「SHAPEFILE」を参照してください。

場合によっては、生成されるジオメトリのサイズが Amazon Redshift にジオメトリを保存するための最大値よりも大きくなることがあります。その場合は、次のように COPY オプション SIMPLIFY または SIMPLIFY AUTO を使用して、取り込み中の形状を単純化できます。

- SIMPLIFY *tolerance* を指定して、Ramer-Douglas-Peucker アルゴリズムと指定された許容値を使用して、取り込み中にすべてのジオメトリを簡略化します。
- Ramer-Douglas-Peucker アルゴリズムを使用して最大サイズよりも大きい形状のみを簡略化するには、許容誤差なしで SIMPLIFY AUTO を指定します。このアプローチでは、オブジェクトを最大サイズの制限内で保存する際に十分な大きさの最小許容値を計算します。
- Ramer-Douglas-Peucker アルゴリズムと自動的に計算された許容誤差を使用して、最大サイズよりも大きい形状のみを簡略化するには、SIMPLIFY AUTO *max_tolerance* を指定します。このアプローチにより、許容値が最大許容値を超えないようにします。

GEOMETRY データ値の最大サイズについては、「Amazon Redshift で空間データを使用する際の考慮事項」を参照してください。

場合によっては、許容値が十分に低く、レコードが GEOMETRY データ値の最大サイズを下回って縮小できないことがあります。このような場合、COPY コマンドの MAXERROR オプションを使用して、すべてまたは特定の数までの取り込みエラーを無視できます。

COPY コマンドは、GZIP シェープファイルのロードもサポートしています。これを行うには、COPY GZIP パラメータを指定します。このオプションでは、すべてのシェープファイルコンポーネントを個別に圧縮し、同じ圧縮サフィックスを共有する必要があります。

シェープファイルとともに投影定義ファイル (.prj) が存在する場合、Redshift はそのファイルを使用して空間参照系 ID (SRID) を特定します。SRID が有効である場合、結果として得られるジオメトリにはこの SRID が割り当てられています。入力ジオメトリに関連付けられている SRID 値が存在しない場合、結果として得られるジオメトリの SRID 値はゼロになります。セッションレベルでの空間参照系 ID の自動検出は、SET read_srid_on_shapefile_ingestion を OFF に設定することで無効にできます。

SYS_SPATIAL_SIMPLIFY または SVL_SPATIAL_SIMPLIFY システムビューにクエリを実行して、どのレコードが簡略されているかを、計算された許容値とともに表示します。SIMPLIFY *tolerance* を指定すると、このビューには各 COPY オペレーションのレコードが含まれます。それ以外の場合は、簡略化された各ジオメトリのレコードが含まれます。詳細については、「SYS_SPATIAL_SIMPLIFY」または「SVL_SPATIAL_SIMPLIFY」を参照してください。

シェープファイルのロードの例については、「シェープファイルを Amazon Redshift にロードする」を参照してください。

Amazon Redshift 空間データの用語

以下の用語は、一部の Amazon Redshift 空間関数を説明するために使用されます。

境界ボックス

ジオメトリまたはジオグラフィの境界ボックスは、そのジオメトリまたはジオグラフィのすべてのポイントの座標範囲 (ディメンション間) での外積として定義されます。2 次元ジオメトリの場合、境界ボックスは、ジオメトリ内のすべてのポイントを完全に含む長方形となります。例えば、ポリゴン `POLYGON((0 0,1 0,0 2,0 0))` の境界ボックスは、点 (0, 0) と (1, 2) が、それぞれ左下および右上隅として定義される長方形です。Amazon Redshift は、ジオメトリ内の境界ボックスを事前に計算して格納します。この処理により、幾何学的な述語と空間結合を高速化します。例えば、2 つのジオメトリの境界ボックスが交差しないのであれば、これらの 2 つのジオメトリは交差することはなく、したがって ST_Intersects 述語により実行される空間結合の結果セットに含めることはできません。

境界ボックスに対し空間関数を使用して、追加 (AddBBox)、ドロップ (DropBBox)、サポートの判断 (SupportsBBox) が実行できます。Amazon Redshift は、すべてのジオメトリサブタイプで、境界ボックスの事前計算をサポートしています。

次の例は、テーブル内の既存のジオメトリを更新して、境界ボックスを使用しながら、そのジオメトリを格納する方法を示しています。使用しているクラスターのバージョンが 1.0.26809 以降の場合は、デフォルトで、すべての新しいジオメトリは事前に計算された境界ボックスを使用しで作成されます。

```
UPDATE my_table SET geom = AddBBox(geom) WHERE SupportsBBox(geom) = false;
```

既存のジオメトリの更新処理後は、更新されたテーブルに対して VACUUM コマンドを実行することをお勧めします。詳細については、「VACUUM」を参照してください。

セッション中にジオメトリを境界ボックスによりエンコードするかどうかを設定するには、「default_geometry_encoding」を参照してください。

幾何学的妥当性

Amazon Redshift で使用されるジオメトリックアルゴリズムは、入力ジオメトリが有効なジオメトリであると仮定しています。アルゴリズムへの入力が有効でない場合、結果は未定義です。以下のセクションでは、各ジオメトリのサブタイプに対して Amazon Redshift が使用するジオメトリの妥当性の定義について説明します。

Point

ポイントは、次の条件の内 1 つが true である場合に有効と見なされます。

- ポイントが空である。
- すべてのポイントの座標が、有限の浮動小数点数で示されている。

ポイントは空にすることができます。

Linestring

ライン文字列は、次の条件のいずれかが true である場合に有効と見なされます。

- ライン文字列は空です。つまり、ポイントは含まれていません。
- 空ではないライン文字列内のすべてのポイントは、有限の浮動小数点数である座標を持っています。
- ライン文字列が空ではない場合、1 次元である必要があります。つまり、ポイントまで縮退することはできません。

ライン文字列には、空のポイントを含めることはできません。

ライン文字列は、連続した重複ポイントを持つことができます。

ライン文字列は、それ自身と交差することができます。

Polygon

ポリゴンは、次の条件のいずれかが true である場合に有効と見なされます。

- ポリゴンは空です。つまり、リングが含まれていません。
- 空でない場合は、次の条件がすべて true である場合、ポリゴンは有効です。
  - ポリゴンのすべてのリングは有効です。リングは、次の条件のすべてが true である場合に有効と見なされます。

- リングのすべての点は、有限の浮動小数点数である座標を持っています。
- リングは閉じています。つまり、最初のポイントと最後のポイントが一致します。
- リングには自己交差がありません。
- リングは 2 次元です。
- ポリゴンのリングの向きは一貫しています。つまり、任意のリングを横断する場合、ポリゴンの内部は右または左のいずれかになります。つまり、ポリゴンの外部リングが時計回りまたは反時計回りに向けられている場合、ポリゴンの内部リングはすべて同じ反時計回りまたは時計回りに向けられている必要があります。
- すべての内部リングは、ポリゴンの外部リング内にある必要があります。
- 内部リングはネストできません。つまり、内部リングを別の内部リング内に配置することはできません。
- 内部リングと外部リングは、有限数のポイントでのみ交差できます。
- ポリゴンの内部は簡単に接続する必要があります。

ポリゴンに空のポイントを含めることはできません。

Multipoint

マルチポイントは、次の条件のいずれかが true である場合に有効と見なされます。

- マルチポイントは空です。つまり、ポイントは含まれていません。
- マルチポイントは空ではなく、含まれるすべてのポイントが、ポイント有効性の定義に従って有効である。

マルチポイントには、1 つ以上の空のポイントを含めることができます。

マルチポイントは重複ポイントを持つことができます。

マルチライン文字列

マルチライン文字列は、次の条件のいずれかが true である場合に有効と見なされます。

- マルチライン文字列は空です。つまり、ライン文字列は含まれていません。
- 空ではないマルチライン文字列内のすべてのライン文字列は、ライン文字列の有効性の定義に従って有効になります。

空のライン文字列のみで構成される空ではないマルチライン文字列は、有効であると見なされます。

マルチライン文字列内の空のライン文字列は、そのマルチライン文字列の有効性に影響を与えません。

マルチライン文字列には、連続するポイントが重複するライン文字列を含めることができます。

マルチライン文字列は、自己交差することができます。

マルチライン文字列には空のポイントを含めることはできません。

Multipolygon

マルチポリゴンは、次の条件のいずれかが true である場合に有効と見なされます。

- マルチポリゴンにはポリゴンが含まれていません (空です)。
- マルチポリゴンは空ではなく、以下のすべてが true です。
  - マルチポリゴンのすべてのポリゴンは有効です。
  - マルチポリゴン内の 2 つのポリゴンが無限数のポイントで交差することはできません。特に、これは、任意の 2 つのポリゴンの内部が交差できず、有限数のポイントでしか接触できないことを意味します。

マルチポリゴン内の空のポリゴンは、マルチポリゴンを無効化しません。

マルチポリゴンには空のポイントを含めることはできません。

ジオメトリコレクション

ジオメトリコレクションは、次の条件のいずれかが true である場合に有効と見なされます。

- ジオメトリコレクションは空です。つまり、ジオメトリは含まれていません。
- 空でないジオメトリコレクション内のすべてのジオメトリが有効です。

この定義は、再帰的にではありますが、ネストされたジオメトリコレクションにも適用されます。

ジオメトリコレクションには、空のポイントと、空のポイントを持つマルチポイントを含めることができます。

幾何学的な単純度

Amazon Redshift で使用されるジオメトリックアルゴリズムは、入力ジオメトリが有効なジオメトリであると仮定しています。アルゴリズムへの入力が有効でない場合、簡略化のチェックは定義されていません。以下のセクションでは、各ジオメトリのサブタイプに対して Amazon Redshift が使用するジオメトリの簡略化の定義について説明します。

Point

有効なポイントは、以下の条件のいずれかが true である場合にシンプルと見なされます。

- 有効なポイントは、常に単純であるとみなされます。
- 空のポイントはシンプルであるとみなされます。

Linestring

有効なライン文字列は、次の条件のいずれかが true である場合にシンプルと見なされます。

- ライン文字列が空です。
- ライン文字列は空ではなく、以下のすべての条件が true です。
  - 重複する連続ポイントはありません。
  - 一致する可能性のある最初のポイントと最後のポイントを除いて、自己交差点はありません。つまり、ライン文字列は、境界点以外で自己交差することはできません。

Polygon

有効なポリゴンは、重複する連続するポイントが含まれていない場合、単純であると見なされます。

Multipoint

有効なマルチポイントは、次の条件のいずれかが true である場合にシンプルと見なされます。

- マルチポイントは空です。つまり、ポイントは含まれていません。
- マルチポイントの空でないポイント 2 つが一致することはありません。

マルチライン文字列

有効なマルチライン文字列は、次の条件のいずれかが true である場合にシンプルと見なされます。

- マルチライン文字列が空です。
- マルチライン文字列は空ではなく、以下のすべての条件が true です。
  - すべてのライン文字列はシンプルです。
  - マルチライン文字列の 2 つのライン文字列は、2 つのライン文字列の境界点であるポイントを除いて、交差しません。

空のライン文字列のみで構成される空ではないマルチライン文字列は、空であると見なされます。

マルチライン文字列内の空のライン文字列は、そのシンプルさに影響しません。

マルチライン文字列内の閉じたライン文字列は、マルチライン文字列内の他のライン文字列と交差できません。

マルチライン文字列には、連続するポイントが重複するライン文字列を含めることができません。

Multipolygon

有効なマルチポリゴンは重複する連続ポイントが含まれていない場合、単純であると見なされます。

ジオメトリコレクション

有効なジオメトリコレクションは、次の条件のいずれかが true である場合にシンプルと見なされます。

- ジオメトリコレクションは空です。つまり、ジオメトリは含まれていません。
- 空でないジオメトリコレクション内のすべてのジオメトリは単純です。

この定義は、再帰的にではありますが、ネストされたジオメトリコレクションにも適用されます。

H3

H3 は階層型の地理空間インデックスグリッドシステムで、空間座標を平方メートルの解像度までインデックス化できます。インデックス化されたデータは、異なるデータセット間で結合し、さまざまな精度で集約できます。H3 では、最近傍データ、最短経路、勾配平滑化など、グリッドに基づくさまざまなアルゴリズムと最適化が可能になります。H3 インデックスは、六角形または五角形にできるセルを指します。スペースは解像度に応じて階層的に分割されます。H3 は 0 から 15 までの 16 の解像度をサポートしています。`0` が最も粗く、`15` が最も細かいです。

Amazon Redshift では次の H3 空間関数を提供しています。

- H3_FromLongLat
- H3_FromPoint
- H3_Polyfill

Amazon Redshift で空間データを使用する際の考慮事項

Amazon Redshift で空間データを使用する際の考慮事項は以下のとおりです。

- GEOMETRY あるいは GEOGRAPHY オブジェクトの最大サイズは、1,048,447 バイトです。
- Amazon Redshift Spectrum では、空間データをネイティブにサポートしていません。したがって、GEOMETRY もしくは GEOGRAPHY の列がある外部テーブルを作成または修正することはできません。
- Python のユーザー定義関数 (UDF) のデータでは、GEOMETRY と GEOGRAPHY のデータ型はサポートされません。
- Amazon Redshift テーブルでは、GEOMETRY や GEOGRAPHY 列をソートキーとして、あるいは分散キーとして使用することはできません。
- SQL ORDER BY、GROUP BY、または DISTINCT 句の中で、GEOMETRY や GEOGRAPHY 列を使用することはできません。
- GEOMETRY と GEOGRAPHY 列が使用できない SQL 関数は多数あります。
- GEOMETRY および GEOGRAPHY 列に関しては、すべての形式で UNLOAD オペレーションを実行できる訳ではありません。テキストまたはコンマ区切り値 (CSV) ファイルに対しては、GEOMETRY あるいは GEOGRAPHY の列を UNLOAD することが可能です。この処理により、GEOMETRY あるいは GEOGRAPHY のデータは 16 進数の EWKB 形式で書き込まれます。EWKB データのサイズが 4 MB 以上の場合、後でテーブルにデータをロードできなくなるため、警告が表示されます。
- GEOMETRY と GEOGRAPHY のデータでサポートされている圧縮エンコードは RAW です。
- JDBC または ODBC ドライバを使用する際は、カスタム型マッピングを使います。この場合、クライアントアプリケーションには、ResultSet オブジェクトのどのパラメータが、GEOMETRY もしくは GEOGRAPHY のオブジェクトであるかに関する情報が必要です。ResultSetMetadata のオペレーションにより、タイプ VARCHAR が返されます。
- SHAPEFILE から地理的データをコピーするには、まず GEOMETRY 列の中にデータを取り込んだ後、対象のオブジェクトを GEOGRAPHY オブジェクトにキャストします。

以下に示す非空間関数では、GEOMETRY と GEOGRAPHY 型の入力または GEOMETRY と GEOGRAPHY 型の列に対応することができます。

- 集計関数 COUNT
- 条件式 COALESCE および NVL
- CASE 式
- GEOMETRY および GEOGRAPHY でのデフォルトのエンコーディングは RAW です。詳細については、「圧縮エンコード」を参照してください。

Amazon Redshift での横串検索を使用したデータのクエリの実行

Amazon Redshift で横串検索を使用することにより、オペレーションデータベース、データウェアハウス、データレイク全体のデータをクエリして分析することができます。横串検索機能を使用すると、外部データベースのライブデータの Amazon Redshift からのクエリを、Amazon Redshift と Amazon S3 環境全体のクエリと統合できます。横串検索は、Amazon RDS for PostgreSQL、Amazon Aurora PostgreSQL 互換エディション、Amazon RDS for MySQL、および Amazon Aurora MySQL 互換エディション の外部データベースと連携することができます。

フェデレーテッドクエリにより、ビジネスインテリジェンス (BI) およびレポートアプリケーションの一部としてライブデータを組み込むことができます。たとえば、Amazon Redshift へのデータの取り込みを容易にするために、横串検索を使用して次の操作を実行できます。

- 運用データベースを直接照会します。
- 変換をすばやく適用します。
- 複雑な抽出、変換、ロード (ETL) パイプラインを必要とせずに、ターゲットテーブルにデータをロードします。

ネットワーク上でのデータ移動を減らし、パフォーマンスを向上させるために、Amazon Redshift は横串検索の計算の一部をリモートオペレーションデータベースに直接分散します。また、Amazon Redshift は並列処理能力を使用して、必要に応じてこれらのクエリの実行をサポートします。

横串検索を実行すると、Amazon Redshift は、まずリーダーノードから RDS または Aurora DB クラスター DB インスタンスへのクライアント接続を確立し、テーブルメタデータを取得します。コンピューティングノードから、Amazon Redshift は述語がプッシュダウンされたサブクエリを発行し、結果の行を取得します。次に、Amazon Redshift は結果の行をコンピューティングノード間で分散してさらに処理します。

Amazon Aurora PostgreSQL データベースまたは Amazon RDS for PostgreSQL データベースに送信されるクエリの詳細は、システムビュー SVL_FEDERATED_QUERY にログ記録されます 。

トピック

- PostgreSQL への横串検索を使用した開始方法
- AWS CloudFormation での PostgreSQL への横串検索の使用開始
- MySQL への横串検索の使用を開始する

- フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成
- 横串検索の使用例
- Amazon Redshift とサポートされている PostgreSQL および MySQL データベース間のデータ型の相違点
- Amazon Redshift でフェデレーティッドデータにアクセスする際の考慮事項

PostgreSQL への横串検索を使用した開始方法

フェデレーテッドクエリを作成するには、次の一般的なアプローチに従います。

1. Amazon Redshift クラスターから Amazon RDS または Aurora PostgreSQL DB インスタンスへの接続を設定します。

   これを行うには、RDS PostgreSQL または Aurora PostgreSQL DB インスタンスが Amazon Redshift クラスターからの接続を受け入れることができることを確認します。Amazon Redshift クラスターと Amazon RDS または Aurora PostgreSQL インスタンスは、同じ Virtual Private Cloud (VPC) とサブネットグループに含めることをお勧めします。これにより、Amazon Redshift クラスターのセキュリティグループを RDS または Aurora PostgreSQL DB インスタンスのセキュリティグループのインバウンドルールに追加できます。

   Amazon Redshift が RDS または Aurora PostgreSQL インスタンスへの接続を可能にする VPC ピアリングやその他のネットワーキングを設定することもできます。VPC ネットワークの詳細については、以下を参照してください。

   - Amazon VPCピアリングガイドのVPC ピアリングア機能とは?を参照してください。
   - Amazon RDS ユーザーガイドのVPC での DB インスタンスの操作

   > Note
   >
   > 拡張 VPC ルーティングを有効にする必要がある場合があります。例えば、Amazon Redshift クラスターが RDS または Aurora PostgreSQL インスタンスとは異なる VPC にある場合や、両者は同じ VPC にあるが、ルートが拡張 VPC を必要とする場合です。そうしないと、横串検索の実行時にタイムアウトエラーが発生することがあります。

2. AWS Secrets Manager で、RDS PostgreSQL と Aurora PostgreSQL データベース用のシークレットを設定します。次に、AWS Identity and Access Management (IAM) アクセスポリシーとロールのシークレットを参照します。詳細については、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。

> **Note**
>
> クラスターで拡張 VPC ルーティングを使用している場合は、AWS Secrets Manager のインターフェイス VPC エンドポイントを設定する必要があります。これは、Amazon Redshift クラスターの VPC とサブネットが、パブリックな AWS Secrets Manager エンドポイントにアクセスできない場合に必要です。VPC インターフェイスエンドポイントを使用する場合、VPC 内の Amazon Redshift クラスターと AWS Secrets Manager の間の通信は、VPC からエンドポイントインターフェイスにプライベートにルーティングされます。詳細については、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントの作成」を参照してください。

3. 以前に作成した IAM ロールを Amazon Redshift クラスターに適用します。詳細については、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。
4. 外部スキーマを使用して RDS PostgreSQL と Aurora PostgreSQL データベースに接続します。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。フェデレーテッドクエリの使用方法の例については、「横串検索の使用例」を参照してください。
5. RDS PostgreSQL と Aurora PostgreSQL データベースを参照する外部スキーマを参照する SQL クエリを実行します。

AWS CloudFormation での PostgreSQL への横串検索の使用開始

横串検索を使用して、オペレーショナルデータベース間でクエリを実行できます。この入門ガイドでは、サンプルの AWS CloudFormation スタック を使用してセットアップを自動化し、Amazon Redshift クラスターから Aurora PostgreSQL サーバーレスデータベースへの横串検索を有効にします。SQL ステートメントを実行してリソースをプロビジョニングする必要はなく、検索をすばやく起動して実行できます。

このスタックでは、サンプルデータを格納したテーブルを含む Aurora PostgreSQL インスタンスを参照する、外部スキーマを作成します。外部スキーマ内にあるテーブルには、Redshift クラスターからクエリできます。

これとは別に、CloudFormation を使用せずに SQL ステートメントを実行し、外部スキーマを設定しながら横串検索を開始する場合は、「PostgreSQL への横串検索を使用した開始方法」を参照してください。

横串検索のために CloudFormation スタックを実行する前に、Data API が有効化された Amazon Aurora PostgreSQL 互換エディションのサーバーレスデータベースを使用していることを、確認してください。Data API は、データベースのプロパティで有効化できます。この設定が見つからない場合は、Aurora PostgreSQL のサーバーレスインスタンスを実行していることを再確認してください。また、RA3 ノードを使用する Amazon Redshift クラスターがあることを確認します。Redshift クラスターと Aurora PostgreSQL インスタンスは、同じ Virtual Private Cloud (VPC) とサブネットグループに含めることをお勧めします。これにより、Amazon Redshift クラスターのセキュリティグループを、Aurora PostgreSQL データベースインスタンスのセキュリティグループのインバウンドルールに追加できます。

Amazon Redshift クラスターの設定の開始方法については、「Amazon Redshift のプロビジョニングされたデータウェアハウス」を参照してください。CloudFormation を使用したリソースのセットアップの詳細については、「What is AWS CloudFormation?」を参照してください。Aurora DB クラスターデータベースをセットアップする方法の詳細については、「Aurora DB クラスター Serverless v1 DB クラスターの作成」を参照してください。

Redshift での横串検索用の CloudFormation スタックの起動

次の手順を使用して、Amazon Redshift 用に CloudFormation スタックを起動して、横串検索を有効にします。作業開始の前に、Amazon Redshift クラスターとサーバーレス Aurora PostgreSQL インスタンスがセットアップされていることを確認してください。

横串検索のために CloudFormation スタックを起動するには

1. AWS Management Console の [Launch CFN stack] (CFN スタックを起動) をクリックし、CloudFormation サービスを起動します。

   プロンプトが表示されたら、サインインします。

   Amazon S3 に保存されている CloudFormation テンプレートファイルを参照しながら、スタック作成プロセスが開始されます。CloudFormation テンプレートは、JSON 形式で作成されたテキストファイルで、スタックを構成する AWS リソースに関する宣言が記述されています。
2. [Next] (次へ) をクリックして、スタックの詳細を入力します。
3. [Parameters] (パラメータ) で、クラスターについて以下のように入力します。

   - Amazon Redshift のクラスター名 (例えば、**ra3-consumer-cluster**)
   - 特定のデータベース名 (例えば、**dev**)
   - データベースのユーザー名 (例えば、**consumeruser**)

さらに、ユーザー、データベース名、ポート、エンドポイントなど、Aurora DB クラスターデータベースのパラメータも入力します。スタックは複数のデータベースオブジェクトを作成するため、クラスターとサーバーレスデータベースはテスト用のものを使用することをお勧めします。

[Next] を選択します。

スタックに関するオプションが表示されます。

4. [Next] (次へ) をクリックして、デフォルト設定を受け入れます。
5. [機能] で、[AWS CloudFormation によって IAM リソースが作成される場合があることを承認します] を選択します。
6. [スタックの作成] を選択します。

[スタックの作成] を選択します。CloudFormation は、10 分間程度でテンプレートリソースをプロビジョニングし、外部スキーマを作成します。

スタックの作成中にエラーが発生した場合は、以下の手順を実行します。

- エラーの解決に役立つ情報については、CloudFormation の [Events] (イベント) タブを開きます。
- Redshift クラスターの名前、データベース名、およびデータベースのユーザー名が正しく入力されていることを確認してください。また、Aurora PostgreSQL インスタンスのパラメータも確認します。
- クラスターに RA3 ノードがあることを確認します。
- 同じサブネットとセキュリティグループ内に、データベースと Redshift クラスターが置かれていることを確認します。

外部スキーマからのデータのクエリ

以下の手順を使用するには、説明で示されているクラスターとデータベースでクエリを実行するための適切なアクセス許可が必要です。

横串検索を使用して外部データベースに対しクエリを実行するには

1. Redshift のクエリエディタなどのクライアントツールを使用して、スタックを作成した際に入力した Redshift データベースに接続します。
2. スタックによって作成された外部スキーマをクエリします。

```
select * from svv_external_schemas;
```

SVV_EXTERNAL_SCHEMAS ビューで、使用可能な外部スキーマに関する情報が返されます。ここでは、スタックによって作成された外部スキーマ (myfederated_schema) が返されます。設定をしておくことで、他の外部スキーマを返させることもできます。このビューでは、対象スキーマに関連付けられたデータベースも返します。データベースは、スタックの作成時に入力した Aurora DB クラスターデータベースです。スタックは、Aurora DB クラスターデータベースにテーブル (category) と別のテーブル (sales) を追加します。

3. Aurora PostgreSQL データベースを参照する外部スキーマのテーブルに対して、SQL クエリを実行します。以下に、クエリの例を示します。

```
SELECT count(*) FROM myfederated_schema.category;
```

category テーブルは複数のレコードを返します。また、salesテーブルからレコードを返すこともできます。

```
SELECT count(*) FROM myfederated_schema.sales;
```

その他の例については、「横串検索の使用例」を参照してください。

MySQL への横串検索の使用を開始する

MySQL データベースへの横串検索を作成するには、次の一般的なアプローチに従います。

1. Amazon Redshift クラスターから Amazon RDS または Aurora MySQL DB インスタンスへの接続を設定します。

   これを行うには、RDS MySQL または Aurora MySQL DB インスタンスが Amazon Redshift クラスターからの接続を受け入れることができることを確認します。Amazon Redshift クラスターと Amazon RDS または Aurora MySQL インスタンスは、同じ Virtual Private Cloud (VPC) とサブネットグループに含めることをお勧めします。これにより、Amazon Redshift クラスターのセキュリティグループを RDS または Aurora MySQL DB インスタンスのセキュリティグループのインバウンドルールに追加できます。

Amazon Redshift が RDS または Aurora MySQL インスタンスへの接続を可能にする VPC ピアリングやその他のネットワーキングを設定することもできます。VPC ネットワークの詳細については、以下を参照してください。

- Amazon VPCピアリングガイドのVPC ピアリングア機能とは?を参照してください。
- Amazon RDS ユーザーガイドのVPC での DB インスタンスの操作

> **Note**
>
> Amazon Redshift クラスターが RDS または Aurora MySQL インスタンスとは異なる VPC にある場合は、拡張 VPC ルーティングを有効にします。そうしないと、横串検索の実行時にタイムアウトエラーが発生することがあります。

2. AWS Secrets Manager で、RDS MySQL と Aurora MySQL データベース用のシークレットを設定します。次に、AWS Identity and Access Management (IAM) アクセスポリシーとロールのシークレットを参照します。詳細については、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。

> **Note**
>
> クラスターで拡張 VPC ルーティングを使用している場合は、AWS Secrets Manager のインターフェイス VPC エンドポイントを設定する必要があります。これは、Amazon Redshift クラスターの VPC とサブネットが、パブリックな AWS Secrets Manager エンドポイントにアクセスできない場合に必要です。VPC インターフェイスエンドポイントを使用する場合、VPC 内の Amazon Redshift クラスターと AWS Secrets Manager の間の通信は、VPC からエンドポイントインターフェイスにプライベートにルーティングされます。詳細については、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントの作成」を参照してください。

3. 以前に作成した IAM ロールを Amazon Redshift クラスターに適用します。詳細については、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。
4. 外部スキーマを使用して RDS MySQL と Aurora MySQL データベースに接続します。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。横串検索の使用方法の例については、「MySQL での横串検索の使用例」を参照してください。
5. RDS MySQL と Aurora MySQL データベースを参照する外部スキーマを参照する SQL クエリを実行します。

フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成

以下のステップは、フェデレーテッドクエリで使用するシークレットと IAM ロールを作成する方法を示しています。

前提条件

横串検索で使用するシークレットと IAM ロールを作成するには、次の前提条件を満たす必要があります。

- ユーザー名とパスワード認証を使用する RDS PostgreSQL、Aurora PostgreSQL DB インスタンス、RDS MySQL、または Aurora MySQL DB インスタンス。
- 横串検索をサポートするクラスターメンテナンスバージョンを持つ Amazon Redshift クラスター。

AWS Secrets Manager でシークレット (ユーザー名とパスワード) を作成するには

1. RDS または Aurora DB クラスターインスタンスを所有するアカウントで Secrets Manager コンソールにサインインします。
2. [新しいシークレットを保存] を選択します。
3. [RDS データベースの認証情報] タイルを選択します。[ユーザー名] と [パスワード] に、インスタンスの値を入力します。[暗号化キー] の値を確認または選択します。次に、シークレットがアクセスする RDS データベースを選択します。

   > **Note**
   >
   > デフォルトの暗号化キー (`DefaultEncryptionKey`) を使用することをお勧めします。カスタム暗号化キーを使用する場合、シークレットにアクセスするために使用される IAM ロールをキーユーザーとして追加する必要があります。

4. シークレットの名前を入力し、デフォルトの選択肢を使用して作成ステップを続行して、[Store (保存)] を選択します。
5. シークレットを表示し、シークレットを識別するために作成した [シークレット ARN] 値をメモします。

シークレットを使用してセキュリティポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. 次のような JSON を使用してポリシーを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

   シークレットを取得するには、リストと読み取りアクションが必要です。作成した特定のシークレットにリソースを制限することをお勧めします。これを行うには、シークレットの Amazon リソースネーム (ARN) を使用してリソースを制限します。IAM コンソールのビジュアルエディタを使用して、アクセス許可とリソースを指定することもできます。
3. ポリシーに名前を付けて、作成を完了します。
4. [IAM ロール] に移動します。
5. [Redshift - カスタマイズ可能] の IAM ロールを作成します。

6. 先ほど作成した IAM ポリシーを既存の IAM ロールにアタッチするか、新しい IAM ロールを作成してポリシーをアタッチします。
7. IAM ロールの [Trust relationships (信頼関係)] タブで、信頼エンティティ `redshift.amazonaws.com` が含まれていることを確認します。
8. 作成したロール ARN を書き留めます。この ARN はシークレットにアクセスできます。

IAM ロールを Amazon Redshift クラスターにアタッチするには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [クラスター] を選択します。現在のAWSリージョンにあるアカウントのクラスターがリストされています。
3. リストでクラスター名を選択して、クラスターの詳細を表示します。
4. [アクション] で、[IAM ロールの管理] を選択します。[IAM ロールの管理] ページが表示されます。
5. IAM ロールをクラスターに追加します。

横串検索の使用例

次の例では、横串検索の実行方法を示しています。Amazon Redshift データベースに接続されている SQL クライアントを使用して SQL を実行します。

PostgreSQL での横串検索の使用例

次の例は、Amazon Redshift データベース、Aurora PostgreSQL データベース、および Amazon S3 を参照するフェデレーションクエリを設定する方法を示しています。この例は、横串検索の機能を示しています。独自の環境で実行するには、環境に合わせて変更します。これを行うための前提条件については、「PostgreSQL への横串検索を使用した開始方法」を参照してください。

Aurora PostgreSQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA apg
FROM POSTGRES
DATABASE 'database-1' SCHEMA 'myschema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
```

```
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

Amazon S3 を参照する別の外部スキーマを作成します。これは Amazon Redshift Spectrum を使用します。さらに、スキーマを使用するアクセス権を public に付与します。

```
CREATE EXTERNAL SCHEMA s3
FROM DATA CATALOG
DATABASE 'default' REGION 'us-west-2'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-S3';

GRANT USAGE ON SCHEMA s3 TO public;
```

Amazon Redshift テーブル内の行数を表示します。

```
SELECT count(*) FROM public.lineitem;

  count
----------
25075099
```

Aurora PostgreSQL テーブル内の行数を表示します。

```
SELECT count(*) FROM apg.lineitem;

count
-------
11760
```

Amazon S3 の行数を表示します。

```
SELECT count(*) FROM s3.lineitem_1t_part;

   count
------------
6144008876
```

Amazon Redshift、Aurora PostgreSQL、および Amazon S3 からテーブルのビューを作成します。このビューは、フェデレーテッドクエリの実行に使用されます。

```
CREATE VIEW lineitem_all AS
```

```
 SELECT
l_orderkey,l_partkey,l_suppkey,l_linenumber,l_quantity,l_extendedprice,l_discount,l_tax,l_retu
        l_shipdate::date,l_commitdate::date,l_receiptdate::date,
l_shipinstruct ,l_shipmode,l_comment
 FROM s3.lineitem_1t_part
 UNION ALL SELECT * FROM public.lineitem
 UNION ALL SELECT * FROM apg.lineitem
    with no schema binding;
```

ビュー `lineitem_all` 内の行数を述語とともに表示し、結果を制限します。

```
SELECT count(*) from lineitem_all WHERE l_quantity = 10;

   count
-----------
123373836
```

ある商品の毎年 1 月の売上高がいくらかを調べます。

```
SELECT extract(year from l_shipdate) as year,
       extract(month from l_shipdate) as month,
       count(*) as orders
FROM lineitem_all
WHERE extract(month from l_shipdate) = 1
AND l_quantity < 2
GROUP BY 1,2
ORDER BY 1,2;

 year | month | orders
------+-------+---------
 1992 |     1 |  196019
 1993 |     1 | 1582034
 1994 |     1 | 1583181
 1995 |     1 | 1583919
 1996 |     1 | 1583622
 1997 |     1 | 1586541
 1998 |     1 | 1583198
 2016 |     1 |   15542
 2017 |     1 |   15414
 2018 |     1 |   15527
 2019 |     1 |     151
```

大文字と小文字が混在した名前の使用例

データベース、スキーマ、テーブル、または列の大文字と小文字が混在する名前を持つ、サポート対象の PostgreSQL リモートデータベースをクエリするには、enable_case_sensitive_identifier を true に設定します。このセッションのパラメータの詳細については、「enable_case_sensitive_identifier」を参照してください。

```
SET enable_case_sensitive_identifier TO TRUE;
```

通常、データベース名とスキーマ名は小文字です。次の例では、データベースとスキーマに小文字の名前、テーブルと列に大文字と小文字が混在する名前を持つ、サポート対象の PostgreSQL リモートデータベースに接続する方法を示しています。

小文字のデータベース名 (dblower) と小文字のスキーマ名 (schemalower) を持つ Aurora PostgreSQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA apg_lower
FROM POSTGRES
DATABASE 'dblower' SCHEMA 'schemalower'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

クエリが実行されるセッションで、enable_case_sensitive_identifier を true に設定します。

```
SET enable_case_sensitive_identifier TO TRUE;
```

横串検索を実行して、PostgreSQL データベースからすべてのデータを選択します。テーブル (MixedCaseTab) と列 (MixedCaseName) には大文字と小文字が混在した名前を使っています。結果は 1 行 (Harry) です。

```
select * from apg_lower."MixedCaseTab";
```

```

 MixedCaseName
-------
```

```
 Harry
```

次の例では、データベース、スキーマ、テーブル、および列に対して大文字と小文字が混在した名前を持つ、サポート対象の PostgreSQL リモートデータベースに接続する方法を示しています。

外部スキーマを作成する前に、`enable_case_sensitive_identifier` を `true` に設定してください。外部スキーマを作成する前に `enable_case_sensitive_identifier` が `true` に設定されていない場合、データベースが存在しないというエラーが発生します。

大文字と小文字が混在するデータベース名 (`UpperDB`) とスキーマ名 (`UpperSchema`) を持つ Aurora PostgreSQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA apg_upper
FROM POSTGRES
DATABASE 'UpperDB' SCHEMA 'UpperSchema'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

横串検索を実行して、PostgreSQL データベースからすべてのデータを選択します。テーブル (`MixedCaseTab`) と列 (`MixedCaseName`) には大文字と小文字が混在した名前を使っています。結果は 1 行 (`Harry`) です。

```
select * from apg_upper."MixedCaseTab";
```

```

 MixedCaseName
-------
 Harry
```

MySQL での横串検索の使用例

次の例は、Aurora MySQL データベースを参照する横串検索を設定する方法を示しています。この例は、フェデレーテッドクエリの動作を示しています。独自の環境で実行するには、環境に合わせて変更します。これを行うための前提条件については、「MySQL への横串検索の使用を開始する」を参照してください。

この例は、以下の前提条件によって異なります:

- Aurora MySQL データベースのSecrets Manager で設定されたシークレットです。このシークレットは、IAM アクセスポリシーとロールで参照されます。詳細については、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。
- Amazon Redshift と Aurora MySQL をリンクするように設定されたセキュリティグループ。

Aurora MySQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA amysql
FROM MYSQL
DATABASE 'functional'
URI 'endpoint to remote hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-SecretsManager-RO'
SECRET_ARN 'arn:aws:secretsmanager:us-west-2:123456789012:secret:federation/test/
dataplane-apg-creds-YbVKQw';
```

Aurora MySQL テーブルの SQL 選択の例を実行して、Aurora MySQL の employees テーブルから 1 行を表示します。

```
SELECT level FROM amysql.employees LIMIT 1;

 level
-------
     8
```

Amazon Redshift とサポートされている PostgreSQL および MySQL データベース間のデータ型の相違点

次の表は、対応する Amazon RDS PostgreSQL または Aurora PostgreSQL データ型への Amazon Redshift データ型のマッピングを示しています。

Amazon Redshift のデータ型	RDS PostgreSQL または Aurora PostgreSQL のデータ型	説明
SMALLINT	SMALLINT	符号付き 2 バイト整数

Amazon Redshift のデータ型	RDS PostgreSQL または Aurora PostgreSQL のデータ型	説明
INTEGER	INTEGER	符号付き 4 バイト整数
BIGINT	BIGINT	符号付き 8 バイト整数
DECIMAL	DECIMAL	精度の選択が可能な真数
REAL	REAL	単精度浮動小数点数
DOUBLE PRECISION	DOUBLE PRECISION	倍精度浮動小数点数
BOOLEAN	BOOLEAN	論理ブール演算型 (true/false)
CHAR	CHAR	固定長のキャラクタ文字列
VARCHAR	VARCHAR	ユーザーによって定義された制限を持つ可変長キャラクタ文字列
DATE	DATE	カレンダー日付 (年、月、日)
TIMESTAMP	TIMESTAMP	日付と時刻 (タイムゾーンなし)
TIMESTAMPTZ	TIMESTAMPTZ	日付と時刻 (タイムゾーンあり)
GEOMETRY	PostGIS GEOMETRY	空間データ

次の RDS PostgreSQL および Aurora PostgreSQL データ型は、Amazon Redshift で VARCHAR (64K) に変換されます。

- JSON, JSONB
- 配列
- BIT、BIT VARYING
- BYTEA
- コンポジット型
- 日付と時刻のタイプは、INTERVAL、TIME、TIME WITH TIMEZONE です
- 列挙型
- 通貨型
- ネットワークアドレス型
- 数値型 SERIAL、BIGSERIAL、SMALLSERIAL、MONEY
- オブジェクト識別子型
- pg_lsn type
- 疑似タイプ
- 範囲型
- テキスト検索型
- TXID_SNAPSHOT
- UUID
- XML 型

次の表は、対応する Amazon RDS MySQL または Aurora MySQL データ型への Amazon Redshift データ型のマッピングを示しています。

Amazon Redshift のデータ型	RDS MySQL または Aurora MySQL のデータ型	説明
BOOLEAN	TINYINT(1)	論理ブール演算型 (true または false)
SMALLINT	TINYINT (UNSIGNED)	符号付き 2 バイト整数

Amazon Redshift のデータ型	RDS MySQL または Aurora MySQL のデータ型	説明
SMALLINT	SMALLINT	符号付き 2 バイト整数
INTEGER	SMALLINT UNSIGNED	符号付き 4 バイト整数
INTEGER	MEDIUMINT (UNSIGNED)	符号付き 4 バイト整数
INTEGER	INT	符号付き 4 バイト整数
BIGINT	INT UNSIGNED	符号付き 8 バイト整数
BIGINT	BIGINT	符号付き 8 バイト整数
DECIMAL	BIGINT UNSIGNED	精度の選択が可能な真数
DECIMAL	DECIMAL(M,D)	精度の選択が可能な真数
REAL	FLOAT	単精度浮動小数点数
DOUBLE PRECISION	DOUBLE	倍精度浮動小数点数
CHAR	CHAR	固定長のキャラクタ文字列
VARCHAR	VARCHAR	ユーザーによって定義された制限を持つ可変長キャラクタ文字列

Amazon Redshift のデータ型	RDS MySQL または Aurora MySQL のデータ型	説明
DATE	DATE	カレンダー日付 (年、月、日)
TIME	TIME	時間（タイムゾーンなし）
TIMESTAMP	TIMESTAMP	日付と時刻 (タイムゾーンなし)
TIMESTAMP	DATETIME	時間（タイムゾーンなし）
VARCHAR(4)	YEAR	年を表す可変長文字

TIME データが範囲外の場合 (00:00:00 ~ 24:00:00)、エラーが発生します。

次の RDS MySQL および Aurora MySQL データ型は、Amazon Redshift で VARCHAR (64K) に変換されます。

- BIT
- BINARY
- VARBINARY
- TINYBLOB、BLOB、MEDIUMBLOB、LONGBLOB
- TINYTEXT、TEXT、MEDIUMTEXT、LONGTEXT
- ENUM
- SET
- SPATIAL

Amazon Redshift でフェデレーティッドデータにアクセスする際の考慮事項

一部の Amazon Redshift 機能は、フェデレーテッドデータへのアクセスをサポートしていません。関連する制限事項と考慮事項については、以下を参照してください。

Amazon Redshift で横串検索を使用する場合の制約事項と考慮事項を次に示します。

- フェデレーテッドクエリは、外部データソースへの読み取りアクセスをサポートします。外部データソースにデータベースオブジェクトを書き込んだり、作成したりすることはできません。
- 場合によっては、Amazon Redshift とは異なる AWS リージョンにある、Amazon RDS または Aurora DB クラスターデータベースにアクセスすることがあります。このような場合に AWS リージョン間でデータを転送すると、通常はネットワークレイテンシーと料金が発生します。Amazon Redshift クラスターと同じ AWS リージョン内のローカルエンドポイントでの、Aurora グローバルデータベースの使用をお勧めします。Aurora グローバルデータベースは、専用インフラストラクチャを使用して、任意の 2 つの AWS リージョン間でのストレージベースのレプリケーションを行います。この際の平均的なレイテンシーは 1 秒未満です。
- Amazon RDS または Aurora DB クラスターにアクセスするコストを考慮してください。例えば、この機能を使用して Aurora DB クラスターにアクセスする場合、Aurora DB クラスターの料金は IOPS に基づきます。
- 横串検索では、RDS または Aurora DB クラスターから Amazon Redshift にアクセスすることはできません。
- 横串検索は、Amazon RDS と Amazon RDS の両方または Aurora DB クラスターが使用可能な AWS リージョンでのみ利用できます。
- フェデレーテッドクエリは現在 `ALTER SCHEMA` をサポートしていません。スキーマを変更するには、 `DROP` を使用してから `CREATE EXTERNAL SCHEMA` を使用します。
- フェデレーテッドクエリは、同時実行スケーリングでは機能しません。
- 横串検索は、現在、PostgreSQL の外部データラッパーを介したアクセスをサポートしていません。
- RDS MySQL または Aurora MySQL への横串検索は、READ COMMITTED レベルでのトランザクション分離をサポートします。
- 指定しない場合、Amazon Redshift はポート 3306 で RDS for MySQL または Aurora MySQL に接続します。MySQL の外部スキーマを作成する前に、MySQL ポート番号を確認してください。
- 指定しない場合、Amazon Redshift はポート 5432 で RDS PostgreSQL または Aurora PostgreSQL に接続します。PostgreSQL の外部スキーマを作成する前に、PostgreSQL ポート番号を確認します。
- MySQL からTIMESTAMPおよびDATEデータ型を取得する場合、ゼロ値はNULLとして扱われます。
- Aurora DB クラスターデータベースのリーダーエンドポイントを使用すると、「無効なスナップショット」エラーが発生する可能性があります。これは、以下のいずれかの方法で回避できます。

- 特定の Aurora DB クラスターインスタンスエンドポイントを (Aurora クラスターのクラスターエンドポイントの代わりに) 使用してください。この方法では、PostgreSQL データベースからの結果に対して REPEATABLE READ トランザクション分離を使用します。
- Aurora DB クラスターのリーダーエンドポイントを使用し、セッションで `pg_federation_repeatable_read` を false に設定します。この方法では、PostgreSQL データベースからの結果に対して READ COMMITTED トランザクション分離を使用します。Aurora DB クラスターのリーダーエンドポイントの詳細については、「Amazon Aurora ユーザーガイド」の「Aurora DB クラスターエンドポイントのタイプ」を参照してください。`pg_federation_repeatable_read` の詳細については、「pg_federation_repeatable_read」を参照してください。

PostgreSQL データベースへの横串検索を使用する場合のトランザクションに関する考慮事項を次に示します。

- クエリがフェデレーテッドテーブルで構成されている場合、リーダーノードはリモートデータベースで READ ONLY REPEATABLE READ トランザクションを開始します。このトランザクションは、Amazon Redshift トランザクションの期間中残ります。
- リーダーノードは `pg_export_snapshot` を呼び出してリモートデータベースのスナップショットを作成し、影響を受けるテーブルに対して読み取りロックを行います。
- コンピューティングノードはトランザクションを開始し、リーダーノードで作成されたスナップショットを使用してリモートデータベースにクエリを発行します。

フェデレーションデータベースのサポート対象バージョン

Amazon Redshift 外部スキーマは、外部 RDS PostgreSQL または Aurora PostgreSQL のデータベースを参照できます。その場合、次の制限が適用されます。

- Aurora DB クラスターを参照する外部スキーマを作成する場合、Aurora PostgreSQL データベースはバージョン 9.6 以降である必要があります。
- Amazon RDS を参照する外部スキーマを作成する場合、Amazon RDS PostgreSQL データベースはバージョン 9.6 以降である必要があります。

Amazon Redshift 外部スキーマは、外部 RDS MySQL または Aurora MySQL のデータベースを参照できます。その場合、次の制限が適用されます。

- Aurora DB クラスターを参照する外部スキーマを作成する場合、Aurora MySQL データベースはバージョン 5.6 以降である必要があります。
- Amazon RDS を参照する外部スキーマを作成する場合、RDS MySQL データベースはバージョン 5.6 以降である必要があります。

Amazon Redshift Spectrum

このセクションでは、Redshift Spectrum を使用して Amazon S3 からデータを効率的に読み取る方法について説明します。

Amazon Redshift Spectrum を使用すると、効率的にクエリを実行し、Amazon Redshift テーブルにデータをロードすることなく、Amazon S3 のファイルから構造化および半構造化されたデータを取得できます。Redshift Spectrum クエリでは超並列処理を採用しており、大きなデータセットに対する処理が非常に高速で実行されます。処理の多くは Redshift Spectrum レイヤーで生じ、データの大部分が Amazon S3 に保持されます。複数のクラスターが Amazon S3 で同じデータセットを同時にクエリできます。クラスターごとにデータをコピーする必要はありません。

トピック

Amazon Redshift Spectrum の概要

このトピックでは、Redshift Spectrum を使用して Amazon S3 から効率的に読み取る方法の詳細について説明します。

Amazon Redshift Spectrum は、クラスターに依存しない専用の Amazon Redshift サーバー上にあります。Amazon Redshift は、述語フィルタリングや集計など、大量の演算を行う多くのタスクを Redshift Spectrum レイヤーにプッシュします。したがって、Redshift Spectrum クエリが使用するクラスタの処理容量は他のクエリよりもはるかに少なくなります。Redshift Spectrum はさらに、インテリジェントに拡張します。クエリの需要に基づいて、Redshift Spectrum は潜在的に数千のインスタンスを使用して超並列処理を活用できます。

Redshift Spectrum テーブルは、ファイルの構造を定義して外部データカタログ内のテーブルとして登録することで作成します。外部データカタログは、AWS Glue、Amazon Athena を持つデータカタログ、もしくはユーザー所有の Apache Hive メタストアを指定することができます。外部テーブルは、データ定義言語 (DDL) コマンドを使用して、または外部データカタログに接続するその他の任意のツールを使用して Amazon Redshift から作成および管理できます。外部データカタログへの変更は、ただちにすべての Amazon Redshift クラスターで利用できます。

オプションで、外部テーブルを 1 つ以上の列でパーティション化できます。外部テーブルの一部としてパーティションを定義すると、パフォーマンスが向上します。パフォーマンスが向上するのは、Amazon Redshift クエリオプティマイザがクエリのデータを含まないパーティションを削除するためです。

Spectrum テーブルのマテリアライズドビューによって、コストとパフォーマンスが大幅に向上します。詳細については、「Amazon Redshift Spectrum の外部データレイクテーブルのマテリアライズドビュー」を参照してください。

Redshift Spectrum テーブルを定義すると、他の Amazon Redshift テーブルと同じようにクエリを実行してテーブル結合できるようになります。Redshift Spectrum は、外部テーブルに対する更新オペレーションをサポートしていません。Redshift Spectrum テーブルを複数の Amazon Redshift クラスターに追加して、同じ AWS リージョン内の任意のクラスターから Amazon S3 の同じデータにクエリを実行できます。Amazon S3 データファイルを更新すると、データはすぐにあらゆる Amazon Redshift クラスターからクエリに利用可能になります。

アクセスする AWS Glue データカタログは、セキュリティを強化するために暗号化されている可能性があります。AWS Glue カタログが暗号化されている場合、AWS Glueカタログにアクセスするには AWS Glue の AWS Key Management Service (AWS KMS) キーが必要です。AWS Glue カタログの暗号化は、一部の AWS リージョンでは使用できません。サポートされている AWS リージョンのリストについては、AWS Glueデベロッパーガイドの「AWS Glue の暗号化と安全なアクセス」を参照してください。AWS Glue でのデータカタログ暗号化の詳細については、AWS Glueデベロッパーガイドの「AWS Glue データカタログの暗号化」を参照してください。

> **Note**
>
> PG_TABLE_DEF、STV_TBL_PERM、PG_CLASS、または information_schema など、標準の Amazon Redshift テーブルに使用したものと同じリソースを使用して Redshift Spectrum テーブルの詳細を表示することはできません。ビジネスインテリジェンスまたは分析ツールが Redshift Spectrum 外部テーブルを認識しない場合は、SVV_EXTERNAL_TABLESおよび SVV_EXTERNAL_COLUMNS にクエリを実行するようにアプリケーションを設定します。

Amazon Redshift Spectrum リージョン

そのリージョン固有のドキュメントで特に明記されていない限り、Amazon Redshift が提供されている AWS リージョン リージョンであれば、Redshift Spectrum を使用することができます。商用リージョンでの AWS リージョン の可用性については、「Amazon Web Services 全般のリファレンス」の「Redshift API」の「サービスエンドポイント」を参照してください。

Amazon Redshift Spectrum の制限事項

このトピックでは、Redshift Spectrum の使用に関する制限について説明します。

Redshift Spectrum を使用する際は、以下の考慮事項に留意してください。

- Amazon Redshift クラスターと Amazon S3 バケットは、同じ AWS リージョンに存在する必要があります。
- Redshift Spectrum は、プロビジョニングされたクラスターで拡張された VPC ルーティングをサポートしていません。Amazon S3 データにアクセスするには、追加の設定ステップを実行する必要がある場合があります。詳細については、「Amazon Redshift 管理ガイド」の「Redshift Spectrum and enhanced VPC routing」(Redshift Spectrum と拡張された VPC のルーティング) を参照してください。
- Redshift Spectrum は Amazon S3 のアクセスポイントのエイリアスをサポートしています。詳細については、Amazon Simple Storage Service ユーザーガイドのアクセスポイントにバケットスタイルのエイリアスを使用するを参照してください。ただし、Redshift Spectrum では、Amazon S3 アクセスポイントのエイリアスを持つ VPC をサポートしていません。詳細については、「Amazon Redshift 管理ガイド」の「Redshift Spectrum and enhanced VPC routing」(Redshift Spectrum と拡張された VPC のルーティング) を参照してください。
- 外部テーブルに対してオペレーションの更新または削除を実行することはできません。指定されたスキーマで新しい外部テーブルを作成するには、CREATE EXTERNAL TABLE を使用できます。CREATE EXTERNAL TABLE の詳細については、「CREATE EXTERNAL TABLE」

を参照してください。SELECT クエリの結果を外部カタログの既存の外部テーブルに挿入するには、INSERT (外部テーブル) を使用できます。INSERT (外部テーブル) の詳細については、「INSERT (外部テーブル)」を参照してください。

- AWS Lake Formationに対し有効になっている AWS Glue Data Catalog を使用する場合を除き、外部テーブルのユーザー権限を制御することはできません。代わりに、外部スキーマに対してアクセス権限の付与および取り消しを実行できます。AWS Lake Formation の使用方法の詳細については、「Redshift Spectrum と AWS Lake Formation」を参照してください。
- Redshift Spectrum クエリを実行するには、データベースユーザーがデータベースに一時テーブルを作成するアクセス権限を持っている必要があります。次の例では、データベース `spectrumdb` の一時アクセス権限を `spectrumusers` ユーザーグループに付与しています。

```
grant temp on database spectrumdb to group spectrumusers;
```

  詳細については、「GRANT」を参照してください。
- Athena データカタログまたは AWS Glue データカタログをメタデータストアとして使用する場合は、「Amazon Redshift 管理ガイド」の「クォータと制限」を参照してください。
- Redshift Spectrum は、Kerberos を使用する Amazon EMR をサポートしていません。

Amazon Redshift Spectrum の開始方法

このチュートリアルでは、Amazon Redshift Spectrum を使用して Amazon S3 上のファイルのデータに直接クエリを実行する方法を説明します。すでにクラスターと SQL クライアントがある場合、このチュートリアルは最小限のセットアップで完了することができます。

> **Note**
>
> Redshift Spectrum クエリには追加料金が発生します。このチュートリアルのサンプルクエリは通常料金で実行できます。料金の詳細については、「Amazon Redshift Spectrum 料金表」を参照してください。

前提条件

Redshift Spectrum を使用するには、SQL コマンドを実行するために、クラスターに接続された Amazon Redshift クラスターと SQL クライアントが必要です。クラスターと Amazon S3 内のデータファイルは同じ AWS リージョン に存在する必要があります。

Amazon Redshift クラスターの作成方法の詳細については、「Amazon Redshift 入門ガイド」の「Amazon Redshift でプロビジョニングされたデータウェアハウス」を参照してください。クラスターに接続する方法については、「Amazon Redshift 入門ガイド」の「Amazon Redshift データウェアハウスに接続する」を参照してください。

次の例では、サンプルデータは米国東部 (バージニア北部)リージョン (`us-east-1`) にあるため、`us-east-1` にあるクラスターも必要です。または、Amazon S3 を使用して、次のバケットとフォルダのデータオブジェクトをクラスターがある AWS リージョン のバケットにコピーできます。

- `s3://redshift-downloads/tickit/spectrum/customers/*`
- `s3://redshift-downloads/tickit/spectrum/sales_partition/*`
- `s3://redshift-downloads/tickit/spectrum/sales/*`
- `s3://redshift-downloads/tickit/spectrum/salesevent/*`

次のような Amazon S3 コマンドを実行して、米国東部 (バージニア北部) にあるサンプルデータを AWS リージョン にコピーします。コマンドを実行する前に、Amazon S3 のコピーコマンドに合ったバケットとフォルダをバケットに作成します。Amazon S3 のコピーコマンドの出力により、ファイルが希望する AWS リージョン の *bucket-name* にコピーされたことが確認されます。

```
aws s3 cp s3://redshift-downloads/tickit/spectrum/ s3://bucket-name/tickit/spectrum/ --copy-props none --recursive
```

AWS CloudFormation での Amazon Redshift Spectrum の開始方法

以下に示す手順の代わりに、Redshift Spectrum DataLake の AWS CloudFormation テンプレートにアクセスし、クエリの実行が可能な Amazon S3 バケットを含むスタックを作成することもできます。詳細については、「AWS CloudFormation スタックを起動して Amazon S3 内のデータにクエリを実行する」を参照してください。

ステップバイステップによる Redshift Spectrum の使用開始

Amazon Redshift Spectrum の使用を開始するには、次のステップに従います。

- ステップ 1. Amazon Redshift 用の IAM ロールを作成する
- ステップ 2: IAM ロールをクラスターと関連付ける

- ステップ 3: 外部スキーマと外部テーブルを作成する
- ステップ 4: Amazon S3 のデータにクエリを実行する

ステップ 1. Amazon Redshift 用の IAM ロールを作成する

クラスターには、AWS Glue または Amazon Athena に置かれている外部データカタログ、および Amazon S3 内のデータファイルにアクセスするための承認が必要です。この承認を提供するには、クラスターにアタッチされた AWS Identity and Access Management (IAM) ロールを参照します。ロールと Amazon Redshift の詳細な使用方法については、IAM ロールを使用して COPY および UNLOAD オペレーションを認可するを参照してください。

> **Note**
>
> 特定のケースでは、Athena データカタログを AWS Glue データカタログに移行することができます。これを実行するには、クラスターが AWS Glue 対応の AWS リージョンに存在していること、および Athena データカタログ内に Redshift Spectrum 外部テーブルを持っていることが条件となります。Redshift Spectrum で AWS Glue データカタログを使用するには、IAM ポリシーの変更が必要になる場合があります。詳細については、Athena ユーザーガイドの「AWS Glue データカタログへのアップグレード」を参照してください。

Amazon Redshift のロールを作成する場合は、次のいずれかのアプローチを選択します。

- Athena データカタログまたは AWS Glue データカタログで Redshift Spectrum を使用する場合は、Amazon Redshift 用の IAM ロールを作成するにはに示されている手順に従います。
- AWS Lake Formation に対し有効になっている AWS Glue Data Catalog で Redshift Spectrum を使用する場合は、以下に示されている手順に従います。
  - AWS Lake Formation に対し有効化されている AWS Glue Data Catalog を使用して、Amazon Redshift の IAM ロールを作成するには
  - Lake Formation データベースでクエリを実行するテーブルの SELECT 権限を付与するには

Amazon Redshift 用の IAM ロールを作成するには

1. IAM コンソールを開きます。
2. ナビゲーションペインで [ロール] を選択します。
3. [Create role] を選択します。

4. AWS サービス を信頼されたエンティティとして選択し、次に Redshift をユースケースとして選択します。
5. [他の AWS のサービス のユースケース] で、[Redshift - カスタマイズ可能]、[次へ] の順に選択します。
6. [アクセス許可ポリシーをアタッチする] ページが表示されます。`AmazonS3ReadOnlyAccess` および `AWSGlueConsoleFullAccess` を選択します (AWS Glue データカタログを使用する場合)。または、`AmazonAthenaFullAccess`を選択します (Athena データカタログを使用する場合)。[Next] を選択します。

> **Note**
>
> `AmazonS3ReadOnlyAccess` ポリシーは、すべての Amazon S3 バケットに対する読み込み専用アクセス権をクラスターに付与します。AWS サンプルデータバケットへのアクセスのみを許可するには、新しいポリシーを作成して以下のアクセス許可を追加します。
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
>             "Effect": "Allow",
>             "Action": [
>                 "s3:Get*",
>                 "s3:List*"
>             ],
>             "Resource": "arn:aws:s3:::redshift-downloads/*"
>         }
>     ]
> }
> ```

7. [ロール名] に、**`myspectrum_role`**などのロール名を入力します。
8. 情報を確認してから、[ロールの作成] を選択します。
9. ナビゲーションペインで [Roles (ロール)] を選択します。新しいロールの名前を選択して概要を確認し、次にロール ARN をクリップボードにコピーします。この値は、作成したロールの Amazon リソースネーム (ARN) です。この値を使用して、外部テーブルの作成時に Amazon S3 のデータファイルを参照します。

AWS Lake Formation に対し有効化されている AWS Glue Data Catalog を使用して、Amazon Redshift の IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。

   [Policies] (ポリシー) を初めて選択する場合は、[Welcome to Managed Policies] (マネージドポリシーにようこそ) ページが表示されます。[Get Started] (今すぐ始める) を選択します。
3. [Create policy] を選択します。
4. [JSON] タブでポリシーの作成を選択します。
5. 次の JSON ポリシードキュメントに貼り付けます。これにより、データカタログへのアクセスは付与されますが、Lake Formation の管理者権限は拒否されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RedshiftPolicyForLF",
            "Effect": "Allow",
            "Action": [
                "glue:*",
                "lakeformation:GetDataAccess"
            ],
            "Resource": "*"
        }
    ]
}
```

6. 完了したら、[確認] を選択してポリシーを確認します。構文エラーがある場合は、Policy Validator によってレポートされます。
7. [ポリシーの確認] ページの [名前] に、**myspectrum_policy**と入力して、作成するポリシーに名前を付けます。[説明] (オプション) を入力します。ポリシーの [Summary] (概要) を参照して、ポリシーによって付与された許可を確認します。次に、[Create policy] (ポリシーの作成) を選択して作業を保存します。

   ポリシーを作成したら、ユーザーにアクセス権を付与できます。

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

Lake Formation データベースでクエリを実行するテーブルの SELECT 権限を付与するには

1. Lake Formation コンソール (https://console.aws.amazon.com/lakeformation/) を開きます。
2. ナビゲーションで、[データレイクアクセス許可]、[付与] の順に選択します。
3. [AWS Lake Formation 開発者ガイド」の「名前付きリソースメソッドによるテーブルのアクセス許可の付与」の手順に従ってください。以下の情報を記述します。
   - [IAM role (IAM ロール)] で、作成した IAM ロール (`myspectrum_role`) を選択します。Amazon Redshift クエリエディタを実行すると、IAM ロールを使用して、データに対するアクセス許可が付与されます。

   > **Note**
   >
   > Lake Formation が有効なデータカタログでテーブルに対する SELECT 許可を付与してクエリを実行するには、次を実行します。
   >
   > - Lake Formation にデータのパスを登録します。
   > - Lake Formation でそのパスへのアクセス許可をユーザーに付与します。
   > - 作成したテーブルは、Lake Formation で登録したパスで確認できます。
4. [Grant] (付与) を選択します。

> **Important**
>
> ベストプラクティスとして、Lake Formation 許可で基本の Amazon S3 オブジェクトにのみアクセスを許可することをお勧めします。承認されていないアクセスを防止するために、Lake Formation 外の Amazon S3 オブジェクトに許可が付与されている場合はそれらをすべて削除します。以前に Amazon S3 オブジェクトにアクセスしたことがあり、Lake Formation を設定する場合は、以前に設定した IAM ポリシーやバケットのアクセス許可をすべて削除します。詳細については、「AWS Lake Formation モデルへの AWS Glue データのアクセス許可のアップグレード」および「Lake Formation のアクセス許可」を参照してください。

ステップ 2: IAM ロールをクラスターと関連付ける

これで、Amazon Redshift が外部データカタログや Amazon S3 にアクセスすることを許可する IAM ロールが作成されました。ここで、そのロールと Amazon Redshift クラスターを関連付ける必要があります。

IAM ロールをクラスターに関連付けるには

1. AWS Management Consoleにサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択し、更新するクラスター名を選択します。
3. [アクション] で、[IAM ロールの管理] を選択します。[IAM ロール] のページが表示されます。
4. [Enter ARN] (ARN の入力) を選択し、ARN または IAM ロール を入力するか、リストから IAM ロールを選択します。その後、[Add IAM role (IAM ロールの追加)] を選択して、[Attached IAM roles (アタッチされている IAM ロール)] のリストに追加します。
5. [完了] を選択し、IAM ロールをクラスターに関連付けます。これで、クラスターが変更され、変更が完了します。

ステップ 3: 外部スキーマと外部テーブルを作成する

外部スキーマに外部テーブルを作成します。外部スキーマは、外部データカタログのデータベースを参照し、ユーザーに代わってクラスターの Amazon S3 へのアクセスを許可する IAM ロール ARN を提供します。外部データベースは、Amazon Athena データカタログ、AWS Glue Data Catalog、

または Amazon EMR などの Apache Hive メタストアに作成できます。この例では、外部スキーマ Amazon Redshift 作成時に Amazon Athena データカタログに外部データベースを作成します。詳細については、「Amazon Redshift Spectrum 用の外部スキーマ」を参照してください。

外部スキーマと外部テーブルを作成するには

1. 外部スキーマを作成するには、次のコマンドの IAM ロール ARN を、ステップ 1 で作成したロール ARN で置き換えます。次に、SQL クライアントでコマンドを実行します。

```
create external schema myspectrum_schema
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

2. 外部テーブルを作成するには、次の CREATE EXTERNAL TABLE コマンドを実行します。

> **Note**
>
> クラスターと Amazon S3 バケットは、同じ AWS リージョン に存在する必要があります。この CREATE EXTERNAL TABLE コマンドの例では、サンプルデータのある Amazon S3 バケットは米国東部 (バージニア北部) AWS リージョン にあります。ソースデータを表示するには、`sales_ts.000` ファイル をダウンロードします。
> この例を変更して、別の AWS リージョン で実行します。目的の AWS リージョン で Amazon S3 バケットを作成します。Amazon S3 コピーコマンドで販売データをコピーします。次に、この例の `CREATE EXTERNAL TABLE` コマンドにあるロケーションオプションをバケットに更新します。
>
> ```
> aws s3 cp s3://redshift-downloads/tickit/spectrum/sales/ s3://bucket-name/
> tickit/spectrum/sales/ --copy-props none --recursive
> ```
>
> Amazon S3 のコピーコマンドの出力により、ファイルが希望する AWS リージョン の *bucket-name* にコピーされたことが確認されます。
>
> ```
> copy: s3://redshift-downloads/tickit/spectrum/sales/sales_ts.000 to
>  s3://bucket-name/tickit/spectrum/sales/sales_ts.000
> ```

```
create external table myspectrum_schema.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

ステップ 4: Amazon S3 のデータにクエリを実行する

外部テーブルを作成すると、他の Amazon Redshift テーブルにクエリを実行する際に使用するものと同じ SELECT ステートメントでこれらのテーブルにクエリを実行できます。これらの SELECT ステートメントクエリには、テーブルの結合、データの集計、および述語のフィルタリングが含まれます。

Amazon S3 でデータをクエリするには

1. MYSPECTRUM_SCHEMA.SALES テーブルの行数を取得します。

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
172462
```

2. Amazon S3 に大きなファクトテーブルを、Amazon Redshift に小さなディメンションテーブルを保持することがベストプラクティスとなります。「データをロードする」でサンプルデータをロードした場合は、EVENT という名前のテーブルがデータベースに作成されます。テーブルが見つからない場合は、次のコマンドを使用して EVENT テーブルを作成します。

```
create table event(
eventid integer not null distkey,
venueid smallint not null,
catid smallint not null,
dateid smallint not null sortkey,
eventname varchar(200),
starttime timestamp);
```

3. 次の COPY コマンドの IAM ロール ARN を「ステップ 1. Amazon Redshift 用の IAM ロールを作成する」で作成したロール ARN と置き換え、[EVENT] テーブルをロードします。オプションで、AWS リージョン `us-east-1` の Amazon S3 バケットから `allevents_pipe.txt` のソースデータをダウンロードして表示できます。

```
copy event from 's3://redshift-downloads/tickit/allevents_pipe.txt'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-east-1';
```

次の例では、外部 Amazon S3 テーブル MYSPECTRUM_SCHEMA.SALES をローカル Amazon Redshift テーブルである EVENT と結合し、トップ 10 イベントの合計セールスを検出します。

```
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid) from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

4. 前のクエリのクエリプランを表示します。Amazon S3 のデータに対して実行された `S3 Seq Scan`、`S3 HashAggregate`、および `S3 Query Scan` のステップに注意してください。

```
explain
select top 10 myspectrum_schema.sales.eventid,
 sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
QUERY PLAN

-----------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Merge Key: sum(sales.derived_col2)

        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200
 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99
 rows=200 width=31)
```

```
                              ->  XN Hash Join DS_BCAST_INNER 
(cost=3119.97..1055769620.49 rows=200000 width=31)

                                    Hash Cond: ("outer".derived_col1 = "inner".eventid)


                                    ->  XN S3 Query Scan sales  (cost=3010.00..5010.50 
rows=200000 width=31)

                                          ->  S3 HashAggregate  (cost=3010.00..3010.50 
rows=200000 width=16)

                                                ->  S3 Seq Scan myspectrum_schema.sales 
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT 
(cost=0.00..2150.00 rows=172000 width=16)
                                                      Filter: (pricepaid > 30.00)


                                    ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                          ->  XN Seq Scan on event  (cost=0.00..87.98 
rows=8798 width=4)
```

AWS CloudFormation スタックを起動して Amazon S3 内のデータにクエリを実行する

Amazon Redshift クラスターを作成してクラスターに接続したら、Redshift Spectrum DataLake の AWS CloudFormation テンプレートをインストールし、データをクエリできるようになります。

CloudFormation は、Redshift Spectrum Getting Started DataLake テンプレートをインストールし、以下を含むスタックを作成します。

- Redshift クラスターに関連付けられている、`myspectrum_role`という名前のロール
- `myspectrum_schema` という名前の外部スキーマ
- Amazon S3 バケット内の、`sales`という名前の外部テーブル
- データがロードされた、`event`という名前の Redshift テーブル

Redshift Spectrum Getting Started DataLake で CloudFormation スタックを起動するには

1. [Launch CFN stack] (CFN スタックを起動する) をクリックします。DataLake.yml テンプレートが選択された状態で、CloudFormation コンソールが開きます。

   また、Redshift Spectrum Getting Started DataLake CloudFormation の CFN テンプレートをダウンロードしてカスタマイズし、CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開いて、カスタマイズしたテンプレートでスタックを作成することもできます。
2. [Next] を選択します。
3. [Parameters] (パラメータ)で、Amazon Redshift クラスター名、データベース名、およびデータベースのユーザー名を入力します。
4. [Next] を選択します。

   スタックに関するオプションが表示されます。
5. [Next] (次へ) をクリックして、デフォルト設定を受け入れます。
6. 情報を確認し、[機能] で、[AWS CloudFormation によって IAM リソースが作成される場合があることを承認します] を選択します。
7. [スタックの作成] を選択します。

スタックの作成中にエラーが発生した場合は、以下の情報をご確認ください。

- エラーの解決に役立つ情報については、CloudFormation の [Events] (イベント) タブを開きます。
- オペレーションを再試行する前に DataLake の CloudFormation スタックを削除します。
- Amazon Redshift データベースに接続されていることを確認します。
- Amazon Redshift クラスター名、データベース名、およびデータベースのユーザー名に関する情報を、正確に入力しているか確認します。

Amazon S3 でデータをクエリする

他の Amazon Redshift テーブルにクエリを実行する際に使用するものと同じ SELECT ステートメントで、外部テーブルでもクエリを実行できます。これらの SELECT ステートメントクエリには、テーブルの結合、データの集計、および述語のフィルタリングが含まれます。

次のクエリは、外部テーブル `myspectrum_schema.sales` 内の行数を返します。

```
select count(*) from myspectrum_schema.sales;
```

```
count
------
172462
```

外部テーブルとローカルテーブルを結合する

次の例では、外部テーブル myspectrum_schema.sales をローカルテーブルである event と結合し、トップ 10 イベントの合計セールスを検出しています。

```
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
 from myspectrum_schema.sales, event
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

クエリプランを表示する

前のクエリのクエリプランを表示します。Amazon S3 のデータに対して実行された S3 Seq Scan、S3 HashAggregate、および S3 Query Scan のステップをメモします。

```
explain
select top 10 myspectrum_schema.sales.eventid, sum(myspectrum_schema.sales.pricepaid)
from myspectrum_schema.sales, event
```

```
where myspectrum_schema.sales.eventid = event.eventid
and myspectrum_schema.sales.pricepaid > 30
group by myspectrum_schema.sales.eventid
order by 2 desc;
```

```
QUERY PLAN

-----------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Merge Key: sum(sales.derived_col2)

        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
 width=31)

                          ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
 rows=200000 width=31)

                                Hash Cond: ("outer".derived_col1 = "inner".eventid)

                                ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
 rows=200000 width=31)
```

```
                                    ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                          ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                Filter: (pricepaid > 30.00)


                              ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                    ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

Amazon Redshift Spectrum 用の IAM ポリシー

このトピックでは、Redshift Spectrum を使用するために必要な IAM アクセス許可について説明します。

デフォルトでは、Amazon Redshift Spectrum は AWS Glue をサポートする AWS リージョンで AWS Glue Data Catalog を使用します。その他の AWS リージョンでは、Redshift Spectrum は Athena データカタログを使用します。クラスターには、AWS Glueまたは Athena に置かれた外部データカタログや、Amazon S3 内のデータファイルにアクセスするための承認が必要です。この承認は、クラスターにアタッチされたAWS Identity and Access Management(IAM) ロールを参照することで提供できます。Apache Hive メタストアを使用してデータカタログを管理している場合は、Athena へのアクセスを提供する必要はありません。

ロールを連鎖することで、クラスターがそのクラスターにアタッチされていない別のロールを引き受けることができます。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

アクセスする AWS Glue カタログは、セキュリティを強化するために暗号化されている可能性があります。AWS Glue カタログが暗号化されている場合、AWS KMSカタログにアクセスするために AWS Glue の AWS Glue キーが必要です。詳細については、AWS Glueデベロッパーガイドの「AWS Glue データカタログの暗号化」を参照してください。

トピック

- Amazon S3 のアクセス許可
- クロスアカウント Amazon S3 のアクセス許可

- Redshift Spectrum を使用したアクセスを付与あるいは制限するポリシー
- 最小限のアクセス許可を付与するポリシー
- Amazon Redshift Spectrum での IAM ロールの連鎖
- AWS Glue データカタログへのアクセスの制御

Amazon S3 のアクセス許可

少なくとも、クラスターには Amazon S3 バケットへの GET および LIST アクセスが必要です。バケットがクラスターと同じ AWS アカウントに存在しない場合、バケットは、データにアクセスするための許可をクラスターに付与する必要もあります。詳細については、「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを認可する」を参照してください。

> **Note**
>
> Amazon S3 バケットは、特定の VPC エンドポイントからのアクセスのみを制限するバケットポリシーを使用できません。

次のポリシーは、あらゆる Amazon S3 バケットに GET および LIST アクセスを付与します。このポリシーは、Redshift Spectrum が COPY 操作と同様に Amazon S3 バケットへアクセスするのを許可します。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "*"
 }]
}
```

次のポリシーは、`amzn-s3-demo-bucket` という名前の Amazon S3 バケットへの GET および LIST アクセスを付与します。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
```

```
  "Action": ["s3:Get*", "s3:List*"],
  "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
 }]
}
```

クロスアカウント Amazon S3 のアクセス許可

別の AWS アカウントに属している Amazon S3 バケット内のデータにアクセスできる許可を Redshift Spectrum に付与するには、次のポリシーを Amazon S3 バケットに追加します。詳細については、「クロスアカウントのバケットのアクセス許可を付与」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Example permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::redshift-account:role/spectrumrole"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/*"
            ]
        }
    ]
}
```

Redshift Spectrum を使用したアクセスを付与あるいは制限するポリシー

Redshift Spectrum のみを使用して Amazon S3 バケットへのアクセスを許可するには、ユーザーエージェント `AWS Redshift/Spectrum` のアクセスを許可する条件を含めます。次のポリシーでは、Redshift Spectrum だけが Amazon S3 バケットにアクセスできます。COPY 操作など、その他のアクセスは除外されます。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "Condition": {"StringEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

同様に、COPY 操作のアクセスは許可し、Redshift Spectrum のアクセスは除外する IAM ロールを作成することもできます。これを行うには、ユーザーエージェント **AWS Redshift/Spectrum** のアクセスを拒否する条件を含めます。次のポリシーは、Redshift Spectrum を除いて、Amazon S3 バケットへのアクセスを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["s3:Get*", "s3:List*"],
        "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "Condition": {"StringNotEquals": {"aws:UserAgent": "AWS Redshift/
Spectrum"}}
    }]
}
```

最小限のアクセス許可を付与するポリシー

次のポリシーは、Redshift Spectrum を Amazon S3、AWS Glue、および Athena で使用するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListMultipartUploadParts",
```

```
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads"
            ],
            "Resource": [
                "arn:aws:s3:::bucketname",
                "arn:aws:s3:::bucketname/folder1/folder2/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS Glue の代わりに Athena をデータカタログに使用する場合、このポリシーには Athena に対する完全なアクセス許可が必要となります。次のポリシーは、Athena リソースへのアクセスを付与します。外部データベースが Hive メタストアにある場合は、Athena アクセスは必要ありません。

```
{
```

```
 "Version": "2012-10-17",
 "Statement": [{
  "Effect": "Allow",
  "Action": ["athena:*"],
  "Resource": ["*"]
 }]
}
```

Amazon Redshift Spectrum での IAM ロールの連鎖

クラスターにロールをアタッチすると、クラスターはそのロールを引き受けて、ユーザーに代わって Amazon S3、Athena、および AWS Glue にアクセスできるようになります。クラスターにアタッチされたロールに必要なリソースへのアクセスがない場合、他のアカウントに属している可能性がある別のロールを連鎖することができます。クラスターは、このデータにアクセスするための連鎖ロールを一時的に引き受けます。また、ロールを連鎖してクロスアカウントアクセスを付与することもできます。最大で 10 個のロールを連鎖できます。連鎖における各ロールは、クラスターが連鎖の末尾のロールを引き受けるまで、連鎖の次のロールを引き受けます。

ロールを連鎖するには、ロール間で信頼関係を確立します。別のロールを引き受けるロールには、特定のロールを引き受けることができるアクセス権限ポリシーがあることが必要です。また、アクセス許可を渡すロールは、別のロールにアクセス許可を渡すことができる信頼ポリシーを保持していることが必要です。詳細については、Amazon Redshift で IAM ロールを連鎖するを参照してください。

CREATE EXTERNAL SCHEMA コマンドを実行すると、ロールの ARN のカンマ区切りのリストを含めることで、ロールを連鎖することができます。

> **Note**
> 連鎖したロールのリストは、空白を含むことができません。

次の例では、`MyRedshiftRole`がクラスターにアタッチされます。`MyRedshiftRole` は、アカウント `AcmeData` に属するロール `111122223333` を引き受けます。

```
create external schema acme from data catalog
database 'acmedb' region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole,arn:aws:iam::111122223333:role/
AcmeData';
```

AWS Glue データカタログへのアクセスの制御

データカタログに AWS Glue を使用する場合、IAM ポリシーで AWS Glue データカタログにきめ細かなアクセスコントロールを適用できます。例えば、特定の IAM ロールに少数のデータベースとテーブルのみを公開する場合があります。

以下のセクションで、AWS Glueデータカタログに保存されているデータへのさまざまなレベルアクセスの IAM ポリシーについて説明します。

トピック

- データベース操作のポリシー
- テーブル操作のポリシー
- パーティション操作のポリシー

データベース操作のポリシー

データベースを表示および作成するアクセス許可をユーザーに付与する場合、ユーザーにはデータベースと AWS Glue データカタログの両方へのアクセス権限が必要です。

次のサンプルクエリは、データベースを作成します。

```
CREATE EXTERNAL SCHEMA example_db
FROM DATA CATALOG DATABASE 'example_db' region 'us-west-2'
IAM_ROLE 'arn:aws:iam::redshift-account:role/spectrumrole'
CREATE EXTERNAL DATABASE IF NOT EXISTS
```

次の IAM ポリシーは、データベースを作成するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:CreateDatabase"
```

```
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:catalog"
            ]
        }
    ]
}
```

次のサンプルクエリは、現在のデータベースを表示します。

```
SELECT * FROM SVV_EXTERNAL_DATABASES WHERE
databasename = 'example_db1' or databasename = 'example_db2';
```

次の IAM ポリシーは、現在のデータベースを表示するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabases"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:database/example_db1",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db2",
                "arn:aws:glue:us-west-2:redshift-account:catalog"

            ]
        }
    ]
}
```

テーブル操作のポリシー

テーブルで表示、作成、削除、変更またはその他のアクションを実行するアクセス許可をユーザーに付与する場合、ユーザーにいくつかのタイプのアクセスを許可する必要があります。テーブルそのものに加え、テーブルが属するデータベース、およびカタログへのアクセス許可が必要です。

次のサンプルクエリは、外部テーブルを作成します。

```
CREATE EXTERNAL TABLE example_db.example_tbl0(
    col0 INT,
    col1 VARCHAR(255)
) PARTITIONED BY (part INT) STORED AS TEXTFILE
LOCATION 's3://test/s3/location/';
```

次の IAM ポリシーは、外部テーブルを作成するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

次のサンプルクエリは、それぞれ現在の外部テーブルを表示します。

```
SELECT * FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
```

```
tablename = 'example_tbl1';
```

```
SELECT * FROM svv_external_columns
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

```
SELECT parameters FROM svv_external_tables
WHERE tablename = 'example_tbl0' OR
tablename = 'example_tbl1';
```

次の IAM ポリシーは、現在の外部テーブルを表示するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTables"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/
example_tbl0",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl1"

            ]
        }
    ]
}
```

次のサンプルクエリは、既存のテーブルを変更します。

```
ALTER TABLE example_db.example_tbl0
SET TABLE PROPERTIES ('numRows' = '100');
```

次の IAM ポリシーは、既存のテーブルを変更するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"

            ]
        }
    ]
}
```

次のサンプルクエリは、既存のテーブルを削除します。

```
DROP TABLE example_db.example_tbl0;
```

次の IAM ポリシーは、既存のテーブルを削除するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeleteTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

パーティション操作のポリシー

パーティションレベルの操作 (表示、作成、削除、変更など) を実行するアクセス許可をユーザーに付与する場合、ユーザーにはパーティションが属するテーブルへのアクセス許可が必要です。また、関連するデータベースと AWS Glue データカタログへのアクセス許可も必要です。

次のサンプルクエリは、パーティションを作成します。

```
ALTER TABLE example_db.example_tbl0
ADD PARTITION (part=0) LOCATION 's3://test/s3/location/part=0/';
ALTER TABLE example_db.example_t
ADD PARTITION (part=1) LOCATION 's3://test/s3/location/part=1/';
```

次の IAM ポリシーは、パーティションを作成するために必要な最小限のアクセス許可を付与します。

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetTable",
                "glue:BatchCreatePartition"
```

```
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

次のサンプルクエリは、現在のパーティションを表示します。

```
SELECT * FROM svv_external_partitions
WHERE schemname = 'example_db' AND
tablename = 'example_tbl0'
```

次の IAM ポリシーは、現在のパーティションを表示するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartitions",
                "glue:GetTables",
                "glue:GetTable"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

次のサンプルクエリは、既存のパーティションを変更します。

```
ALTER TABLE example_db.example_tbl0 PARTITION(part='0')
SET LOCATION 's3://test/s3/new/location/part=0/';
```

次の IAM ポリシーは、既存のパーティションを変更するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetPartition",
                "glue:UpdatePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

次のサンプルクエリは、既存のパーティションを削除します。

```
ALTER TABLE example_db.example_tbl0 DROP PARTITION(part='0');
```

次の IAM ポリシーは、既存のパーティションを削除するために必要な最小限のアクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:DeletePartition"
            ],
            "Resource": [
                "arn:aws:glue:us-west-2:redshift-account:catalog",
                "arn:aws:glue:us-west-2:redshift-account:database/example_db",
                "arn:aws:glue:us-west-2:redshift-account:table/example_db/example_tbl0"
            ]
        }
    ]
}
```

Redshift Spectrum と AWS Lake Formation

このトピックでは、Lake Formation で Redshift Spectrum を使用する方法について説明します。Lake Formation は、分析データを共有するためのサービスです。

AWS Lake Formation を使用して、Amazon S3 に格納されているデータに対しデータベース、テーブル、および列レベルのアクセスポリシーを一元的に定義および執行することができます。Lake Formation で有効になっている AWS Glue Data Catalog にデータを登録したら、Redshift Spectrum などの複数のサービスを使用したクエリができるようになります。

Lake Formation は、データカタログのセキュリティとガバナンスを提供します。Lake Formation 内では、データカタログオブジェクト (例: データベース、テーブル、列、基本の Amazon S3 ストレージ) へのアクセス許可の付与または取り消しを行うことができます。

> **Important**
>
> Redshift Spectrum は、Lake Formation が利用可能な AWS リージョンにおいて、Lake Formation が有効になっているデータカタログでのみ使用が可能です。利用可能なリージョンの一覧については、「AWS 全般のリファレンス」の「AWS Lake Formation エンドポイントとクォータ」を参照してください。

Lake Formation で Redshift Spectrum を使用すると、次のことができます。

- データレイク内のすべてのデータに対するアクセス許可とアクセス制御ポリシーを付与および取り消しする、一元化された場所として Lake Formation を使用します。Lake Formation には、データカタログ内のデータベースとテーブルへのアクセスを制御するためのアクセス許可の階層があります。詳細については、AWS Lake Formation デベロッパーガイドの「Lake Formation 許可の概要」を参照してください。
- 外部テーブルを作成し、データレイク内のデータに対してクエリを実行します。アカウントのユーザーがクエリを実行する前に、データレイクのアカウント管理者は、ソースデータを含む既存の Amazon S3 パスを Lake Formation に登録します。また、管理者はテーブルを作成し、アクセス許可をユーザーに付与します。データベース、テーブル、または列に対するアクセス権を付与できます。管理者は Lake Formation のデータフィルターを使用して、Amazon S3 に保存されている機密データに対するきめ細かなアクセスコントロールを許可できます。詳細については、「行レベルおよびセルレベルのセキュリティでのデータフィルターの使用」を参照してください。

  データがデータカタログに登録された後、ユーザーがクエリを実行しようとする度に、Lake Formation はその特定のプリンシパルのテーブルへのアクセスを確認します。Lake Formation は Redshift Spectrum に一時的な認証情報を引き渡し、クエリを実行します。
- `GetCredentials` または `GetClusterCredentials` で取得した IAM 認証情報を使用して自動的にマウントされた AWS Glue Data Catalog に対して Redshift Spectrum クエリを実行し、データベースユーザー (IAMR: Username または IAM: Username) ごとに Lake Formation アクセス許可を管理します。

Lake Formation が有効なデータカタログで Redshift Spectrum を使用する場合は、次のいずれかが必要です。

- データカタログへのアクセス許可を持つクラスターに関連付けられた IAM ロール。
- 外部リソースへのアクセスを管理するように設定されたフェデレーション IAM ID。詳細については、「フェデレーション ID を使用して、ローカルリソースと Amazon Redshift Spectrum の外部テーブルへの Amazon Redshift アクセスを管理する」を参照してください。

> **Important**
>
> Lake Formation でデータカタログを有効にして Redshift Spectrum を使用する場合は、IAM ロールを連鎖することはできません。

Redshift Spectrum で使用するように AWS Lake Formation を設定するために必要な手順の詳細については、「AWS Lake Formation デベロッパーガイド」の「チュートリアル: Lake Formation での JDBC ソースからのデータレイクの作成」を参照してください。Redshift Spectrum との統合の詳細については、「Amazon Redshift Spectrum を使用してデータレイク内のデータをクエリする」を参照してください。このトピックで使用するデータと AWS リソースは、チュートリアルの以前のステップに応じて異なります。

行レベルおよびセルレベルのセキュリティでのデータフィルターの使用

AWS Lake Formation でデータフィルターを定義し、データカタログに定義されているデータに対する Redshift Spectrum クエリの行レベルおよびセルレベルのアクセスを制御できます。これを設定するには、以下のタスクを実行します。

- 以下の情報を使用して、Lake Formation でデータフィルターを作成します。
  - クエリ結果に含める、またはクエリ結果から除外する列のリストを含む列指定。
  - クエリ結果に含める行を指定する行フィルター式。

  データフィルターの作成方法の詳細については、AWS Lake Formation デベロッパーガイドの「Lake Formation でのデータフィルター」を参照してください。
- Amazon Redshift で、Lake Formation に対応したデータカタログのテーブルを参照する外部テーブルを作成します。Redshift Spectrum を使用して Lake Formation テーブルをクエリする方法の詳細については、AWS Lake Formation デベロッパーガイドの「Amazon Redshift Spectrum を使用してデータレイク内のデータをクエリする」を参照してください。

Amazon Redshift でテーブルを定義したら、Lake Formation テーブルにクエリを実行して、データフィルターが許可する行と列にのみアクセスできます。

Lake Formation で行レベルとセルレベルのセキュリティを設定し、Redshift Spectrum を使用してクエリを実行する方法の詳細なガイドについては、「Use Amazon Redshift Spectrum with row-level and cell-level security policies defined in AWS Lake Formation」を参照してください。

Amazon Redshift Spectrum でクエリ用のデータファイルを作成する

このセクションでは、Redshift Spectrum がサポートする形式で Amazon S3 にデータファイルを作成する方法について説明します。

Amazon Redshift Spectrum のクエリに使用するデータファイルは通常、他のアプリケーションで使用するものと同じタイプのファイルです。例えば、Amazon Athena、Amazon EMR、および Amazon QuickSight で同じ種類のファイルが使用されます。Amazon S3 から直接元の形式でデータをクエリすることができます。これを実行するには、データファイルは、Redshift Spectrum がサポートしている形式で、クラスターがアクセスできる Amazon S3 バケットに存在している必要があります。

データファイルを含む Amazon S3 バケットと Amazon Redshift クラスターは、同じ AWS リージョンに存在する必要があります。サポートされる AWS リージョンの詳細については、「Amazon Redshift Spectrum リージョン」を参照してください。

Redshift Spectrum のデータ形式

Redshift Spectrum は次の構造化されたデータ形式および半構造化されたデータ形式をサポートします。

ファイル形式	列指向	並列読み取りをサポート	分割単位
Parquet	はい	はい	[行] グループ
ORC	はい	はい	Stripe
RCFile	はい	はい	[行] グループ
TextFile	いいえ	はい	行
SequenceFile	いいえ	はい	行またはブロック
RegexSerde	いいえ	はい	行
OpenCSV	いいえ	はい	行
AVRO	いいえ	はい	ブロック
Ion	いいえ	いいえ	該当なし
JSON	いいえ	いいえ	該当なし

前述の表の見出しは、次のことを示しています。

- 列指向 – ファイル形式が、行指向の構造ではなく、列指向の構造でデータを物理的に保存するかどうかを指定します。
- 並列読み込みをサポート – ファイル形式でファイル内の個々のブロックの読み込みをサポートするかどうかを指定します。個々のブロックを読み込むと、単一のリクエストで完全なファイルを読み込む代わりに、複数の独立した Redshift Spectrum リクエストにまたがってファイルを分散処理できます。
- 分割単位 – 並列に読み込むことができるファイル形式の場合、分割単位は 1 つの Redshift Spectrum リクエストで処理できるデータの最小チャンクです。

> **Note**
>
> テキストファイルのタイムスタンプ値は、`yyyy-MM-dd HH:mm:ss.SSSSSS`形式であることが必要です。次のタイムスタンプ値は、`2017-05-01 11:30:59.000000`となっています。

Apache Parquet など、列指向ストレージファイル形式を使用することをお勧めします。列指向ストレージファイル形式により、必要な列のみを選択し、Amazon S3 からのデータ転送を最小限に抑えることができます。

Redshift Spectrum の圧縮タイプ

ストレージスペースの縮小、パフォーマンスの向上、コストの最小化を行うため、データファイルを圧縮することを強くおすすめします。Redshift Spectrum は、ファイル拡張子に基づいてファイル圧縮のタイプを認識します。

Redshift Spectrum は、次の圧縮タイプと拡張子をサポートしています。

圧縮アルゴリズム	ファイル拡張子	並列読み取りをサポート
Gzip	.gz	いいえ
Bzip2	.bz2	はい
Snappy	.snappy	いいえ

さまざまなレベルで圧縮を適用できます。通常、ファイル全体を圧縮するか、ファイル内の個々のブロックを圧縮します。ファイルレベルで列形式を圧縮しても、パフォーマンス上の利点はありません。

Redshift Spectrum がファイルを並行して読み込めるようにするには、次の条件を満たす必要があります。

- ファイル形式は、並列読み取りをサポートしています。
- ファイルレベルの圧縮があれば、並列読み込みがサポートされます。

ファイル内の各分割ユニットが 1 つの Redshift Spectrum リクエストによって処理されるため、並列に読み込むことができる圧縮アルゴリズムを使用して圧縮されているかどうかは関係ありません。例えば、Snappy 圧縮の Parquet ファイルです。Parquet ファイル内の個々の行グループは Snappy を使用して圧縮されますが、ファイルの最上位構造は圧縮されていないままです。この場合、各 Redshift Spectrum リクエストは Amazon S3 から個々の行グループを読み込み、処理できるため、ファイルを並列に読み込むことができます。

Redshift Spectrum の暗号化

Redshift Spectrum は、次の暗号化オプションを使用して暗号化されたデータファイルを透過的に復号化します。

- Amazon S3 によって管理される AES-256 暗号化キーを使用したサーバー側暗号化 (SSE-S3)。
- AWS Key Management Service で管理されたキーによるサーバー側の暗号化 (SSE-KMS)。

Redshift Spectrum は、Amazon S3 クライアント側暗号化をサポートしていません。サーバー側の暗号化の詳細については、Amazon Simple Storage Service ユーザーガイドの[サーバー側の暗号化を使用したデータの保護]を参照してください。

Amazon Redshift は、超並列処理 (MPP) により、大量のデータに対して非常に複雑なクエリを高速で実行できます。Redshift Spectrum は、ファイルのスキャンで必要な複数の Redshift Spectrum インスタンスを使用して、同じ原理を外部データに拡張します。テーブルごとにファイルを個別のフォルダに置きます。

次のことを実行してデータを並列処理用に最適化できます。

- ファイル形式または圧縮で並列読み取りがサポートされていない場合は、大きなファイルを多数の小さなファイルに分割します。ファイルサイズは 64 MB～1 GBをお勧めします。

- ファイルはすべてほぼ同じサイズで保存します。一部のファイルのサイズが他のファイルを大きく上回ると、Redshift Spectrum はワークロードを均等に分散できません。

Amazon Redshift Spectrum 用の外部スキーマ

このトピックでは、Redshift Spectrum で外部スキーマの作成および使用する方法について説明します。外部スキーマは、Amazon Redshift クラスター外のデータにアクセスするための参照として使用するテーブルのコレクションです。これらのテーブルには、Redshift Spectrum が読み取る外部データに関するメタデータが含まれています。

外部テーブルは、CREATE EXTERNAL SCHEMAステートメントを使用して作成した外部スキーマで作成する必要があります。

> **Note**
> 一部のアプリケーションでは、データベースとスキーマという用語をほぼ同じ意味で使用しています。Amazon Redshift では、スキーマという用語を使用します。

Amazon Redshift 外部スキーマは、外部データカタログ内の外部データベースを参照します。Amazon Redshift の外部データベースは、Amazon Athena、AWS Glue Data Catalog、または Apache Hive メタストア (Amazon EMR など) で作成できます。Amazon Redshift で外部データベースを作成すると、データベースは Athena データカタログに保存されます。Hive メタストアでデータベースを作成するには、Hive アプリケーションでデータベースを作成する必要があります。

Amazon Redshift には、ユーザーに代わって Athena のデータカタログや Amazon S3 のデータファイルにアクセスするための承認が必要です。この承認を提供するには、最初に AWS Identity and Access Management (IAM) ロールを作成します。その後、ロールをクラスターにアタッチし Amazon Redshift `CREATE EXTERNAL SCHEMA` ステートメントのロール用に Amazon リソースネーム (ARN) を提供します。認可の詳細については、「Amazon Redshift Spectrum 用の IAM ポリシー」を参照してください。

> **Note**
> 現在、Athena データカタログに Redshift Spectrum 外部テーブルがある場合は、Athena データカタログを AWS Glue データカタログに移行することが可能です。Redshift Spectrum で AWS Glue データカタログを使用するには、IAM ポリシーの変更が必要になる場合があり

ます。詳細については、Amazon Athena ユーザーガイドの「AWS Glue データカタログへのアップグレード」を参照してください。

外部スキーマの作成と同時に外部データベースを作成するには、FROM DATA CATALOGステートメントで CREATE EXTERNAL DATABASE を指定して CREATE EXTERNAL SCHEMA 句を含めます。

次の例では、外部データベース spectrum_schema を使用して spectrum_db という名前の外部スキーマを作成します。

```
create external schema spectrum_schema from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

Athena を使用してデータカタログを管理する場合は、Athena データベース名と Athena データカタログが置かれている AWS リージョンを指定します。

次の例では、Athena データカタログにあるデフォルトの sampledb データベースを使用して外部スキーマを作成します。

```
create external schema athena_schema from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
region 'us-east-2';
```

Note

region パラメータは、Amazon S3 のデータファイルの場所ではなく、Athena データカタログが置かれている AWS リージョンを参照します。

Amazon EMR などの Hive メタストアを使用してデータカタログを管理する場合、セキュリティグループはクラスター間でトラフィックを許可するように設定する必要があります。

CREATE EXTERNAL SCHEMA ステートメントで、FROM HIVE METASTOREを指定し、メタストアの URI とポート番号を含めます。次の例では、hive_dbという名前の Hive メタストアデータベースを使って外部スキーマを作成します。

```
create external schema hive_schema
```

```
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
```

クラスターの外部スキーマを表示するには、PG_EXTERNAL_SCHEMA カタログテーブル、あるいは SVV_EXTERNAL_SCHEMAS ビューのクエリを実行します。次の例では、PG_EXTERNAL_SCHEMA と PG_NAMESPACE を結合する SVV_EXTERNAL_SCHEMAS クエリを実行します。

```
select * from svv_external_schemas
```

完全なコマンドの構文と例については、「CREATE EXTERNAL SCHEMA」を参照してください。

Amazon Redshift Spectrum での外部カタログの使用

Amazon Redshift Spectrum データベースと外部データテーブルのメタデータは、外部データカタログに保存されます。デフォルトでは、Redshift Spectrum メタデータは Athena データカタログに保存されます。Redshift Spectrum のデータベースとテーブルは、Athena コンソールで表示して管理できます。

Hive データ定義言語 (DDL)、Athena、または Hive メタストア (Amazon EMR など) を使用して、外部データベースと外部テーブルを作成および管理することもできます。

> **Note**
>
> Amazon Redshift を使用して Redshift Spectrum 外部データベースと外部テーブルを作成および管理することをおすすめします。

Athena および AWS Glue の Redshift Spectrum データベースの表示

外部データベースは、CREATE EXTERNAL SCHEMA ステートメントの一環として CREATE EXTERNAL DATABASE IF NOT EXISTS 句を含めることで作成できます。このケースでは、外部データベースのメタデータは、 データカタログに保存されます。作成した外部テーブルのメタデータも、外部スキーマに修飾され、 データカタログに保存されます。

Athena および AWS Glue は、サポートされる各 AWS リージョン リージョン用にデータカタログを保持します。テーブルのメタデータを表示するには、Athena または AWS Glue コンソールにログオ

ンします。Athena で、データソース、ユーザーの AWS Glue を選択し、データベースの詳細を表示します。AWS Glue で、データベース、ユーザーの外部データベースを選択し、データベースの詳細を表示します。

Athena を使用して外部テーブルを作成および管理する場合は、CREATE EXTERNAL SCHEMA を使用してデータベースを登録します。例えば、次のコマンドは sampledb という名前の Athena データベースを登録します。

```
create external schema athena_sample
from data catalog
database 'sampledb'
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole'
region 'us-east-1';
```

SVV_EXTERNAL_TABLES システムビューにクエリを実行すると、Athena sampledb データベースにテーブルが表示され、Amazon Redshift で作成したテーブルも表示されます。

```
select * from svv_external_tables;
```

```
schemaname    | tablename         | location

--------------+------------------
+-------------------------------------------------------
athena_sample | elb_logs          | s3://athena-examples/elb/plaintext
athena_sample | lineitem_1t_csv   | s3://myspectrum/tpch/1000/lineitem_csv

athena_sample | lineitem_1t_part | s3://myspectrum/tpch/1000/lineitem_partition

spectrum      | sales             | s3://redshift-downloads/tickit/spectrum/sales

spectrum      | sales_part        | s3://redshift-downloads/tickit/spectrum/sales_part
```

Apache Hive メタストアデータベースの登録

Apache Hive メタストアで外部テーブルを作成すると、CREATE EXTERNAL SCHEMA を使用してこれらのテーブルを Redshift Spectrum に登録できます。

CREATE EXTERNAL SCHEMA ステートメントで FROM HIVE METASTORE 句を指定し、Hive メタストア URI とポート番号を指定します。IAM ロールには Amazon S3 への許可が必要ですが、Athena 許可は必要ありません。次の例では、Hive メタストアを登録します。

```
create external schema if not exists hive_schema
from hive metastore
database 'hive_database'
uri 'ip-10-0-111-111.us-west-2.compute.internal' port 9083
iam_role 'arn:aws:iam::123456789012:role/mySpectrumRole';
```

Amazon Redshift クラスターが Amazon EMR クラスターへのアクセスを有効にする

Hive メタストアが Amazon EMR にある場合、Amazon Redshift クラスターに Amazon EMR クラスターへのアクセスを許可する必要があります。これを実行するには、Amazon EC2 セキュリティグループを作成します。そのためには、セキュリティグループを作成し、Amazon Redshift クラスターのセキュリティグループや Amazon EMR クラスターのセキュリティグループから EC2 セキュリティグループへのすべてのインバウンドトラフィックを許可します。その後、Amazon Redshift クラスターと Amazon EMR クラスターの両方に EC2 セキュリティを追加します。

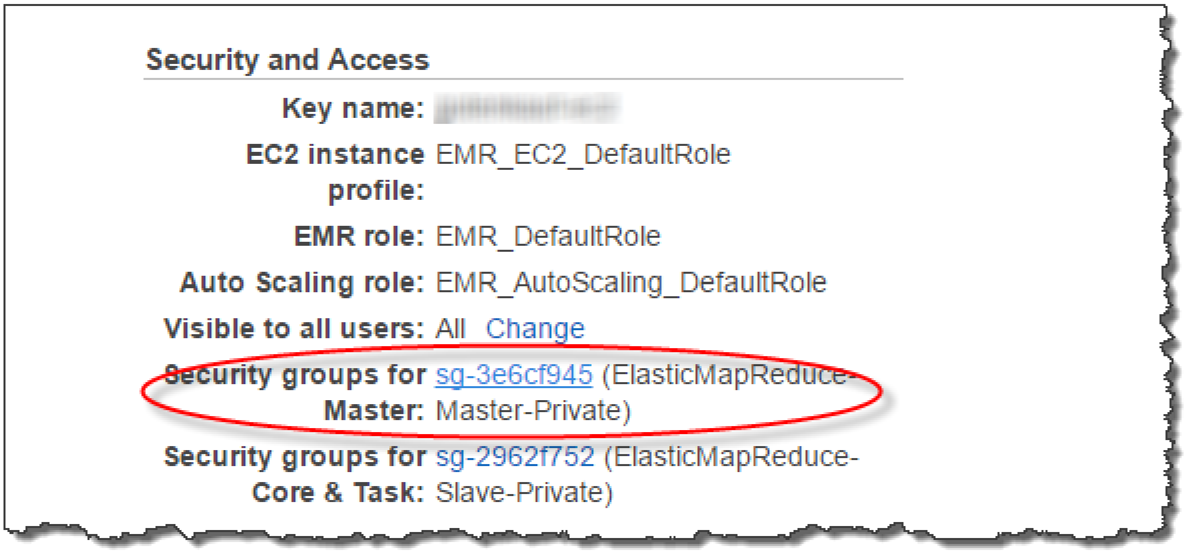
Amazon Redshift クラスターのセキュリティグループ名を表示する

セキュリティグループを表示するには、以下を実行します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択し、リストからクラスターを選択してその詳細を開きます。
3. [プロパティ] をクリックし、[ネットワークとセキュリティの設定] セクションを開きます。
4. [VPC セキュリティグループ] から自分のセキュリティグループ見つけ、それを書き留めます。

Amazon EMR のマスターノードセキュリティグループ名を表示する

1. Amazon EMR クラスターを開きます。詳細については、Amazon EMR 管理ガイドの「セキュリティ構成を使用してクラスターのセキュリティを設定する」を参照してください。
2. [Security and access] (セキュリティとアクセス) で、Amazon EMR のマスターノードセキュリティグループ名を書き留めます。



Amazon Redshift と Amazon EMR 間の接続を許可するために、Amazon EC2 セキュリティグループを作成または変更するには

1. Amazon EC2 ダッシュボードで [セキュリティグループ)] をクリックします。詳細については、「Amazon EC2 ユーザーガイド」の「セキュリティグループのルール」を参照してください。
2. [セキュリティグループの作成] を選択します。
3. VPC を使用する場合、Amazon Redshift と Amazon EMR のクラスターが存在する VPC を選択します。
4. インバウンドルールを追加します。
    1. [タイプ] で [カスタム TCP] を選択します。
    2. [ソース] で [カスタム] を選択します。
    3. Amazon Redshift セキュリティグループの名前を入力します。
5. 別のインバウンドルールを追加します。
    1. [タイプ] で [TCP] を選択します。
    2. [Port Range (ポート範囲)] に、「9083」と入力します。

> **Note**
>
> EMR HMS のデフォルトのポートは 9083 です。HMS が別のポートを使用する場合、そのポートをインバウンドルールと外部スキーマ定義に指定します。

3. [ソース] で [カスタム] を選択します。

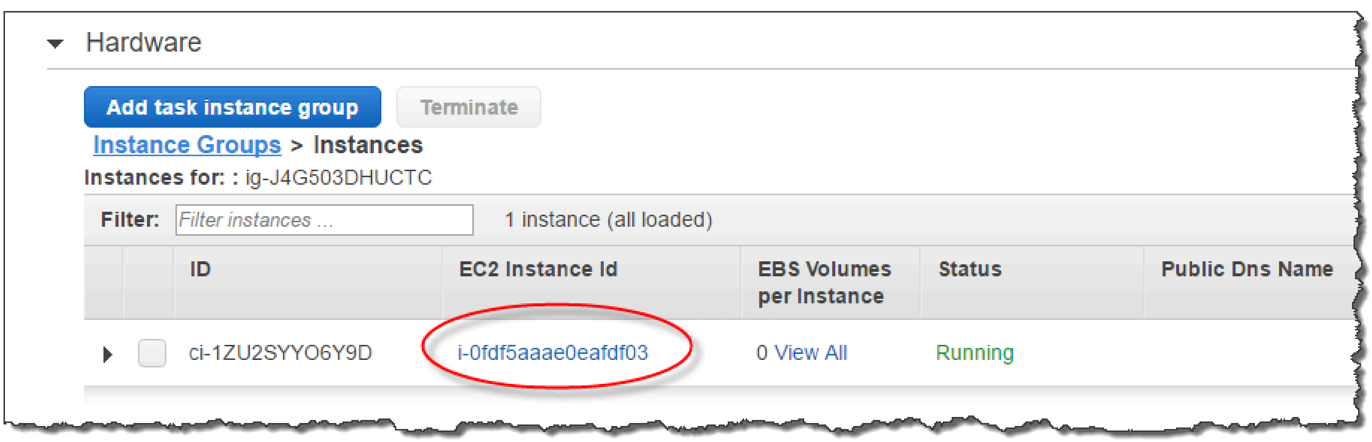
6. セキュリティグループの名前と説明を入力します。
7. [セキュリティグループの作成] を選択します。

前のステップで作成した Amazon EC2 セキュリティグループを Amazon Redshift クラスターに追加するには

1. Amazon Redshift で、クラスターを選択します。
2. [プロパティ] を選択します。
3. [Network and security settings] (ネットワークとセキュリティ設定) を表示して、[Edit] (編集) をクリックします。
4. [VPC セキュリティグループ] で、新しいセキュリティグループ名を選択します。
5. [変更を保存] をクリックします。

Amazon EMR クラスターに Amazon EC2 セキュリティグループを追加するには

1. Amazon EMR で、クラスターを選択します。詳細については、Amazon EMR 管理ガイドの「セキュリティ構成を使用してクラスターのセキュリティを設定する」を参照してください。
2. [ハードウェア] でマスターノードのリンクを選択します。
3. [EC2 インスタンス ID] 列からリンクを選択します。



4. [アクション]、[セキュリティ]、[セキュリティグループを変更] の順にクリックします。
5. [Associated sercurity groups] (関連付けられたセキュリティグループ) で、新しいセキュリティグループを選択してから [Add security group] (セキュリティグループを追加) をクリックします。

6. [Save] を選択します。

Redshift Spectrum 用の外部テーブル

このトピックでは、Redshift Spectrum で外部テーブルの作成および使用する方法について説明します。外部テーブルは、Amazon Redshift クラスター外のデータにアクセスするための参照として使用するテーブルです。これらのテーブルには、Redshift Spectrum が読み取る外部データに関するメタデータが含まれています。

外部テーブルは外部スキーマで作成します。外部テーブルを作成するには、外部スキーマの所有者またはスーパーユーザーである必要があります。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用して所有者を変更します。次の例は、`spectrum_schema`スキーマの所有者を `newowner` に変更します。

```
alter schema spectrum_schema owner to newowner;
```

Redshift Spectrum クエリを実行するには、次のアクセス権限が必要です。

- スキーマのアクセス権限の使用
- 現在のデータベースに一時テーブルを作成するアクセス権限

次の例では、スキーマ `spectrum_schema` の使用許可を `spectrumusers` ユーザーグループに付与しています。

```
grant usage on schema spectrum_schema to group spectrumusers;
```

次の例では、データベース `spectrumdb` の一時アクセス権限を `spectrumusers` ユーザーグループに付与しています。

```
grant temp on database spectrumdb to group spectrumusers;
```

外部テーブルは、Amazon Redshift、AWS Glue、Amazon Athena、または Apache Hive メタストア内に作成できます。詳細については、AWS Glueデベロッパーガイドの「AWS Glue の使用開始」、Amazon Athena ユーザーガイドの「使用開始」、または Amazon EMR デベロッパーガイドの「Apache Hive」を参照してください。

外部テーブルが 、AWS Glue、Athena、または Hive メタストアで定義されている場合は、最初に外部データベースを参照する外部スキーマを作成します。その後、テーブル名の先頭にスキーマ名を付けることで、Amazon Redshift にテーブルを作成することなしに SELECT ステートメント内の外部テーブルを参照できます。詳細については、「Amazon Redshift Spectrum 用の外部スキーマ」を参照してください。

Amazon Redshift で AWS Glue Data Catalog のテーブルを表示できるようにするには、Amazon Redshift IAM ロールに `glue:GetTable` を追加します。それ以外の場合は、以下のようなエラーが発生する場合があります。

```
RedshiftIamRoleSession is not authorized to perform: glue:GetTable on resource: *;
```

例えば、Athena 外部カタログに `lineitem_athena` という名前の外部テーブルが定義されているとします。この場合、`athena_schema`という名前の外部スキーマを定義し、次の SELECT ステートメントを使ってテーブルにクエリを実行できます。

```
select count(*) from athena_schema.lineitem_athena;
```

Amazon Redshift で外部テーブルを定義するには、CREATE EXTERNAL TABLE コマンドを使用します。外部テーブルステートメントはテーブル列、データファイルの型式、Amazon S3 内でのデータの場所を定義します。Redshift Spectrum は、指定されたフォルダおよびサブフォルダ内のファイルをスキャンします。Redshift Spectrum は、隠しファイル、ファイル名がピリオド、下線、ハッシュマーク (「.」、「_」、「#」) で始まるファイル、またはファイル名がチルド (「~」) で終わるファイルは無視します。

次の例では、SALES という名前のテーブルを `spectrum` という名前の Amazon Redshift 外部スキーマに作成します。データはタブ区切りのテキストファイルになっています。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
```

```
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='172000');
```

外部テーブルを表示するには、SVV_EXTERNAL_TABLESシステムビューに対してクエリを実行します。

疑似列

デフォルトでは、Amazon Redshift は疑似列 ($path、$size および $spectrum_oid) を使用して外部テーブルを作成します。$path 列を選択すると、Amazon S3 のデータファイルへのパスが表示され、$size 列を選択するとクエリによって返された各行のデータファイルのサイズが表示されます。$spectrum_oid 列は、Redshift Spectrum で相関クエリを実行する機能を提供します。例については、「例: Redshift Spectrum での相関サブクエリの実行」を参照してください。列名 ($path、$size および $spectrum_oid) は、二重引用符で囲う必要があります。SELECT * 句は、疑似列を返しません。次の例に示すように、$path、$sizeおよび $spectrum_oid の列名をクエリに明示的に含める必要があります。

```
select "$path", "$size", "$spectrum_oid"
from spectrum.sales_part where saledate = '2008-12-01';
```

セッションの疑似列の作成を無効にするには、spectrum_enable_pseudo_columns設定パラメータを false に設定します。詳細については、「spectrum_enable_pseudo_columns」を参照してください。enable_spectrum_oid を false に設定して、$spectrum_oid擬似列のみを無効にすることもできます。詳細については、「enable_spectrum_oid」を参照してください。ただし、$spectrum_oid疑似列 を無効にすると、Redshift Spectrum での相関クエリのサポートも無効になります。

> **Important**
>
> Redshift Spectrum では、Amazon S3 のデータファイルをスキャンして結果セットのサイズを確認しているため、$size、$path または $spectrum_oid を選択すると料金が発生します。詳細については、Amazon Redshift の料金を参照してください。

疑似列の例

次の例では、外部テーブルの関連データファイルの合計サイズを返します。

```
select distinct "$path", "$size"
from spectrum.sales_part;

 $path                                                                 | $size
-----------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/ |  1644
```

Redshift Spectrum 外部テーブルのパーティション化

データをパーティション化する際は、パーティションキーをフィルタリングすることで Redshift Spectrum がスキャンするデータ量を制限できます。すべてのキーでデータをパーティション化できます。

一般的な方法では、時間に基づいてデータをパーティション化します。例えば、年、月、日、時ごとにパーティション化を行います。ソースのデータが多岐にわたる場合は、データソース ID と日付ごとにパーティション化できます。

次の手順でデータをパーティション化する方法を示します。

データをパーティション化するには

1. パーティションキーに従って Amazon S3 のフォルダにデータを保存します。

   パーティション値ごとにフォルダを 1 つ作成し、パーティションキーとパーティション値を使ってフォルダ名を設定します。たとえば日付でパーティション化する場合、フォルダ名は [`saledate=2017-04-01`] や [`saledate=2017-04-02`] になります。Redshift Spectrum は、パーティションフォルダおよびサブフォルダ内のファイルをスキャンします。Redshift Spectrum は、隠しファイル、ファイル名がピリオド、下線、ハッシュマーク (「.」、「_」、「#」) で始まるファイル、またはファイル名がチルド (「~」) で終わるファイルは無視します。
2. 外部テーブルを作成し、PARTITIONED BY 句でパーティションキーを指定します。

   パーティションキーをテーブル列の名前と同じにすることはできません。SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、または TIMESTAMP データ型を使用できます。
3. パーティションを追加します。

ALTER TABLE ... ADD PARTITION を使用して各パーティションを追加し、パーティション列とキーバリュー、Amazon S3 内のパーティションフォルダの場所を指定します。単一の ALTER TABLE ... ADD ステートメントを使用して複数のパーティションを追加できます。次の例では、'2008-01'と '2008-03' のパーティションを追加します。

```
alter table spectrum.sales_part add
partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-01/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-03/';
```

> **Note**
>
> AWS Glue を使用する場合、単一の ALTER TABLE ステートメントを使用して、最大 100 パーティションまで追加できます。

データのパーティション化の例

次の例では、1 つのパーティションキーでパーティション化された外部テーブルと、2 つのパーティションキーでパーティション化された外部テーブルを作成します。

この例のサンプルデータは、認証済みのすべての AWS ユーザーに読み取りアクセスを許可する Amazon S3 バケットに格納されています。クラスターと外部データファイルは、同じ AWS リージョン 存在する必要があります。サンプルデータバケットは、米国東部 (バージニア北部) リージョン (us-east-1) にあります。Redshift Spectrum を使用してデータにアクセスするには、クラスターも us-east-1 に存在する必要があります。Amazon S3 内のフォルダをリスト表示するには次のコマンドを実行します。

```
aws s3 ls s3://redshift-downloads/tickit/spectrum/sales_partition/
```

```
PRE saledate=2008-01/
PRE saledate=2008-03/
PRE saledate=2008-04/
PRE saledate=2008-05/
PRE saledate=2008-06/
```

```
PRE saledate=2008-12/
```

外部スキーマがない場合は、次のコマンドを実行します。AWS Identity and Access Management (IAM) ロールを Amazon リソースネーム (ARN) に置き換えます。

```
create external schema spectrum
from data catalog
database 'spectrumdb'
iam_role 'arn:aws:iam::123456789012:role/myspectrumrole'
create external database if not exists;
```

例 1: 1 つのパーティションキーによるパーティション化

次の例では、月別にパーティション化された外部テーブルを作成します。

月別にパーティション化された外部テーブルを作成するには、次のコマンドを実行します。

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate char(10))
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='172000');
```

パーティションを追加するには、次の ALTER TABLE コマンドを実行します。

```
alter table spectrum.sales_part add
partition(saledate='2008-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'
```

```
partition(saledate='2008-03')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/'

partition(saledate='2008-04')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
```

パーティション化されたテーブルからデータを選択するには、次のクエリを実行します。

```
select top 5 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
   4124 | 21179.00
   1924 | 20569.00
   2294 | 18830.00
   2260 | 17669.00
   6032 | 17265.00
```

外部テーブルパーティションを表示するには、SVV_EXTERNAL_PARTITIONSシステムビューにクエリを実行します。

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values      | location

-----------+------------+-------------
+-------------------------------------------------------------------
spectrum   | sales_part | ["2008-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-03"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
```

例 2: 複数のパーティションキーによるパーティション化

date と eventid でパーティション化された外部テーブルを作成するには、次のコマンドを実行します。

```
create external table spectrum.sales_event(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (salesmonth char(10), event integer)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/salesevent/'
table properties ('numRows'='172000');
```

パーティションを追加するには、次の ALTER TABLE コマンドを実行します。

```
alter table spectrum.sales_event add
partition(salesmonth='2008-01', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=101/'

partition(salesmonth='2008-01', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=102/'

partition(salesmonth='2008-01', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-01/
event=103/'

partition(salesmonth='2008-02', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=101/'

partition(salesmonth='2008-02', event='102')
```

```
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=102/'

partition(salesmonth='2008-02', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-02/
event=103/'

partition(salesmonth='2008-03', event='101')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=101/'

partition(salesmonth='2008-03', event='102')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=102/'

partition(salesmonth='2008-03', event='103')
location 's3://redshift-downloads/tickit/spectrum/salesevent/salesmonth=2008-03/
event=103/';
```

次のクエリを実行するには、パーティション化されたテーブルからデータを選択します。

```
select spectrum.sales_event.salesmonth, event.eventname,
 sum(spectrum.sales_event.pricepaid)
from spectrum.sales_event, event
where spectrum.sales_event.eventid = event.eventid
  and salesmonth = '2008-02'
 and (event = '101'
 or event = '102'
 or event = '103')
group by event.eventname, spectrum.sales_event.salesmonth
order by 3 desc;
```

```
salesmonth | eventname       | sum
-----------+-----------------+--------
2008-02    | The Magic Flute | 5062.00
2008-02    | La Sonnambula   | 3498.00
2008-02    | Die Walkure     |  534.00
```

外部テーブル列を ORC 列にマッピングする

Amazon Redshift Spectrum 外部テーブルを使用して、ORC 形式のファイルからデータをクエリします。最適化された行列 (ORC) 形式は、ネストデータ構造をサポートする列指向ストレージファイル

形式です。ネストデータをクエリする方法の詳細については、[Amazon Redshift Spectrum を使用したネストデータのクエリ]を参照してください。

ORC ファイルのデータを参照する外部テーブルを作成する場合、外部テーブルの各列を ORC データにある列にマッピングします。それを行うには、次のいずれかの方法を使用します。

- 位置によるマッピング
- 列名によるマッピング

列名によるマッピングがデフォルトです。

位置によるマッピング

位置マッピングでは、外部テーブルで統合された最初の列は ORC データファイルの最初の列にマッピングし、2 番目は 2 番目に、のようになります。位置によるマッピングでは、外部テーブルと ORC ファイルの列の順序が一致する必要があります。列の順序が一致しない場合、名前で列をマッピングすることができます。

> **Important**
>
> 以前のリリースでは、Redshift Spectrum は位置によるマッピングをデフォルトで使用しました。既存のテーブルに引き続き位置マッピングを使用する必要がある場合は、次の例に示すように、テーブルプロパティ `orc.schema.resolution` を `position` に設定します。
>
> ```
> alter table spectrum.orc_example
> set table properties('orc.schema.resolution'='position');
> ```

たとえば、テーブル `SPECTRUM.ORC_EXAMPLE` は次のように定義されています。

```
create external table spectrum.orc_example(
int_col int,
float_col float,
nested_col struct<
  "int_col" : int,
  "map_col" : map<int, array<float >>
   >
) stored as orc
location 's3://example/orc/files/';
```

テーブル構造は次のように抽象化することができます。

```
• 'int_col' : int
• 'float_col' : float
• 'nested_col' : struct
   o 'int_col' : int
   o 'map_col' : map
      - key : int
      - value : array
         - value : float
```

基盤となる ORC ファイルには次の構造があります。

```
• ORC file root(id = 0)
   o 'int_col' : int (id = 1)
   o 'float_col' : float (id = 2)
   o 'nested_col' : struct (id = 3)
      - 'int_col' : int (id = 4)
      - 'map_col' : map (id = 5)
         - key : int (id = 6)
         - value : array (id = 7)
            - value : float (id = 8)
```

この例では、外部テーブルの各列を ORC ファイルにある列に厳密に位置によってマッピングできます。以下にマッピングを示します。

外部テーブル列名	ORC 列 ID	ORC 列名
int_col	1	int_col
float_col	2	float_col
nested_col	3	nested_col
nested_col.int_col	4	int_col
nested_col.map_col	5	map_col
nested_col.map_col.key	6	NA
nested_col.map_col.value	7	NA

外部テーブル列名	ORC 列 ID	ORC 列名
nested_col.map_col.value.item	8	NA

列名によるマッピング

名前マッピングを使用して、外部テーブルの列を同じレベル、同じ名前で ORC ファイルの名前を付けられた列にマッピングします。

たとえば、前の例、SPECTRUM.ORC_EXAMPLEのテーブルを次のファイル構造を使用する ORC ファイルにマッピングするとします。

```
• ORC file root(id = 0)
   o 'nested_col' : struct (id = 1)
      - 'map_col' : map (id = 2)
         - key : int (id = 3)
         - value : array (id = 4)
            - value : float (id = 5)
      - 'int_col' : int (id = 6)
   o 'int_col' : int (id = 7)
   o 'float_col' : float (id = 8)
```

位置マッピングを使用して、Redshift Spectrum は以下のマッピングを試行します。

外部テーブル列名	ORC 列 ID	ORC 列名
int_col	1	struct
float_col	7	int_col
nested_col	8	float_col

前述の位置マッピングでテーブルに対してクエリを実行すると、構造が異なるため SELECT コマンドはタイプ検証で失敗します。

列名マッピングを使用して、同じ外部テーブルを前の例に示されている両方のファイル構造にマッピングできます。テーブル列 int_col、float_col、nested_colは、ORC ファイルにある同じ名前の列に列名でマッピングします。外部テーブルにある nested_col という名前の列

は、map_colと int_col という名前のサブ列がある struct 列です。サブ列も列名で ORC ファイルにある対応する列に正しくマッピングします。

Apache Hudi で管理されるデータの外部テーブルの作成

Apache Hudi 書き込みコピー (CoW) 形式のデータをクエリするには、Amazon Redshift Spectrum の外部テーブルを使用できます。Hudi Copy On Write テーブルは、Amazon S3 に保存されている Apache Parquet ファイルのコレクションです。Apache Hudi バージョン 0.5.2、0.6.0、0.7.0、0.8.0、0.9.0、0.10.0、0.10.1、0.11.0、0.11.1 で、挿入、削除、アップサート書き込みオペレーションによって作成および変更された、Copy On Write (CoW) テーブルを読み取ることができます。例えば、ブートストラップテーブルはサポートされていません。詳細については、オープンソースの Apache Hudi ドキュメントから書き込みテーブルでコピーを参照してください。

Hudi CoW 形式のデータを参照する外部テーブルを作成する場合、外部テーブルの各列を Hudi データにある列にマッピングします。マッピングは列ごとに行われます。

パーティション分割された Hudi テーブルおよびパーティション分割されていない Hudi テーブルのデータ定義言語 (DDL) ステートメントは、他の Apache Parquet ファイル形式のステートメントと似ています。Hudi テーブルの場合、INPUTFORMAT を org.apache.hudi.hadoop.HoodieParquetInputFormat として定義します。LOCATION パラメータは、.hoodie フォルダを含む Hudi テーブルベースフォルダを指している必要があります。これは、Hudi コミットタイムラインを確立するために必要です。場合によっては、Hudi テーブルに対する SELECT オペレーションが失敗し、有効な Hudi コミットのタイムラインが見つかりませんというメッセージが表示されることがあります。その場合は、.hoodieフォルダが正しい場所にあり、有効な Hudi コミットのタイムラインが含まれているかどうかを確認してください。

> **Note**
>
> Apache Hudi 形式は、AWS Glue Data Catalogを使用する場合にのみサポートされます。Apache Hive メタストアを外部カタログとして使用する場合は、サポートされません。

パーティション分割されていないテーブルを定義する DDL の形式を次に示します。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
```

```
LOCATION 's3://s3-bucket/prefix'
```

パーティション分割されたテーブルを定義する DDL の形式を次に示します。

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://s3-bucket/prefix'
```

パーティション分割された Hudi テーブルにパーティションを追加するには、ALTER TABLE ADD PARTITION コマンドを実行します。ここで、`LOCATION` パラメータは、パーティションに属するファイルを含む Amazon S3 サブフォルダを指します。

パーティションを追加する DDL の形式を以下に示します。

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION 's3://s3-bucket/prefix/partition-path'
```

Delta Lake で管理されるデータの外部テーブルの作成

Delta Lake テーブルのデータをクエリするには、Amazon Redshift Spectrum 外部テーブルを使用できます。

Redshift Spectrum から Delta Lake テーブルにアクセスするには、クエリの前にマニフェストを生成します。Delta Lake マニフェストには、Delta Lake テーブルの一貫したスナップショットを構成するファイルのリストが含まれています。パーティション化されたテーブルには、パーティションごとに 1 つのマニフェストがあります。Delta Lake テーブルは、Amazon S3 に保存されている Apache Parquet ファイルのコレクションです。詳細については、オープンソースの Delta Lake のドキュメントから Delta Lake を参照してください。

Delta Lake テーブルのデータを参照する外部テーブルを作成する場合、外部テーブルの各列を Delta Lake テーブルにある列にマッピングします。マッピングは列名ごとに行われます。

パーティション分割された、パーティション分割されていない Delta Lake テーブルの DDL は、他の Apache Parquet ファイル形式の場合と同様です。Delta Lake テーブルの場合、`INPUTFORMAT`

を org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat として、OUTPUTFORMAT を org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat として定義します。LOCATION パラメータは、テーブルベースフォルダ内のマニフェストフォルダを指している必要があります。Delta Lake テーブルに対する SELECT オペレーションが失敗した場合、考えられる理由については「Delta Lake テーブルの制限事項とトラブルシューティング」を参照してください。

パーティション分割されていないテーブルを定義する DDL の形式を次に示します。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket/prefix/_symlink_format_manifest'
```

パーティション分割されたテーブルを定義する DDL の形式を次に示します。

```
CREATE EXTERNAL TABLE tbl_name (columns)
PARTITIONED BY(pcolumn1 pcolumn1-type[,...])
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://s3-bucket>/prefix/_symlink_format_manifest'
```

パーティション化された Delta Lake テーブルにパーティションを追加するには、ALTER TABLE ADD PARTITION コマンドを実行します。ここで、LOCATION パラメータは、パーティションのマニフェストを含む Amazon S3 サブフォルダを指します。

パーティションを追加する DDL の形式を以下に示します。

```
ALTER TABLE tbl_name
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path'
```

または、Delta Lake マニフェストファイルを直接指す DDL を実行します。

```
ALTER TABLE tbl_name
```

```
ADD IF NOT EXISTS PARTITION(pcolumn1=pvalue1[,...])
LOCATION
's3://s3-bucket/prefix/_symlink_format_manifest/partition-path/manifest'
```

Delta Lake テーブルの制限事項とトラブルシューティング

Redshift Spectrum から Delta Lake テーブルをクエリする場合は、次の点を考慮してください。

- マニフェストが存在しなくなったスナップショットまたはパーティションを指している場合、新しい有効なマニフェストが生成されるまでクエリは失敗します。例えば、これは、基になるテーブルに対する VACUUM オペレーションの結果である可能性があります。
- Delta Lake マニフェストは、パーティションレベルの整合性のみを提供します。

次のテーブルでは、Delta Lake テーブルをクエリするときに特定のエラーが発生する可能性のある理由について説明します。

エラーメッセージ	考えられる理由
バケットs3-bucket-1 の Delta Lake マニフェストに、バケット s3-bucket-2 のエントリを含めることはできません。	マニフェストエントリは、指定されたバケットとは異なる Amazon S3 バケット内のファイルを指しています。
Delta Lake ファイルは、同じフォルダにあると予想されます。	マニフェストエントリは、指定されたものとは異なる Amazon S3 プレフィックスを持つファイルを指します。
Delta Lake マニフェストのマニフェストパスに表示されるファイルのファイル名が見つかりませんでした。	マニフェストに表示されているファイルが Amazon S3 で見つかりませんでした。
Delta Lake マニフェストの取得中にエラーが発生しました。	マニフェストは Amazon S3 では見つかりませんでした。
無効な S3 パスです。	マニフェストファイルのエントリが有効な Amazon S3 パスではないか、マニフェストファイルが破損しています。

Amazon Redshift での Apache Iceberg テーブルの使用

このトピックでは、Redshift Spectrum または Redshift Serverless で Apache Iceberg 形式のテーブルを使用する方法について説明します。Apache Iceberg は、巨大な分析テーブル用の高パフォーマンス形式です。

Redshift Spectrum または Redshift Serverless を使用して、AWS Glue Data Catalogでカタログ化されている Apache Iceberg テーブルにクエリを実行できます。Apache Iceberg は、データレイク用のオープンソースのテーブル形式です。詳細については、Apache Iceberg ドキュメントで「[Apache Iceberg]」を参照してください。

Amazon Redshift では、Apache Iceberg テーブルに対してクエリを実行する際に、トランザクションの一貫性を保ちます。Amazon Redshift を使用してクエリを実行しながら、Amazon Athena や Amazon EMR などの ACID (原子性、一貫性、分離性、持続性) 準拠のサービスを通じてテーブル内のデータを操作できます。Amazon Redshift では、Apache Iceberg メタデータに保存されているテーブル統計を使用してクエリプランを最適化し、クエリ処理中のファイルスキャンを減らすことができます。Amazon Redshift SQL を使用すると、Redshift テーブルをデータレイクテーブルと結合できます。

Amazon Redshift で Iceberg テーブルの使用を開始するには

1. Amazon Athena や Amazon EMR などの互換性のあるサービスを使用して AWS Glue Data Catalogのデータベースに Apache Iceberg テーブルを作成します。Athena を使用して Iceberg テーブルを作成するには、「Amazon Athena ユーザーガイド」の「Apache Iceberg テーブルの使用」を参照してください。
2. Amazon Redshift クラスターまたは Redshift Serverless ワークグループを作成し、データレイクへのアクセスを許可する IAM ロールを関連付けます。クラスターやワークグループの作成方法については、「Amazon Redshift 入門ガイド」の「Amazon Redshift でプロビジョニングされたデータウェアハウス」と「Amazon Redshift Serverless データウェアハウスの使用を開始」を参照してください。
3. クエリエディタ v2 またはサードパーティの SQL クライアントを使用して、クラスターまたはワークグループに接続します。クエリエディタ v2 を使用して接続する方法については、「Amazon Redshift 管理ガイド」の「SQL クライアントツールを使用して Amazon Redshift データウェアハウスに接続する」を参照してください。
4. Iceberg テーブルを含む特定のデータカタログデータベース用の外部スキーマを Amazon Redshift データベースに作成します。外部スキーマの作成の詳細については、「Amazon Redshift Spectrum 用の外部スキーマ」を参照してください。

5. SQL クエリを実行して、作成した外部スキーマ内の Iceberg テーブルにアクセスします。

Amazon Redshift で Apache Iceberg テーブルを使用する際の考慮事項

Amazon Redshift で Iceberg テーブルを使用する場合は、以下の点を考慮してください。

- Iceberg バージョンのサポート — Amazon Redshift は、以下のバージョンの Iceberg テーブルに対するクエリの実行をサポートしています。
  - バージョン 1 は、イミュータブルなデータファイルを使用して、大規模な分析テーブルをどのように管理するかを定義します。
  - バージョン 2 は、行レベルの更新と削除をサポートするとともに、既存のデータファイルを変更せずに維持し、削除ファイルを使用してテーブルデータの変更を処理する機能を追加します。

  バージョン 1 とバージョン 2 のテーブルの違いについては、Apache Iceberg ドキュメントで「フォーマットバージョンの変更」を参照してください。
- クエリのみ — Amazon Redshift は Apache Iceberg テーブルへの読み取り専用アクセスをサポートしています。また、トランザクションの一貫性のある選択クエリをサポートしています。Amazon Athena などのサービスを使用して、AWS Glue Data Catalogで Iceberg テーブルのスキーマを定義および更新できます。
- パーティションの追加 — Apache Iceberg テーブルにパーティションを手動で追加する必要はありません。Apache Iceberg テーブルの新しいパーティションは Amazon Redshift によって自動的に検出されるため、テーブル定義のパーティションを更新するための手動操作は必要ありません。パーティション仕様の変更も、ユーザーの介入なしにクエリに自動的に適用されます。
- Amazon Redshift への Iceberg データのインジェスト — INSERT INTO コマンドまたは CREATE TABLE AS コマンドを使用して、Iceberg テーブルからローカルの Amazon Redshift テーブルにデータをインポートできます。現在、COPY コマンドを使用して Apache Iceberg テーブルの内容をローカルの Amazon Redshift テーブルにインジェストすることはできません。
- マテリアライズドビュー — Amazon Redshift の他の外部テーブルと同じように、Apache Iceberg テーブルでマテリアライズドビューを作成できます。他のデータレイクテーブル形式の場合と同じ考慮事項が Apache Iceberg テーブルにも当てはまります。データレイクテーブルのインクリメンタル更新、自動更新、自動クエリ書き換え、自動 MV は現在サポートされていません。
- AWS Lake Formation のきめ細かなアクセス制御 — Amazon Redshift は、Apache Iceberg テーブルに対する AWS Lake Formation のきめ細かなアクセス制御をサポートしています。
- ユーザー定義のデータ処理パラメータ — Amazon Redshift は、Apache Iceberg テーブルに対するユーザー定義のデータ処理パラメータをサポートしています。既存のファイルに対するユーザー定

義のデータ処理パラメータを使用して、外部テーブルでクエリされるデータを調整し、スキャンエラーを回避します。これらのパラメータは、テーブルスキーマとファイル内の実際のデータとの不一致を処理する機能を提供します。Apache Iceberg テーブルに対してもユーザー定義のデータ処理パラメータを使用できます。

- タイムトラベルクエリ — タイムトラベルクエリは現在、Apache Iceberg テーブルに対してサポートされていません。
- 料金 — クラスターから Iceberg テーブルにアクセスすると、Redshift Spectrum の料金が請求されます。ワークグループから Iceberg テーブルにアクセスすると、Redshift Serverless の料金が請求されます。Redshift Spectrum と Redshift Serverless の料金の詳細については、「Amazon Redshift の料金」を参照してください。

Apache Iceberg テーブルでサポートされているデータ型

このトピックでは、Redshift Spectrum が Apache Iceberg 形式のテーブルから読み取ることができるサポートされているデータ型について説明します。

Amazon Redshift は、以下のデータ型が含まれている Iceberg テーブルをクエリできます。

```
binary
boolean
date
decimal
double
float
int
list
long
map
string
struct
timestamp without time zone
```

Iceberg のデータ型の詳細については、Apache Iceberg ドキュメントで Iceberg のスキーマを参照してください。

次の表に、Amazon Redshift のデータ型と Iceberg テーブルのデータ型の関係を示します。

Iceberg の型	Amazon Redshift の型	メモ
boolean	boolean	
-	tinyint	Amazon Redshift の Iceberg テーブルではサポートされていません。
-	smallint	Amazon Redshift の Iceberg テーブルではサポートされていません。
int	int	Amazon Redshift の SQL ステートメントの場合、これは INTEGER 型です。
long	bigint	
double	double	
float	float	
decimal(P , S)	decimal(P , S)	P は精度、S はスケールです。
-	char	Redshift Spectrum の Apache Iceberg テーブルではサポートされていません。
string	string	Amazon Redshift の SQL ステートメントの場合、これは VARCHAR 型です。
binary	binary	
date	date	
time	-	
timestamp	timestamp	

Iceberg の型	Amazon Redshift の型	メモ
timestamptz	-	
list<E>	array	
map<K,V>	map	
struct<...>	struct	
fixed(L)	-	現在、fixed(L) 型は Redshift Spectrum ではサポートされていません。

Amazon Redshift のデータ型の詳細については、「データ型」を参照してください。

Amazon Redshift Spectrum クエリパフォーマンス

このトピックでは、Redshift Spectrum クエリのパフォーマンスを向上させる方法について説明します。

クエリプランを参照し、Amazon Redshift Spectrum レイヤーにプッシュされているステップを確認します。

次のステップは、Redshift Spectrum クエリに関連しています。

- S3 Seq Scan
- S3 HashAggregate
- S3 Query Scan
- Seq Scan PartitionInfo
- Partition Loop

次の例では、外部テーブルとローカルテーブルを結合するクエリのクエリプランを示します。Amazon S3 内のデータに対して実行された S3 Seq Scan および S3 HashAggregate の各ステップをメモします。

```
explain
select top 10 spectrum.sales.eventid, sum(spectrum.sales.pricepaid)
from spectrum.sales, event
where spectrum.sales.eventid = event.eventid
and spectrum.sales.pricepaid > 30
group by spectrum.sales.eventid
order by 2 desc;
```

```
QUERY PLAN

-----------------------------------------------------------------------------
XN Limit  (cost=1001055770628.63..1001055770628.65 rows=10 width=31)

  ->  XN Merge  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

        Merge Key: sum(sales.derived_col2)

        ->  XN Network  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

              Send to leader

              ->  XN Sort  (cost=1001055770628.63..1001055770629.13 rows=200 width=31)

                    Sort Key: sum(sales.derived_col2)

                    ->  XN HashAggregate  (cost=1055770620.49..1055770620.99 rows=200
 width=31)
```

```
                              ->  XN Hash Join DS_BCAST_INNER  (cost=3119.97..1055769620.49
rows=200000 width=31)

                                    Hash Cond: ("outer".derived_col1 = "inner".eventid)


                                    ->  XN S3 Query Scan sales  (cost=3010.00..5010.50
rows=200000 width=31)

                                          ->  S3 HashAggregate  (cost=3010.00..3010.50
rows=200000 width=16)

                                                ->  S3 Seq Scan spectrum.sales
location:"s3://redshift-downloads/tickit/spectrum/sales" format:TEXT
(cost=0.00..2150.00 rows=172000 width=16)
                                                      Filter: (pricepaid > 30.00)


                                    ->  XN Hash  (cost=87.98..87.98 rows=8798 width=4)


                                          ->  XN Seq Scan on event  (cost=0.00..87.98
rows=8798 width=4)
```

クエリプラン内の次の要素をメモします。

- `S3 Seq Scan` ノードは、フィルタ `pricepaid > 30.00` が Redshift Spectrum レイヤーで処理されたことを示します。

  `XN S3 Query Scan` ノードの下にあるフィルターノードは、データ上部の Amazon Redshift の述語処理が Redshift Spectrum レイヤーから返されたことを示します。
- `S3 HashAggregate` ノードは、Redshift Spectrum レイヤーで句 (`group by spectrum.sales.eventid`) ごとにグループの集計が行われたことを示します。

Redshift Spectrum のパフォーマンスは次の方法で向上させることができます。

- Apache Parquet 形式のデータファイルを使用します。Parquet は列形式でデータを保存するため、Redshift Spectrum は不要な列をスキャンから削除できます。データがテキストファイル形式である場合、Redshift Spectrum はファイル全体をスキャンする必要があります。
- 複数のファイルを使用して並列処理用に最適化します。ファイルサイズ を 64 MB 以上のまま維持します。ファイルをほぼ同じサイズにし、データサイズスキューを回避します。Apache Parquet

ファイルと設定の推奨事項については、「Apache Parquet ドキュメント」の「File Format: Configurations」を参照してください。

- クエリで使用する列を可能な限り少なくします。
- 大きなファクトテーブルは Amazon S3 に置き、使用頻度の高い小さなディメンションテーブルはローカルの Amazon Redshift データベースに置きます。
- TABLE PROPERTIES numRows パラメータを設定して、外部テーブル統計を更新します。CREATE EXTERNAL TABLE または ALTER TABLE に使用し、TABLE PROPERTIES numRows パラメータを設定して、テーブルの行数を反映できるようになりました。Amazon Redshift は、外部テーブルを分析して、クエリオプティマイザがクエリプランを生成するために使用するテーブル統計を生成することはありません。外部テーブルに対してテーブル統計が設定されていない場合、Amazon Redshift はクエリ実行プランを生成します。Amazon Redshift は、外部テーブルの方が大きくローカルテーブルの方が小さいという前提に基づいてこのプランを生成します。
- Amazon Redshift クエリプランナーは、述語と集計を可能な限り Redshift Spectrum クエリレイヤーにプッシュします。Amazon S3 から大量のデータが返されると、クラスターのリソースによって処理が制限されます。Redshift Spectrum は自動的に拡張してサイズの大きいリクエストを処理します。このように、Redshift Spectrum レイヤーに処理をプッシュできる場合は常に全体的なパフォーマンスは向上します。
- Redshift Spectrum レイヤーにプッシュできるフィルタリングと集計を使用するようにクエリを書き込んでください。

  以下に、Redshift Spectrum レイヤーにプッシュできるオペレーションの例を示します。

  - GROUP BY 句
  - 比較条件とパターンマッチング条件 (LIKE など)
  - COUNT、SUM、AVG、MIN、MAX などの集計関数。
  - 文字列関数。

  Redshift Spectrum レイヤーにプッシュできないオペレーションには、DISTINCT や ORDER BY が含まれます。
- パーティションを使用してスキャンされるデータを制限します。最も一般的な述語に基づいてデータをパーティション化し、パーティション列をフィルタリングしてパーティションを取り除きます。詳細については、「Redshift Spectrum 外部テーブルのパーティション化」を参照してください。

SVL_S3PARTITION にクエリを実行し、パーティションの総数と適格なパーティションの数を表示します。

- AWS Glue の統計ジェネレータを使用して、AWS Glue Data Catalog テーブルの列レベルの統計を計算します。AWS Glue がデータカタログ内のテーブルの統計を生成すると、Amazon Redshift Spectrum はその統計を自動的に使用してクエリプランを最適化します。AWS Glue を使用して列レベルの統計を計算する方法の詳細については、「AWS Glue 開発者ガイド」の「列統計の処理」を参照してください「。

データ処理オプション

このトピックでは、Redshift Spectrum が予期しない形式のデータを処理する方法を設定する方法について説明します。

外部テーブルの作成時にテーブルパラメータを設定して、外部テーブルでクエリされるデータを調整することができます。設定しなければ、スキャンエラーが発生する可能性があります。詳細については、「CREATE EXTERNAL TABLE」で「TABLE PROPERTIES」を参照してください。例については、「データ処理の例」を参照してください。エラーのリストについては、「SVL_SPECTRUM_SCAN_ERROR」を参照してください。

外部テーブルの作成時に以下の TABLE PROPERTIES を設定して、外部テーブルでクエリされるデータの入力処理を指定できます。

- `column_count_mismatch_handling` を使用して、ファイルに、外部テーブル定義で指定された列数よりも行の値が少ないか、または多いかを識別します。
- `invalid_char_handling` は、VARCHAR、CHAR、および文字列データが含まれる列内での無効な文字の入力処理を指定します。`invalid_char_handling` に REPLACE を指定するときは、使用する置換文字を指定できます。
- `numeric_overflow_handling` は、整数と小数のデータが含まれる列内でのキャストオーバーフロー処理を指定します。
- `surplus_bytes_handling` は、VARBYTE データを含む列の余剰バイトの入力処理を指定します。
- `surplus_char_handling` は、VARCHAR、CHAR、および文字列データが含まれる列内での余剰文字の入力処理を指定します。

最大エラー数を超えるクエリをキャンセルする設定オプションを設定できます。詳細については、「spectrum_query_maxerror」を参照してください。

例: Redshift Spectrum での相関サブクエリの実行

このトピックでは、Redshift Spectrum で相関サブクエリを実行する方法を説明します。相関サブクエリは、外部クエリの値を使用するクエリです。

Redshift Spectrum で相関サブクエリを実行することができます。`$spectrum_oid` 列は、Redshift Spectrum で相関クエリを実行する機能を提供します。相関サブクエリを実行するには、擬似列 `$spectrum_oid` を有効にする必要がありますが、SQL ステートメントには表示されません。詳細については、「疑似列」を参照してください。

この例で外部スキーマと外部テーブルを作成するには、「Amazon Redshift Spectrum の開始方法」を参照してください。

Redshift Spectrum の相関サブクエリの例を次に示します。

```
select *
from myspectrum_schema.sales s
where exists
( select *
from myspectrum_schema.listing l
where l.listid = s.listid )
order by salesid
limit 5;
```

```

salesid  listid   sellerid   buyerid   eventid   dateid  qtysold  pricepaid
 commission   saletime
1        1        36861      21191     7872      1875    4        728          109.2
    2008-02-18 02:36:48
2        4        8117       11498     4337      1983    2        76           11.4
    2008-06-06 05:00:16
3        5        1616       17433     8647      1983    2        350          52.5
    2008-06-06 08:26:17
4        5        1616       19715     8647      1986    1        175          26.25
    2008-06-09 08:38:52
5        6        47402      14115     8240      2069    2        154          23.1
    2008-08-31 09:17:02
```

Amazon Redshift Spectrum でのメトリクス

このトピックでは、Redshift Spectrum クエリのモニタリングに使用できるシステムビューについて説明します。

次のシステムビューを使用して、Amazon Redshift Spectrum クエリをモニタリングできます。

- SVL_S3QUERY

  SVL_S3QUERY システムビューを使用して、セグメントおよびノードスライスレベルで Redshift Spectrum クエリ (S3 クエリ) の詳細を確認します。
- SVL_S3QUERY_SUMMARY

  SVL_S3QUERY_SUMMARY ビューを使用して、システムで実行されたすべての Amazon Redshift Spectrum クエリ (S3 クエリ) 概要を取得します。

以下に、SVL_S3QUERY_SUMMARY の確認事項を示します。

- Redshift Spectrum クエリで処理されたファイル数。
- Amazon S3 からスキャンされたバイト数。Redshift Spectrum クエリの料金は、Amazon S3 からスキャンされたデータ量に反映されます。
- Redshift Spectrum レイヤーからクラスターに返されたバイト数。大量のデータが返されると、システムパフォーマンスに影響が及ぶ可能性があります。
- Redshift Spectrum リクエストの最長時間と平均時間。実行時間の長いリクエストはボトルネックの可能性があります。

Amazon Redshift Spectrum でのクエリのトラブルシューティング

このトピックは、Amazon Redshift Spectrum クエリで発生する可能性がある一般的な問題のリファレンスです。

Redshift Spectrum クエリで生成されたエラーを表示するには、SVL_S3LOGシステムテーブルのクエリを実行します。

トピック

- 再試行数の超過
- スロットリングされたアクセス

- リソースの制限を超えました
- パーティション化されたテーブルに行が返されない
- 権限エラー
- データ形式に互換性がない
- Amazon Redshift で Hive DDL を使用する際の構文エラー
- 一時テーブルを作成するアクセス許可
- 無効な範囲
- 無効な Parquet バージョン番号

再試行数の超過

Amazon Redshift Spectrum リクエストがタイムアウトすると、リクエストはキャンセルされ再送信されます。再試行が 5 回失敗するとクエリは失敗し、次のエラーが表示されます。

```
error:  Spectrum Scan Error: Retries exceeded
```

次の原因が考えられます。

- ファイルサイズが大きい (1 GB 超) Amazon S3 のファイルサイズを確認し、サイズの大きなファイルおよびファイルサイズスキューを探します。サイズの大きなファイルは小さなファイル (100 MB～1 GB) に分割します。ほぼ同じファイルサイズになるようにしてください。
- ネットワークスループットの遅延。後でクエリを試してください。

スロットリングされたアクセス

Amazon Redshift Spectrum は、他の AWS のサービスにおけるクオータの対象となります。使用率が高い場合、Redshift Spectrum リクエストの処理速度を低下させる必要がある場合があり、その結果次のエラーが発生します。

```
error:  Spectrum Scan Error: Access throttled
```

次の 2 種類のスロットリングが発生します。

- Amazon S3 によって制限されたアクセス。
- AWS KMS によってスロットリングされたアクセス。

エラーコンテキストは、スロットリングのタイプに関する詳細を提供します。次に、このスロットリングの原因と考えられる解決方法を示します。

Amazon S3 によってスロットリングされたアクセス

プレフィックスの読み込みリクエストレートが高すぎる場合、Amazon S3 は Redshift Spectrum のリクエストを抑制する可能性があります。Amazon S3 で達成できる GET/HEAD リクエストレートについては、Amazon Simple Storage Service ユーザーガイドの Amazon S3 のパフォーマンスの最適化を参照してください。Amazon S3 GET/HEAD リクエストレートでは、プレフィックスに対するすべての GET/HEAD リクエストが考慮されるため、同じプレフィックスにアクセスする異なるアプリケーションが合計リクエストレートを共有します。

Redshift Spectrum リクエストが Amazon S3 によって頻繁にスロットリングされる場合は、Redshift Spectrum が Amazon S3 に提供する Amazon S3 GET/HEAD リクエストの数を減らします。これを行うには、小さなファイルを大きなファイルにマージしてみてください。64 MB 以上のファイルサイズの使用が推奨されます。

また、Redshift Spectrum テーブルのパーティション分割も考慮して、早期フィルタリングによる利点を得て、Amazon S3 でアクセスされるファイルの数を減らすことも検討してください。詳細については、「Redshift Spectrum 外部テーブルのパーティション化」を参照してください。

AWS KMS によってスロットリングされたアクセス

サーバー側の暗号化 (SSE-S3 または SSE-KMS) を使用して Amazon S3 にデータを保存する場合は、Redshift Spectrum がアクセスするファイルごとに、AWS KMSに対する API オペレーションを Amazon S3 が呼び出します。これらのリクエストは、暗号化操作のクォータにカウントされます。詳細については、「AWS KMS クォータのリクエスト」を参照してください。SSE-S3 および SSE-KMS の詳細については、Amazon Simple Storage Service ユーザーガイドの「サーバー側の暗号化を使用したデータの保護」および「Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS KMS」を参照してください。

Redshift Spectrum が AWS KMS に対して行うリクエストの数を減らすには、最初に、アクセスされるファイルの数を減らす必要があります。これを行うには、小さなファイルを大きなファイルにマージしてみてください。64 MB 以上のファイルサイズの使用が推奨されます。

Redshift Spectrum のリクエストが AWS KMS によって頻繁にスロットリングされる場合は、暗号化オペレーションの AWS KMS リクエストレートのクォータ引き上げを、申請することを検討してください。クォータ増加をリクエストするには、Amazon Web Services 全般のリファレンス の「AWS のサービス制限」を参照してください。

リソースの制限を超えました

Redshift Spectrum は、リクエストが使用できるメモリ量に上限を強制します。より多くのメモリを必要とする Redshift Spectrum リクエストが失敗し、次のエラーが発生します。

```
error:  Spectrum Scan Error: Resource limit exceeded
```

Redshift Spectrum リクエストがメモリ割り当てをオーバーランさせる原因としては、次の 2 つが考えられます。

- Redshift Spectrum は、小さなチャンクに分割できない大量のデータを処理します。
- 大規模な集約ステップは、Redshift Spectrum によって処理されます。

分割サイズが 128 MB 以下の並列読み取りをサポートするファイル形式を使用することをお勧めします。サポートされているファイル形式とデータファイルの作成に関する一般的なガイドラインについては、「Amazon Redshift Spectrum でクエリ用のデータファイルを作成する」を参照してください。並列読み取りをサポートしないファイル形式または圧縮アルゴリズムを使用する場合は、ファイルサイズを 64 MB ~ 128 MB にすることをお勧めします。

パーティション化されたテーブルに行が返されない

クエリでパーティション化された外部テーブルから行が返されない場合は、パーティションがこの外部テーブルに追加されているかどうかを確認します。Redshift Spectrum は、`ALTER TABLE … ADD PARTITION` を使用して明示的に追加されている Amazon S3 に位置するファイルのみをスキャンします。SVV_EXTERNAL_PARTITIONS ビューにクエリを実行し既存のパーティションを検索します。欠落しているパーティションごとに `ALTER TABLE … ADD PARTITION` を実行します。

権限エラー

クラスターの IAM ロールで Amazon S3 ファイルオブジェクトへのアクセスが許可されていることを確認します。外部データベースが Amazon Athena にある場合は、IAM ロールが Athena リソースにアクセスできるかどうかを確認します。詳細については、「Amazon Redshift Spectrum 用の IAM ポリシー」を参照してください。

データ形式に互換性がない

Apache Parquet などの列ファイル形式では、列タイプはデータとともに埋め込まれます。CREATE EXTERNAL TABLE 定義内の列タイプは、データファイルの列タイプと一致する必要があります。一致しないと、次のようなエラーが発生します。

```
File 'https://s3bucket/location/file has an incompatible Parquet schema
for column ‘s3://s3bucket/location.col1'. Column type: VARCHAR, Par
```

エラーメッセージは、メッセージの長さ制限で切り捨てられている場合もあります。列名と列タイプを含む完全なエラーメッセージを取得するには、SVL_S3LOGシステムビューのクエリを実行します。

以下の例は、最後に実行されたクエリについての SVL_S3LOG をクエリします。

```
select message
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

次に示すのは、完全なエラーメッセージを表示した結果です。

```
                            message
-----------------------------------------------------------------------------------------
Spectrum Scan Error. File 'https://s3bucket/location/file has an incompatible
Parquet schema for column ' s3bucket/location.col1'.
Column type: VARCHAR, Parquet schema:\noptional int64 l_orderkey [i:0 d:1 r:0]\n
```

エラーを修正するには、外部テーブルを変更し、Parquet ファイルの列タイプと一致させます。

Amazon Redshift で Hive DDL を使用する際の構文エラー

Amazon Redshift は、Hive DDL と類似した CREATE EXTERNAL TABLE 用のデータ定義言語 (DDL) をサポートしています。ただし、2 つの DDL タイプは常に代替可能であるとは限りません。Hive DDL をコピーして Amazon Redshift 外部テーブルを作成または変更した場合、構文エラーが発生する可能性があります。以下に Amazon Redshift と Hive DDL の違いの例を示します。

- Hive DDL がダブルクォーテーションマーク (") をサポートしているのに対して、Amazon Redshift はシングルクォーテーションマーク (') を使用します。

- Amazon Redshift は STRING データタイプをサポートしていません。代わりに VARCHAR を使用します。

一時テーブルを作成するアクセス許可

Redshift Spectrum クエリを実行するには、データベースユーザーがデータベースに一時テーブルを作成するアクセス権限を持っている必要があります。次の例では、データベース `spectrumdb` の一時アクセス権限を `spectrumusers` ユーザーグループに付与しています。

```
grant temp on database spectrumdb  to group spectrumusers;
```

詳細については、「GRANT」を参照してください。

無効な範囲

Redshift Spectrum では、クエリの実行中に、外部テーブルに属する Amazon S3 内のファイルが上書きされないことを想定しています。上書きが発生した場合、次のエラーが発生することがあります。

```
Error: HTTP response error code: 416 Message: InvalidRange The requested range is not
 satisfiable
```

このエラーを回避するには、Redshift Spectrum からクエリされている Amazon S3 ファイルに対しての上書きを防止します。

無効な Parquet バージョン番号

Redshift Spectrum は、アクセスする各 Apache Parquet ファイルのメタデータをチェックします。このチェックに失敗すると、次のようなエラーが発生することがあります。

```
File 'https://s3.region.amazonaws.com/s3bucket/location/file has an invalid version
 number
```

チェックが失敗する原因としては、以下の 2 つが考えられます。

- クエリ中に、対象の Parquet ファイルが上書きされた (無効な範囲 を参照)。
- Parquet ファイルが破損している。

チュートリアル: Amazon Redshift Spectrum を使用したネストデータのクエリ

このチュートリアルでは、Redshift Spectrum を使用してネストデータをクエリする方法を示します。ネストデータは、ネストされたフィールドを含むデータです。ネストされたフィールドは、配列、構造体、オブジェクトなど、単一のエンティティとして結合されるフィールドです。

トピック

概要

Amazon Redshift Spectrum では、ファイル形式が Parquet、ORC、JSON、Ion のネストデータのクエリ実行をサポートしています。Redshift Spectrum は、外部テーブルを使用してデータにアクセスします。`struct`、`array`、`map` などの複合データ型を使用して外部テーブルを作成することもできます。

たとえば、`customers` という名前のフォルダ内のデータファイルに、Amazon S3 の以下のデータが含まれるとします。単一のルート要素はありませんが、このサンプルデータの各 JSON オブジェクトはテーブルの行を表します。

```
{"id": 1,
 "name": {"given": "John", "family": "Smith"},
 "phones": ["123-457789"],
 "orders": [{"shipdate": "2018-03-01T11:59:59.000Z", "price": 100.50},
            {"shipdate": "2018-03-01T09:10:00.000Z", "price": 99.12}]
}
{"id": 2,
 "name": {"given": "Jenny", "family": "Doe"},
 "phones": ["858-8675309", "415-9876543"],
 "orders": []
}
```

```
{"id": 3,
 "name": {"given": "Andy", "family": "Jones"},
 "phones": [],
 "orders": [{"shipdate": "2018-03-02T08:02:15.000Z", "price": 13.50}]
}
```

Amazon Redshift Spectrum を使用して、ファイル内のネストデータにクエリを実行できます。以下のチュートリアルでは、Apache Parquet データでの実行方法を紹介します。

前提条件

Redshift Spectrum をまだ使用していない場合は、[Amazon Redshift Spectrum の開始方法](#) のステップに従って行います。

外部スキーマを作成するには、次のコマンドの IAM ロール ARN を、「[IAM ロールを作成する](#)」で作成したロール ARN に置き換えます。次に、SQL クライアントでコマンドを実行します。

```
create external schema spectrum
from data catalog
database 'myspectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myspectrum_role'
create external database if not exists;
```

ステップ 1: ネストデータを含む外部テーブルを作成する

[ソースデータ](#)は、Amazon S3 からダウンロードして表示できます。

このチュートリアル用に外部テーブルを作成するには、次のコマンドを実行します。

```
CREATE EXTERNAL TABLE spectrum.customers (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

前述の例で、外部テーブル (`spectrum.customers`) では、データ型 `struct` および `array` を使用して、ネストデータを含む列を定義しています。Amazon Redshift Spectrum では、ファイル形式が Parquet、ORC、JSON、Ion のネストデータのクエリ実行をサポートしています。`STORED AS`

パラメータは、Apache Parquet ファイルを表す PARQUET です。LOCATION パラメータは、ネストデータまたはファイルを含む Amazon S3 フォルダを参照する必要があります。詳細については、「CREATE EXTERNAL TABLE」を参照してください。

データ型 array および struct は、任意のレベルでネスト化することができます。たとえば、次の例で示すように、toparray という名前の列を定義できます。

```
toparray array<struct<nestedarray:
         array<struct<morenestedarray:
         array<string>>>>>
```

また、次の例の struct で示すように、データ型 x をネスト化することもできます。

```
x struct<a: string,
         b: struct<c: integer,
                   d: struct<e: string>
                  >
        >
```

ステップ 2: SQL 拡張を使用して Amazon S3 のネストデータにクエリを実行する

Redshift Spectrum では、Amazon Redshift SQL 構文に拡張することで、複合型 (array、map、struct) のクエリをサポートしています。

拡張 1: Struct 列へのアクセス

struct 列からデータを抽出するには、フィールド名をパスに連結するドット表記を使用します。たとえば、次のクエリでは、指定された顧客の姓名が返ります。名にアクセスするには、長いパス c.name.given を使用します。姓にアクセスするには、長いパス c.name.family を使用します。

```
SELECT c.id, c.name.given, c.name.family
FROM   spectrum.customers c;
```

前述のクエリでは、次のデータが返ります。

```
id | given | family
---|-------|-------
1  | John  | Smith
2  | Jenny | Doe
```

```
3  | Andy  | Jones
(3 rows)
```

struct は、別の struct 列にすることができます。つまり、別の struct の列を任意のレベルで使用できます。このように深くネストされた struct 列にアクセスするパスは、任意的に長くすることができます。たとえば、次の例の x 列については、定義を参照してください。

```
x struct<a: string,
         b: struct<c: integer,
                      d: struct<e: string>
                  >
        >
```

e のデータに x.b.d.e としてアクセスできます。

拡張 2: FROM 句の配列範囲

array 列 (および拡張の map 列) からデータを抽出するには、テーブル名ではなく、array 句の FROM 列を指定します。この拡張は、メインクエリの FROM 句だけでなく、サブクエリの FROM 句にも適用されます。

array 要素を位置 (例: c.orders[0]) で参照できます (プレビュー)。

次のユースケースで説明するように、arrays を joins と組み合わせることにより、さまざまな種類のネスト解除を行うことができます。

内部結合を使用したネスト解除

次のクエリでは、注文を含む顧客の顧客 ID と出荷日を選択します。FROM 句の SQL 拡張 c.orders o は、エイリアス c によって異なります。

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c, c.orders o
```

注文を含む顧客 c ごとに、FROM 句によって、顧客 o の注文 c 単位で返ります。その行は、顧客行 c と注文行 o を組み合わせたものです。次に、SELECT 句を使用して、c.id および o.shipdate を維持します。結果は次のとおりです。

```
id|      shipdate
--|----------------------
1 |2018-03-01  11:59:59
```

```
1 |2018-03-01  09:10:00
3 |2018-03-02  08:02:15
(3 rows)
```

エイリアス c では顧客フィールド、o では注文フィールドにアクセスすることができます。

セマンティクスは、標準的な SQL と似ています。FROM 句は、次のネステッドループの実行と考えることができます。続いて、SELECT 句を使用して、出力するフィールドを選択します。

```
for each customer c in spectrum.customers
  for each order o in c.orders
     output c.id and o.shipdate
```

したがって、注文のない顧客は、結果に表示されません。

また、これは、FROM テーブルおよび JOIN 配列を使用して、customers を実行する orders 句と考えることができます。実際、次の例に示すように、クエリを記述することもできます。

```
SELECT c.id, o.shipdate
FROM   spectrum.customers c INNER JOIN c.orders o ON true
```

> **Note**
>
> c という名前のテーブルを持つスキーマ orders が存在する場合、c.orders は、テーブル orders を参照します。customers の配列の列は参照されません。

左結合を使用したネスト解除

次のクエリでは、顧客名とその注文をすべて出力します。顧客が注文を行っていない場合でも、顧客名は返ります。ただし、この場合、次の Jenny Doe の例に示すように、注文列は NULL です。

```
SELECT c.id, c.name.given, c.name.family, o.shipdate, o.price
FROM   spectrum.customers c LEFT JOIN c.orders o ON true
```

前述のクエリでは、次のデータが返ります。

```
id  |  given   | family  |    shipdate          | price
----|----------|---------|----------------------|--------
 1  |  John    | Smith   | 2018-03-01  11:59:59 | 100.5
```

```
 1  |  John   | Smith   | 2018-03-01  09:10:00 |  99.12
 2  |  Jenny  | Doe     |                      |
 3  |  Andy   | Jones   | 2018-03-02  08:02:15 |  13.5
 (4 rows)
```

拡張 3: エイリアスを使用して Scalars の配列に直接アクセスする

FROM 句のエイリアス p がスカラーの配列範囲にある場合、このクエリでは p の値を p として参照します。たとえば、次のクエリでは、顧客名と電話番号のペアが生成されます。

```
SELECT c.name.given, c.name.family, p AS phone
FROM   spectrum.customers c LEFT JOIN c.phones p ON true
```

前述のクエリでは、次のデータが返ります。

```
given  |  family  |  phone
-------|----------|-----------
John   | Smith    | 123-4577891
Jenny  | Doe      | 858-8675309
Jenny  | Doe      | 415-9876543
Andy   | Jones    |
(4 rows)
```

拡張 4: Map 要素へのアクセス

Redshift Spectrum は、map 列および array 列を持つ struct 型を含む key として value データ型を扱うことができます。key は、scalar である必要があります。値は任意のデータ型にすることができます。

たとえば、次のコードでは、電話番号を保存するために、map を使用して外部テーブルを作成します。

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones map<varchar(20), varchar(20)>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

map 型の動作は、array 列および key 列を持つ value 型と似ているため、前述のスキーマは次のように考えることができます。

```
CREATE EXTERNAL TABLE spectrum.customers3 (
  id     int,
  name   struct<given:varchar(20), family:varchar(20)>,
  phones array<struct<key:varchar(20), value:varchar(20)>>,
  orders array<struct<shipdate:timestamp, price:double precision>>
)
STORED AS PARQUET
LOCATION 's3://redshift-downloads/tickit/spectrum/customers/';
```

次のクエリでは、顧客名と携帯電話番号が返ります。この番号は顧客名ごとに返ります。map クエリは、array 型のネスト化された struct のクエリと同じように処理されます。前述のように、以下のクエリでは、外部テーブルを作成した場合にのみデータが返ります。

```
SELECT c.name.given, c.name.family, p.value
FROM   spectrum.customers c, c.phones p
WHERE  p.key = 'mobile';
```

> **Note**
>
> key の map は、ファイル形式が Ion や JSON の場合の string です。

ネストデータのユースケース

このトピックでは、ネストデータのユースケースについて説明します。ネストデータは、ネストされたフィールドを含むデータです。ネストされたフィールドは、配列、構造体、オブジェクトなど、単一のエンティティとして結合されるフィールドです。

以前に説明した拡張は、通常の SQL 機能と組み合わせることができます。以下のユースケースでは、一般的な組み合わせを取り上げます。これらの例は、ネストデータの使用方法を示すのに役立ちます。これらはチュートリアルの範囲外です。

トピック

- ネストデータの取り込み
- サブクエリを使用したネストデータの集約

- Amazon Redshift とネストデータの結合

ネストデータの取り込み

複合データ型を含む外部テーブルからデータを取り込むには、CREATE TABLE AS ステートメントを使用します。以下のクエリでは、LEFT JOIN を使用して、外部テーブルから顧客とその電話番号をすべて抽出し、Amazon Redshift テーブルの CustomerPhones にそのデータを保存します。

```
CREATE TABLE CustomerPhones AS
SELECT  c.name.given, c.name.family, p AS phone
FROM    spectrum.customers c LEFT JOIN c.phones p ON true;
```

サブクエリを使用したネストデータの集約

ネストデータを集約するにはサブクエリを使用します。この方法の概要を次の例に示します。

```
SELECT c.name.given, c.name.family, (SELECT COUNT(*) FROM c.orders o) AS ordercount
FROM   spectrum.customers c;
```

以下のデータが返ります。

```
given    |  family  |  ordercount
--------|----------|--------------
 Jenny  |  Doe     |       0
 John   |  Smith   |       2
 Andy   |  Jones   |       1
 (3 rows)
```

> Note
>
> 親の行でグループ化してネストデータを集約するには、前の例で示した方法が最も簡単です。この例では、ネスト化された行 c.orders は、親の行 c によってグループ化されます。また、id は customer ごとに一意であり、o.shipdate が null になることはないという前提で、次の例に示すように集約することができます。ただし、この方法は通常、前の例ほど効率的ではありません。

```
SELECT    c.name.given, c.name.family, COUNT(o.shipdate) AS ordercount
FROM      spectrum.customers c LEFT JOIN c.orders o ON true
```

```
GROUP BY  c.id, c.name.given, c.name.family;
```

また、FROM 句で祖先クエリのエイリアス (c) を参照し、配列データを抽出するサブクエリを使用して、クエリを記述することもできます。次の例で、この方法を示します。

```
SELECT c.name.given, c.name.family, s.count AS ordercount
FROM   spectrum.customers c, (SELECT count(*) AS count FROM c.orders o) s;
```

Amazon Redshift とネストデータの結合

また、Amazon Redshift データを外部テーブルのネストデータと結合することもできます。たとえば、Amazon S3 に次のネストデータがあるとします。

```
CREATE EXTERNAL TABLE spectrum.customers2 (
  id      int,
  name    struct<given:varchar(20), family:varchar(20)>,
  phones  array<varchar(20)>,
  orders  array<struct<shipdate:timestamp, item:int>>
);
```

さらに、Amazon Redshift に次のテーブルがあるとします。

```
CREATE TABLE prices (
  id int,
  price double precision
);
```

次のクエリでは、前の値に基づいて各顧客の購入の合計数と金額を検索します。以下はあくまでも例です。データは、前に説明したテーブルを作成した場合にのみ返ります。

```
SELECT   c.name.given, c.name.family, COUNT(o.date) AS ordercount, SUM(p.price) AS
 ordersum
FROM     spectrum.customers2 c, c.orders o, prices p ON o.item = p.id
GROUP BY c.id, c.name.given, c.name.family;
```

ネストデータの制限 (プレビュー)

このトピックでは、Redshift Spectrum でネストされたデータを読み込むための制限事項について説明します。ネストデータは、ネストされたフィールドを含むデータです。ネストされたフィールドは、配列、構造体、オブジェクトなど、単一のエンティティとして結合されるフィールドです。

> **Note**
>
> 以下のリストで「(プレビュー)」と記載されている制限は、以下のリージョンで作成されたプレビュークラスターとプレビューワークグループにのみ適用されます。
>
> - 米国東部 (オハイオ) (us-east-2)
> - 米国東部 (バージニア北部) (us-east-1)
> - 米国西部 (北カリフォルニア) (us-west-1)
> - アジアパシフィック (東京) (ap-northeast-1)
> - 欧州 (アイルランド) (eu-west-1)
> - 欧州 (ストックホルム) (eu-north-1)
>
> プレビュークラスターの設定の詳細については、「Amazon Redshift 管理ガイド」の「プレビュークラスターの作成」を参照してください。プレビューワークグループの設定の詳細については、「Amazon Redshift 管理ガイド」の「プレビューワークグループの作成」を参照してください。

ネストデータには以下の制限が適用されます。

- `array` 型または `map` 型には他の `array` 型または `map` 型を含めることができますが、ネスト化された `arrays` または `maps` のクエリで `scalar` 値が返されないことが前提です (プレビュー)。
- Amazon Redshift Spectrum では、外部テーブルとしてのみ複合データ型がサポートされています。
- サブクエリの結果列は、最上位である必要があります (プレビュー)。
- `OUTER JOIN` 式によって、ネスト化されたテーブルが参照される場合は、テーブルとそのネスト化された配列 (およびマップ) でのみ参照されます。`OUTER JOIN` 式でネスト化されたテーブルを参照しない場合は、任意の数のネスト化されていないテーブルを参照することができます。
- サブクエリの `FROM` 句でネスト化されたテーブルが参照されている場合は、その他のテーブルを参照することはできません。
- サブクエリが、親テーブルを参照するネスト化されたテーブルに依存する場合、親テーブルは `FROM` 句でのみ使用できます。`SELECT` 句や `WHERE` 句など、他の句で親を使用することはできません。例えば、次のクエリは、サブクエリの `SELECT` 句で親テーブル `c` を参照しているため、実行されません。

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(c.id) FROM c.phones p WHERE p LIKE '858%') > 1;
```

次のクエリは、親である c は、サブクエリの FROM 句でのみ使用されるため、問題なく動作します。

```
SELECT c.name.given
FROM   spectrum.customers c
WHERE (SELECT COUNT(*) FROM c.phones p WHERE p LIKE '858%') > 1;
```

- FROM 句以外の場所にあるネストされたデータにアクセスするサブクエリは、単一の値を返す必要があります。唯一の例外は、(NOT) EXISTS 句の WHERE 演算子です。
- (NOT) IN はサポートされていません。
- ネストされたすべてのタイプの最大ネスト深度は 100 です。この制限は、すべてのファイル形式 (Parquet、ORC、Ion、JSON) に適用されます。
- ネストデータにアクセスする集約サブクエリは、arrays 句の maps および FROM のみを参照できます。外部テーブルは参照できません。
- Redshift Spectrum テーブル内のネストされたデータの疑似列のクエリはサポートされていません。詳細については、「疑似列」を参照してください。
- 配列またはマップの列を FROM 句で指定してデータを抽出する場合、それらの列から、scalar である値のみを選択できます。例えば、以下のクエリは両方とも、配列内の要素を SELECT しようとしています。arr.a を選択するクエリは、arr.a が scalar 値であるため機能します。2 番目のクエリは、array が FROM 句で s3.nested table から抽出された配列であるため、機能しません (プレビュー)。

```
SELECT array_column FROM s3.nested_table;

array_column
-----------------
[{"a":1},{"b":2}]

SELECT arr.a FROM s3.nested_table t, t.array_column arr;

arr.a
-----
1
```

```
--This query fails to run.
SELECT array FROM s3.nested_table tab, tab.array_column array;
```

別の配列やマップから抽出された配列またはマップは、FROM では使用できません。他の配列の中にネストされている配列やその他の複雑な構造体を選択するには、SELECT ステートメントでインデックスを使用することを検討してください。

複雑なネストされた JSON のシリアル化

このトピックでは、ネストデータを JSON 形式でシリアル化する方法について説明します。ネストデータは、ネストされたフィールドを含むデータです。ネストされたフィールドは、配列、構造体、オブジェクトなど、単一のエンティティとして結合されるフィールドです。

このチュートリアルで説明するメソッドの代わりに、最上位のネストされたコレクション列をシリアル化された JSON としてクエリする方法があります。シリアル化を使用して、Redshift Spectrum を使用して、ネストされたデータを JSON として検査、変換、取り込むことができます。このメソッドは、ORC、JSON、Ion、および Parquet 形式でサポートされています。セッション構成パラメータ json_serialization_enable を使用して、シリアル化動作を構成します。設定すると、複雑な JSON データ型が VARCHAR (65535) にシリアル化されます。ネストされた JSON には JSON 関数 でアクセスできます。詳細については、「json_serialization_enable」を参照してください。

たとえば、json_serialization_enable を設定しないと、ネストされた列に直接アクセスする次のクエリが失敗します。

```
SELECT * FROM spectrum.customers LIMIT 1;

=> ERROR:  Nested tables do not support '*' in the SELECT clause.

SELECT name FROM spectrum.customers LIMIT 1;

=> ERROR:  column "name" does not exist in customers
```

json_serialization_enable を設定すると、最上位のコレクションに直接クエリを実行できます。

```
SET json_serialization_enable TO true;

SELECT * FROM spectrum.customers order by id LIMIT 1;
```

```
id | name                                    | phones          | orders
---+----------------------------------------+----------------
+-----------------------------------------------------------------------------------------------------------
1  | {"given": "John", "family": "Smith"} | ["123-457789"] | [{"shipdate":
 "2018-03-01T11:59:59.000Z", "price": 100.50}, {"shipdate": "2018-03-01T09:10:00.000Z",
 "price": 99.12}]

SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "John", "family": "Smith"}
```

ネストされた JSON をシリアル化するときは、以下の項目について考慮してください。

- コレクション列が VARCHAR (65535) としてシリアル化されている場合、それらのネストされたサブフィールドは、クエリ構文の一部として直接アクセスできません (filter 句内など)。ただし、JSON 関数を使用して、ネストされた JSON にアクセスできます。
- 次の特殊な表現はサポートされていません。
  - ORC 組合
  - 複合型キーを持つ ORC マップ
  - Ion データグラム
  - Ion SEXP
- タイムスタンプは ISO シリアル化された文字列として返されます。
- プリミティブマップキーは文字列に昇格されます (たとえば、1 ~ "1")。
- 最上位の null 値は NULL としてシリアル化されます。
- シリアル化によって VARCHAR の最大サイズである 65535 がオーバーフローすると、セルは NULL に設定されます。

JSON 文字列を含む複合型のシリアル化

デフォルトでは、ネストされたコレクションに含まれる文字列値は、エスケープされた JSON 文字列としてシリアル化されます。文字列が有効な JSON である場合、エスケープは望ましくない場合があります。代わりに、VARCHAR であるネストされたサブ要素またはフィールドを JSON として直接記述することができます。`json_serialization_parse_nested_strings` セッションレベルの設定でこの動作を有効にします。`json_serialization_enable` と `json_serialization_parse_nested_strings` の両方が設定されている場合、有効な

JSON 値はエスケープ文字なしでインラインでシリアル化されます。値が有効な JSON でない場合、`json_serialization_parse_nested_strings` 設定値が設定されていないかのようにエスケープされます。詳細については、「json_serialization_parse_nested_strings」を参照してください。

たとえば、前の例のデータの name VARCHAR(20) フィールドに `structs` 複合型として JSON が含まれているとします。

```
name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

`json_serialization_parse_nested_strings` が設定されている場合、name 列は次のようにシリアル化されます。

```
SET json_serialization_enable TO true;
SET json_serialization_parse_nested_strings TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": {"first":"John","middle":"James"}, "family": "Smith"}
```

次のようにエスケープされません。

```
SET json_serialization_enable TO true;
SELECT name FROM spectrum.customers order by id LIMIT 1;

name
---------
{"given": "{\"first\":\"John\",\"middle\":\"James\"}", "family": "Smith"}
```

HyperLogLog スケッチ

このトピックでは、Amazon Redshift で HyperLogLog スケッチを使用する方法について説明します。HyperLogLog は、データセット内の個別要素の数の近似を指す、COUNT DISTINCT 問題のアルゴリズムです。HyperLogLog スケッチは、データセットに関する一意性のデータの配列です。

HyperLogLog は，多重セットの基数を推定するために使用されるアルゴリズムです。Cardinality とは、多重セット内の個別値の数を指します。たとえば、{4,3,6,2,2,6,4,3,6,2,2,3} のセットでは、基数は 4 で、4、3、6、2 という個別の値を持ちます。

HyperLogLog アルゴリズムの精度 (m 値とも呼ばれる) は、推定基数の精度に影響を与える可能性があります。基数の推定では、Amazon Redshift はデフォルトの精度値 15 を使用します。小さいデータセットでは、この値は最大 26 になる可能性があります。したがって、平均相対誤差の範囲は 0.01～0.6% です。

多重セットの基数を計算するとき、HyperLogLog アルゴリズムは、HLL スケッチと呼ばれる構成を生成します。HLL スケッチは、多重セット内の個別値に関する情報をカプセル化します。Amazon Redshift データ型 HLLSKETCH は、このようなスケッチ値を表します。このデータ型は、Amazon Redshift テーブルにスケッチを保存するために使用できます。さらに、Amazon Redshift は、集計関数およびスカラー関数として HLLSKETCH 値に適用できるオペレーションをサポートしています。これらの関数を使用して、HLLSKETCH の基数を抽出し、複数の HLLSKETCH 値を組み合わせることができます。

HLLSKETCH データ型は、大規模なデータセットから基数を抽出するときに、クエリのパフォーマンスに大きな利点をもたらします。これらのデータセットは、HLLSKETCH 値を使用して事前に集計し、テーブルに保存できます。Amazon Redshift は、基になるデータセットにアクセスすることなく、保存されている HLLSKETCH 値から直接基数を抽出できます。

HLL スケッチを処理する場合、Amazon Redshift はスケッチのメモリフットプリントを最小限に抑え、抽出された基数の精度を最大化する最適化を実行します。Amazon Redshift は、HLL スケッチに疎と蜜といった 2 つの表現を使用します。HLLSKETCH は疎フォーマットで始まります。新しい値が挿入されると、そのサイズが大きくなります。そのサイズが密表現のサイズに達すると、Amazon Redshift はスケッチを疎から密に自動的に変換します。

Amazon Redshift は、スケッチが疎形式の場合、HLLSKETCH を JSON としてインポート、エクスポート、およびプリントします。Amazon Redshift は、スケッチが密集形式の場合、HLLSKETCH を Base64 文字列としてインポート、エクスポート、およびプリントします。UNLOAD の詳細については、HLLSKETCH データ型のアンロード を参照してください。テキストまたはコンマ区切り値

(CSV) データを Amazon Redshift にインポートするには、COPY コマンドを使用します。詳細については、「HLLSKETCH データ型のロード」を参照してください。

HyperLogLog で使用される関数の詳細については、HyperLogLog 関数 を参照してください。

トピック

- 考慮事項
- 制約事項
- 例

考慮事項

このトピックでは、Amazon Redshift での HyperLogLog の使用の詳細について説明します。

Amazon Redshift で HyperLogLog を使用する際の考慮事項を次に示します。

- 次の非 HyperLogLog 関数は、タイプ HLLSKETCH の入力またはタイプ HLLSKETCH の列を受け入れることができます。
  - 集計関数 COUNT
  - 条件式 COALESCE および NVL
  - CASE 式
- サポートされているエンコードは RAW です。
- HLLSKETCH 列を含むテーブルに対して UNLOAD オペレーションをテキストまたは CSV に実行できます。UNLOAD HLLSKETCH 列を使用して、HLLSKETCH データを書き込むことができます。Amazon Redshift は、疎表現の場合は JSON 形式で、密表現の場合は Base64 形式でデータを表示します。UNLOAD の詳細については、HLLSKETCH データ型のアンロード を参照してください。

  以下に、JSON 形式で表される疎の HyperLogLog スケッチに使用される形式を示します。

  ```
  {"version":1,"logm":15,"sparse":{"indices":
  [15099259,33107846,37891580,50065963],"values":[2,3,2,1]}}
  ```
- COPY コマンドを使用して、テキストまたは CSV データを Amazon Redshift にインポートできます。詳細については、「HLLSKETCH データ型のロード」を参照してください。
- HLLSKETCH のデフォルトのエンコーディングは RAW です。詳細については、「圧縮エンコード」を参照してください。

制約事項

このトピックでは、Amazon Redshift の HyperLogLog の制限事項について説明します。

Amazon Redshift で HyperLogLog を使用する際の制限事項を次に示します。

- Amazon Redshift テーブルでは、Amazon Redshift テーブルのソートキーまたはディストリビューションキーとして HLLSKETCH 列をサポートしていません。
- Amazon Redshift は、ORDER BY、GROUP BY、または DISTINCT 句の HLLSKETCH 列をサポートしていません。
- HLLSKETCH 列はテキストまたは CSV 形式でのみアンロードできます。次に、Amazon Redshift は HLLSKETCH データを JSON 形式または Base64 形式で書き込みます。UNLOAD の詳細については、UNLOAD を参照してください。
- Amazon Redshift は、精度 (logm 値) が 15 の HyperLogLog スケッチのみをサポートします。
- JDBC ドライバーおよび ODBC ドライバーは、HLLSKETCH データ型をサポートしていません。したがって、結果セットでは VARCHAR を使用して HLLSKETCH 値を表します。
- Amazon Redshift Spectrum は、HLLSKETCH データをネイティブにサポートしていません。よって、HLLSKETCH 列がある外部テーブルを作成または修正することはできません。
- Python ユーザー定義関数 (UDF) のデータ型は、HLLSKETCH データ型をサポートしていません。Python UDF の詳細については、「スカラー Python UDF」を参照してください。

例

このセクションでは、Amazon Redshift で HyperLogLog を使用する例を示します。

トピック

- 例: サブクエリで基数を返す
- 例: サブクエリで結合されたスケッチから HLLSKETCH タイプを返す
- 例: 複数のスケッチを組み合わせて HyperLogLog スケッチを返す
- 例: 外部テーブルを使用して S3 データに対する HyperLogLog スケッチを生成する

例: サブクエリで基数を返す

次の例では、Sales という名前のテーブルでサブクエリ内の各スケッチの基数を返します。

```
CREATE TABLE Sales (customer VARCHAR, country VARCHAR, amount BIGINT);
INSERT INTO Sales VALUES ('David Joe', 'Greece', 14.5),  ('David Joe', 'Greece',
 19.95), ('John Doe', 'USA', 29.95), ('John Doe', 'USA', 19.95), ('George Spanos',
 'Greece', 9.95), ('George Spanos', 'Greece', 2.95);
```

次のクエリは、各国の顧客の HLL スケッチを生成し、基数を抽出します。これは、各国のユニークな顧客を示しています。

```
SELECT hll_cardinality(sketch), country
FROM (SELECT hll_create_sketch(customer) AS sketch, country
        FROM Sales
        GROUP BY country) AS hll_subquery;

hll_cardinality | country
----------------+---------
            1   | USA
            2   | Greece
 ...
```

例: サブクエリで結合されたスケッチから HLLSKETCH タイプを返す

次の例では、サブクエリから個々のスケッチの組み合わせを表す単一の HLLSKETCH 型を返します。スケッチは、HLL_COMBINE 集計関数を使用して結合されます。

```
SELECT hll_combine(sketch)
FROM (SELECT hll_create_sketch(customer) AS sketch
        FROM Sales
        GROUP BY country) AS hll_subquery

                                        hll_combine
--------------------------------------------------------------------------------------------
 {"version":1,"logm":15,"sparse":{"indices":[29808639,35021072,47612452],"values":
[1,1,1]}}
(1 row)
```

例: 複数のスケッチを組み合わせて HyperLogLog スケッチを返す

次の例では、テーブル `page-users` に、ユーザーが特定のウェブサイトでアクセスした各ページで、事前に集計されたスケッチが保存されているとします。このテーブルの各行には、訪問したページを表示するすべてのユーザー ID を表す HyperLogLog スケッチが含まれています。

```
page_users
-- +----------------+-------------+--------------+
-- | _PARTITIONTIME | page        | sketch |
-- +----------------+-------------+--------------+
-- | 2019-07-28     | homepage    | CHAQkAQYA... |
-- | 2019-07-28     | Product A   | CHAQxPnYB... |
-- +----------------+-------------+--------------+
```

次の例では、事前に集計された複数のスケッチを結合し、1 つのスケッチを生成します。このスケッチは、各スケッチがカプセル化する集合基数をカプセル化します。

```
SELECT hll_combine(sketch) as sketch
FROM page_users
```

出力は次の例のようになります。

```
-- +-----------------------------------------+
-- | sketch |
-- +-----------------------------------------+
-- | CHAQ3sGoCxgCIAuCB4iAIBgTIBgqgIAgAwY.... |
-- +-----------------------------------------+
```

新しいスケッチを作成するときに、次に示すように、HLL_CARDINALITY 関数を使用して、集合的な個別値を取得できます。

```
SELECT hll_cardinality(sketch)
FROM (
  SELECT
  hll_combine(sketch) as sketch
  FROM page_users
) AS hll_subquery
```

出力は次の例のようになります。

```
-- +-------+
-- | count |
-- +-------+
-- | 54356 |
-- +-------+
```

例: 外部テーブルを使用して S3 データに対する HyperLogLog スケッチを生成する

以下の例では、基数推定のために Amazon S3 に直接アクセスしないように、HyperLogLog スケッチをキャッシュします。

HyperLogLog スケッチを事前に集計して、Amazon S3 データを保持するために定義された外部テーブルにキャッシュできます。これにより、基になる基本データにアクセスせずに、基数の推定値を抽出できます。

たとえば、タブ区切りテキストファイルのセットを Amazon S3 にアンロードしたとします。次のクエリを実行して、`sales` という名前の Amazon Redshift 外部スキーマに `spectrum` という名前の外部テーブルを定義します。この例における Amazon S3 バケットは 米国東部 (バージニア北部) AWS リージョンにあります。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid smallint,
buyerid smallint,
eventid integer,
dateid integer,
qtysold integer,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t' stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/';
```

任意の日付に商品を購入した個別の購入者を計算するとします。そのために、以下の例では、その年の各日の購入者 ID のスケッチを生成し、その結果を Amazon Redshift テーブル `hll_sales` に保存します。

```
CREATE TABLE hll_sales AS
SELECT saletime, hll_create_sketch(buyerid) AS sketch
FROM spectrum.sales
GROUP BY saletime;

SELECT TOP 5 * FROM hll_sales;
```

出力は次の例のようになります。

```
-- hll_sales

-- | saletime         | sketch
    |
-- +-----------------
+------------------------------------------------------------------+
-- | 7/22/2008 8:30  | {"version":1,"logm":15,"sparse":{"indices":[9281416],"values":
[1]}}
-- | 2/19/2008 0:38  | {"version":1,"logm":15,"sparse":{"indices":[48735497],"values":
[3]}}
-- | 11/5/2008 4:49  | {"version":1,"logm":15,"sparse":{"indices":[27858661],"values":
[1]}}
-- | 10/27/2008 4:08 | {"version":1,"logm":15,"sparse":{"indices":[65295430],"values":
[2]}}
-- | 2/16/2008 9:37  | {"version":1,"logm":15,"sparse":{"indices":[56869618],"values":
[2]}}
-- +----------------
 +------------------------------------------------------------------+
```

次のクエリは、2008 年の感謝祭後の金曜日に商品を購入した個別購入者の推定数を示します。

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE trunc(saletime) = '2008-11-28';
```

出力は次の例のようになります。

```
distinct_buyers
---------------
386
```

特定の日付範囲で商品を購入した個別ユーザーの数が必要であるとします。例としては、感謝祭の後の金曜日から次の月曜日までにすることができます。これを取得するために、次のクエリは `hll_combine` 集計関数を使用します。この機能を使用すると、選択した範囲より 1 日を超えて商品を購入した購入者が二重カウントされないようにすることができます。

```
SELECT hll_cardinality(hll_combine(sketch)) as distinct_buyers
FROM hll_sales
WHERE saletime BETWEEN '2008-11-28' AND '2008-12-01';
```

出力は次の例のようになります。

```
distinct_buyers
---------------
1166
```

hll_sales テーブルを最新の状態に保つには、1 日の終わりに次のクエリを実行します。これを行うと、本日商品を購入した購入者の ID に基づいて HyperLogLog スケッチが生成され、hll_sales テーブルに追加されます。

```
INSERT INTO hll_sales
SELECT saletime, hll_create_sketch(buyerid)
FROM spectrum.sales
WHERE TRUNC(saletime) = to_char(GETDATE(), 'YYYY-MM-DD')
GROUP BY saletime;
```

データベース間でのデータのクエリ

このトピックでは、単一の Amazon Redshift クラスターにある複数の Amazon Redshift データベースで動作するクエリである、クロスデータベースクエリについて説明します。

Amazon Redshift で [cross-database queries (クロスデータベースのクエリ)] を使用すると、Amazon Redshift クラスター内のデータベース間でクエリを実行できます。クロスデータベースクエリを使用すると、接続しているデータベースに関係なく、Amazon Redshift クラスター内の任意のデータベースからデータをクエリできます。クロスデータベースクエリでは、データコピーを排除し、データ組織を簡素化して、同じデータウェアハウスからの複数のビジネスグループをサポートします。

クロスデータベースクエリでは、次のことができます。

- Amazon Redshift クラスター内のデータベース間でデータをクエリします。

  接続しているデータベースからクエリを実行できるだけでなく、許可のある他のデータベースから読み込むこともできます。

  接続されていない他のデータベースでデータベースオブジェクトにクエリを実行すると、それらのデータベースオブジェクトへの読み込みアクセスのみが付与されます。クロスデータベースクエリを使用すると、特定のデータベースに接続しなくても、Amazon Redshift クラスター上の任意のデータベースのデータにアクセスできます。これにより、Amazon Redshift クラスター内の複数のデータベースに分散しているデータをすばやく簡単にクエリして結合することができます。

  また、複数のデータベースのデータセットを 1 つのクエリで結合し、ビジネスインテリジェンス (BI) または分析ツールを使用してデータを分析することもできます。標準の Amazon Redshift SQL コマンドを使用して、ユーザーに対して詳細なテーブルレベルのアクセス制御を引き続き設定できます。これにより、ユーザーが、許可を持つデータの関連するサブセットのみを表示できるようにすることができます。

- オブジェクトをクエリします。

  3 つの部分で表記される完全修飾オブジェクト名を使用して、他のデータベースオブジェクトをクエリできます。任意のデータベースオブジェクトへのフルパスは、データベース名、スキーマ、オブジェクト名の 3 つのコンポーネントで構成されます。フルパス表記 (`database_name.schema_name.object_name`) を使用して、他のデータベースから任意のオブジェクトにアクセスできます。特定の列にアクセスするには、`database_name.schema_name.object_name.column_name` を使用します。

外部スキーマ表記を使用して、別のデータベースのスキーマのエイリアスを作成することもできます。この外部スキーマは、別のデータベースとスキーマのペアを参照します。クエリは、外部スキーマ表記 (*external_schema_name*.*object_name*) を使用して他のデータベースオブジェクトにアクセスできます。

同じ読み込み専用クエリでは、他のデータベースのユーザーテーブル、通常のビュー、マテリアライズドビュー、遅延バインディングビューなど、さまざまなデータベースオブジェクトをクエリできます。

- 許可を管理します。

  Amazon Redshift クラスター内の任意のデータベースにあるオブジェクトに対するアクセス権限を持つユーザーは、これらのオブジェクトをクエリできます。GRANT コマンドを使用して、ユーザーおよびユーザーグループに権限を付与します。また、ユーザーが特定のデータベースオブジェクトへのアクセスを必要としなくなったときに REVOKE コマンドを使用して権限を取り消すこともできます。

- メタデータと BI ツールを操作します。

  外部スキーマを作成して、同じ Amazon Redshift クラスター内の別の Amazon Redshift データベースのスキーマを参照できます。詳細については、CREATE EXTERNAL SCHEMA を参照してください。

  外部スキーマ参照が作成されると、Amazon Redshift はメタデータを探索するためのツールについて、SVV_EXTERNAL_TABLES および SVV_EXTERNAL_COLUMNS の他のデータベースのスキーマの下にテーブルを表示します。

  クロスデータベースクエリを BI ツールと統合するには、次のシステムビューを使用できます。これらは、Amazon Redshift クラスター上の接続データベースや他のデータベースにあるオブジェクトのメタデータに関する情報を表示することができます。

  以下は、Amazon Redshift クラスターにあるすべてのデータベースのすべての Amazon Redshift オブジェクトと外部オブジェクトを表示するシステムビューです。

  - SVV_ALL_COLUMNS
  - SVV_ALL_SCHEMAS
  - SVV_ALL_TABLES

  以下は、Amazon Redshift クラスターにあるすべてのデータベースのすべての Amazon Redshift オブジェクトを表示するシステムビューです。

- SVV_REDSHIFT_COLUMNS
- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES

トピック

考慮事項

このトピックでは、Amazon Redshift でのクロスデータベースクエリの使用の詳細について説明します。

Amazon Redshift でクロスデータベースクエリ機能を使用する場合は、次の点を考慮してください。

- Amazon Redshift は、ra3.4xlarge、ra3.16xlarge、ra3.xlplus、および ra3.large ノードタイプでのクロスデータベースクエリをサポートしています。
- Amazon Redshift は、同じ Amazon Redshift クラスター内の 1 つ以上のデータベースにわたるテーブルまたはビューからのデータの結合をサポートしています。
- Amazon Redshift Serverless は Amazon Redshift クラスターと同じクロスデータベース機能をサポートしています。そのため、サーバーレス名前空間内の 1 つ以上のデータベースにわたるテーブルまたはビューからのデータの結合が可能です。
- 接続データベース上のトランザクション内のすべてのクエリは、トランザクションの開始時点のデータと同じ状態で他のデータベースのデータを読み込みます。このアプローチは、データベース間でクエリトランザクションの一貫性を提供します。Amazon Redshift は、クロスデータベースクエリのトランザクションの一貫性をサポートします。
- データベース全体でメタデータを取得するには、SVV_ALL* および SVV_REDSHIFT* メタデータビューを使用します。information_schema および pg_catalog にあるクロスデータベースメタデータのテーブルまたはビューのクエリに 3 部分からなる記法、または外部スキーマを使用することはできません。

制約事項

このトピックでは、Amazon Redshift でのクロスデータベースクエリの制限事項について説明します。

Amazon Redshift でクロスデータベースクエリ機能を使用する場合は、次の制限事項に注意してください。

- 接続されていない他のデータベースでデータベースオブジェクトにクエリを実行すると、それらのデータベースオブジェクトへの読み込みアクセスのみが付与されます。
- 他のデータベースで作成され、そこからさらに異なるデータベースのオブジェクトを参照するビューを、クエリすることはできません。
- クラスター内の他のデータベースのオブジェクトに対しては、遅延バインディングおよびマテリアライズドビューのみ作成が可能です。クラスター内の他のデータベースのオブジェクトに対して、通常のビューを作成することはできません。
- Amazon Redshift では、データベース間のクエリのための列レベル権限を持つテーブルをサポートしていません。
- Amazon Redshift は、AWS Glue またはフェデレートされたデータベースでのカタログオブジェクトのクエリをサポートしていません。これらのオブジェクトをクエリするには、まず各データベース内の外部データソースを参照する外部スキーマを作成します。
- インターリーブソートキーを含むテーブルに対するクロスデータベースクエリはサポートされていません。

クロスデータベースクエリの例

このトピックには、クロスデータベースクエリを使用する方法の例が含まれています。クロスデータベースクエリは、単一の Amazon Redshift クラスター内の複数のデータベースで動作するクエリです。

次の例を使用して、Amazon Redshift データベースを参照するクロスデータベースクエリの設定方法をご覧ください。

まず、Amazon Redshift クラスターにデータベース db1 と db2 およびユーザー `user1` と `user2` を作成します。詳細については、CREATE DATABASEおよびCREATE USERを参照してください。

```
--As user1 on db1
CREATE DATABASE db1;
```

```
CREATE DATABASE db2;

CREATE USER user1 PASSWORD 'Redshift01';

CREATE USER user2 PASSWORD 'Redshift01';
```

db1 の user1 として、テーブルを作成し、user2 にアクセス権限を付与し、table1 に値を挿入します。詳細については、GRANTおよびINSERTを参照してください。

```
--As user1 on db1
CREATE TABLE table1 (c1 int, c2 int, c3 int);

GRANT SELECT ON table1 TO user2;

INSERT INTO table1 VALUES (1,2,3),(4,5,6),(7,8,9);
```

db2 の user2 として、3 つの部分からなる表記を使用して db2 でデータベース間でクエリを実行します。

```
--As user2 on db2
SELECT * from db1.public.table1 ORDER BY c1;
c1 | c2  | c3
---+-----+----
1  |  2  | 3
4  |  5  | 6
7  |  8  | 9
(3 rows)
```

user2 の db2 として、外部スキーマを作成し、外部スキーマ表記を使用して db2 でデータベース間でクエリを実行します。

```
--As user2 on db2
CREATE EXTERNAL SCHEMA db1_public_sch
FROM REDSHIFT DATABASE 'db1' SCHEMA 'public';

SELECT * FROM db1_public_sch.table1 ORDER BY c1;

c1  | c2 | c3
----+----+----
1   | 2  | 3
```

```
4   | 5  | 6
7   | 8  | 9
(3 rows)
```

さまざまなビューを作成し、それらのビューに許可を付与するには、db1 の user1 のように、次の手順を実行します。

```
--As user1 on db1
CREATE VIEW regular_view AS SELECT c1 FROM table1;

GRANT SELECT ON regular_view TO user2;


CREATE MATERIALIZED VIEW mat_view AS SELECT c2 FROM table1;

GRANT SELECT ON mat_view TO user2;


CREATE VIEW late_bind_view AS SELECT c3 FROM public.table1 WITH NO SCHEMA BINDING;

GRANT SELECT ON late_bind_view TO user2;
```

db2 の user2 として、3 つの部分からなる表記を使用して、次のデータベース間でクエリを実行し、特定のビューを表示します。

```
--As user2 on db2
SELECT * FROM db1.public.regular_view;
c1
----
1
4
7
(3 rows)

SELECT * FROM db1.public.mat_view;
c2
----
8
5
2
(3 rows)
```

```
SELECT * FROM db1.public.late_bind_view;
c3
----
3
6
9
(3 rows)
```

db2 の user2 として、外部スキーマ表記を使用して次のデータベース間でクエリを実行し、遅延バインディングビューをクエリします。

```
--As user2 on db2
SELECT * FROM db1_public_sch.late_bind_view;
c3
----
3
6
9
(3 rows)
```

db2 の user2 として、単一のクエリで接続されたテーブルを使用して次のコマンドを実行します。

```
--As user2 on db2
CREATE TABLE table1 (a int, b int, c int);

INSERT INTO table1 VALUES (1,2,3), (4,5,6), (7,8,9);

SELECT a AS col_1, (db1.public.table1.c2 + b) AS sum_col2, (db1.public.table1.c3 + c)
 AS sum_col3 FROM db1.public.table1, table1 WHERE db1.public.table1.c1 = a;
col_1 | sum_col2 | sum_col3
------+----------+----------
1     | 4        | 6
4     | 10       | 12
7     | 16       | 18
(3 rows)
```

次の例では、クラスター上のすべてのデータベースを一覧表示します。

```
select database_name, database_owner, database_type
from svv_redshift_databases
where database_name in ('db1', 'db2');
```

```
 database_name | database_owner | database_type
---------------+----------------+---------------
 db1           |            100 | local
 db2           |            100 | local
(2 rows)
```

次の例では、クラスター上のすべてのデータベースのすべての Amazon Redshift スキーマを一覧表示します。

```
select database_name, schema_name, schema_owner, schema_type
from svv_redshift_schemas
where database_name in ('db1', 'db2');

 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
(6 rows)
```

次の例では、クラスター上のすべての Amazon Redshift テーブルまたはすべてのデータベースのビューを一覧表示します。

```
select database_name, schema_name, table_name, table_type
from svv_redshift_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | mat_view            | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | regular_view        | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

次の例では、クラスター上のすべてのデータベースのすべての Amazon Redshift と外部スキーマを一覧表示します。

```
select database_name, schema_name, schema_owner, schema_type
from svv_all_schemas where database_name in ('db1', 'db2') ;

 database_name |    schema_name     | schema_owner | schema_type
---------------+--------------------+--------------+-------------
 db1           | pg_catalog         |            1 | local
 db1           | public             |            1 | local
 db1           | information_schema |            1 | local
 db2           | pg_catalog         |            1 | local
 db2           | public             |            1 | local
 db2           | information_schema |            1 | local
 db2           | db1_public_sch     |            1 | external
(7 rows)
```

次の例では、クラスター上のすべてのデータベースのすべての Amazon Redshift と外部テーブルを一覧表示します。

```
select database_name, schema_name, table_name, table_type
from svv_all_tables
where database_name in ('db1', 'db2') and schema_name in ('public');

 database_name | schema_name |     table_name      | table_type
---------------+-------------+---------------------+------------
 db1           | public      | regular_view        | VIEW
 db1           | public      | mv_tbl__mat_view__0 | TABLE
 db1           | public      | mat_view            | VIEW
 db1           | public      | late_bind_view      | VIEW
 db1           | public      | table1              | TABLE
 db2           | public      | table2              | TABLE
(6 rows)
```

クエリエディタでのクロスデータベースクエリの使用

このトピックでは、クエリエディタでクロスデータベースクエリを使用する方法について説明します。クロスデータベースクエリは、単一の Amazon Redshift クラスター内の複数のデータベースで動作するクエリです。

クロスデータベースクエリを使用すると、特定のデータベースに接続しなくても、Amazon Redshift クラスター上の任意のデータベースのデータにアクセスできます。接続されていない他のデータベースに対してクロスデータベースクエリを実行する場合、それらのデータベースオブジェクトへの読み込みアクセスのみが付与されます。

3 つの部分で表記される完全修飾オブジェクト名を使用して、他のデータベースオブジェクトをクエリできます。任意のデータベースオブジェクトへのフルパスは、データベース名、スキーマ、オブジェクト名の 3 つのコンポーネントで構成されます。例は *database_name.schema_name.object_name* です。

クエリエディタ v2 でクロスデータベースクエリを使用するには

1. AWS Management Consoleにサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. Amazon Redshift クエリエディタ v2 でクロスデータベースクエリを使用するクラスターを作成します。詳細については、「Amazon Redshift 管理ガイド」の「クラスターの作成」を参照してください。
3. 適切な許可でクエリエディタへのアクセスを有効にします。詳細については、「Amazon Redshift 管理ガイド」の「クエリエディタ v2 を使用してデータベースのクエリを実行する」を参照してください。
4. ナビゲーションメニューで [クエリエディタ v2] を選択し、次にクラスターのデータベースに接続します。

   クエリエディタ v2 に初めて接続すると、Amazon Redshift はデフォルトで接続されているデータベースのリソースを表示します。
5. これらの他のデータベースのデータベースオブジェクトを表示するためのアクセスを持つ他のデータベースを選択します。オブジェクトを表示するには、適切な許可があることを確認します。データベースを選択すると、Amazon Redshift はデータベースのスキーマリストを表示します。

   スキーマを選択すると、そのスキーマ内のデータベースオブジェクト一覧が表示されます。

   > **Note**
   >
   > Amazon Redshift では、AWS Glue またはフェデレートされたデータベースの一部であるカタログオブジェクトのクエリは、直接サポートされていません。これらをクエリするには、まず各データベース内の外部データソースを参照する外部スキーマを作成します。
   > 3 つの表記からなる Amazon Redshift クロスデータベースクエリは、スキーマ `information_schema` および `pg_catalog` のメタデータテーブルをサポートしていません。これらのメタデータビューはデータベース固有のものであるためです。

6. (オプション) 選択したスキーマのテーブルまたはビューのリストをフィルタリングします。

Amazon Redshift でのデータの共有

Amazon Redshift では、Amazon Redshift クラスター間で、または他の AWS サービスとの間で安全にデータを共有できます。データ共有を使用すると、コピーを作成したり移動したりすることなく、ライブデータを共有できます。データベース管理者とデータエンジニアは、データ共有を使用して、データの制御を維持しながら、分析目的でデータへの安全で読み取り専用のアクセスを提供できます。データアナリスト、ビジネスインテリジェンスの専門家、およびデータサイエンティストは、共有データを活用して、データを複製または移動することなくインサイトを得ることができます。一般的なユースケースには、パートナーとのデータ共有、機能横断的な分析の有効化、組織内のデータの民主化の促進などがあります。以下のセクションでは、Amazon Redshift でのデータ共有の設定と管理の詳細について説明します。

Amazon Redshift のデータ共有により、データを手動で移動またはコピーすることなく、Amazon Redshift クラスター、ワークグループ、AWS アカウント、および AWS リージョン 間で、ライブデータへのアクセスを安全に共有できます。データはライブであるため、すべてのユーザーは更新後すぐに、一貫性のある最新情報をAmazon Redshift で確認できます。

プロビジョニングされたクラスター、サーバーレスワークグループ、アベイラビリティーゾーン、AWS アカウント および AWS リージョン の間でデータを共有できます。クラスタータイプ間およびプロビジョニングされたクラスターとサーバーレス間で共有できます。

読み取りと書き込みの両方のデータベースオブジェクトは、同一の AWS アカウント 内の Amazon Redshift クラスターや Amazon Redshift Serverless ワークグループ間で、または異なる AWS アカウント の間で共有できます。リージョン間でデータを書き込むこともできます。テーブルごとに SELECT、INSERT、UPDATE などのアクセス許可を付与したり、スキーマごとに USAGE と CREATE などのアクセス許可を付与できます。データは、書き込みトランザクションがコミットされるとすぐにライブになり、すべてのウェアハウスで利用できるようになります。

PREVIEW_2023 トラックのデータ共有機能の設定の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

> **Note**
>
> データ共有によるマルチウェアハウス書き込みは、現在 ra3.xlplus クラスターでは使用できません。この機能を使用するには、ra3.4xl クラスター、ra3.16xl クラスター、または Amazon Redshift Serverless ワークグループを作成します。

データ共有の概要

Amazon Redshift では、Amazon Redshift クラスター間でライブデータを安全に共有できるため、データ共有ワークフローが簡素化され、複雑な抽出、変換、ロード (ETL) プロセスの必要性が軽減されます。データ共有を使用すると、Amazon Redshift クラスター間でライブデータを共有できるため、データをコピーまたは複製することなく、最新のデータにリアルタイムでアクセスできます。データベース管理者、データアナリスト、およびデータエンジニアは、チームや組織内または組織間でのデータアクセスとコラボレーションを合理化するために活用することができます。これにより、本番環境のデータを分析チームと共有したり、分散されているデータソース間でリアルタイムレポートを提供したり、アクセス許可を一元的に制御してデータガバナンスを簡素化したりするなどのユースケースが可能になります。以下のセクションでは、Amazon Redshift でのデータ共有の設定と管理の詳細について説明します。

データ共有を使用すると、Amazon Redshift クラスター間でライブデータを安全かつ簡単に共有できます。

データ共有の使用を開始する方法、および AWS Management Consoleを使用してデータ共有を管理する方法については、「データ共有タスクの管理」 を参照してください。

Amazon Redshift のデータ共有ユースケース

Amazon Redshift のデータ共有は、次のユースケースで特に便利です。

- さまざまな種類のビジネスクリティカルなワークロードのサポート – 複数のビジネスインテリジェンス (BI) クラスターまたは分析クラスターとデータを共有する一元的な抽出、変換、およびロード (ETL) クラスターを使用します。このアプローチは、個々のワークロードに対して読み込みワークロードの分離とチャージバックを提供します。料金とパフォーマンスのワークロード固有の要件に応じて、個々のワークロードコンピューティングのサイズとスケーリングを行うことができます。
- クロスグループコラボレーションの有効化 – より広範な分析、データサイエンス、製品間の影響分析のために、チームやビジネスグループ間でシームレスなコラボレーションを実現します。
- サービスとしてのデータの提供 – 組織全体でデータをサービスとして共有できます。
- 環境間でのデータの共有 – 開発、テスト、本番稼働環境間でデータを共有します。さまざまなレベルの詳細なデータを共有することで、チームの俊敏性を向上させることができます。
- Amazon Redshift 内のデータにアクセスするライセンスを供与する – 顧客が数分で検索、サブスクライブ、クエリできる AWS Data Exchange カタログに Amazon Redshift データセットを出品します。

データ共有の書き込みアクセスのデータ共有 (プレビュー)

書き込みのデータ共有には、いくつかの重要なユースケースがあります。

- プロデューサーのビジネスソースデータを更新する — データをサービスとして組織全体で共有できますが、コンシューマーはソースデータに対してアクションを実行することもできます。例えば、最新の値を伝えたり、データの受信を確認したりできます。これらは考えられるビジネスユースケースのほんの一部です。
- プロデューサーに追加レコードを挿入する — コンシューマーは元のソースデータにレコードを追加できます。これらには、必要に応じてコンシューマーからのものとしてマークできます。

データ共有に対して書き込み操作を実行する方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

Amazon Redshift の異なるレベルでのデータ共有

Amazon Redshift を使用すると、さまざまなレベルでデータを共有できます。これらのレベルには、データベース、スキーマ、テーブル、ビュー (通常ビュー、遅延バインディングビュー、マテリアライズドビューを含む)、および SQL ユーザー定義関数 (UDF) が含まれます。特定のデータベースに対して複数のデータ共有を作成できます。データ共有には、共有が作成されたデータベース内の複数のスキーマのオブジェクトを含めることができます。

データ共有にこの柔軟性を加えることで、詳細なアクセスコントロールを実現します。このコントロールは、Amazon Redshift データへのアクセスを必要とするさまざまなユーザーやビジネスに合わせて調整できます。

Amazon Redshift でのデータ共有の一貫性の管理

Amazon Redshift は、すべてのプロデューサクラスターおよびコンシューマークラスターでトランザクションの一貫性を提供し、データの最新で一貫性のあるビューをすべてのコンシューマーと共有します。

プロデューサークラスターのデータを継続的に更新できます。トランザクション内のコンシューマークラスターに対するすべてのクエリは、共有データの同じ状態を読み取ります。Amazon Redshift は、プロデューサークラスターでの別のトランザクションによって変更されたデータで、コンシューマークラスターでのトランザクションの開始後にコミットされたデータを考慮しません。プロデューサークラスターでデータ変更がコミットされた後、コンシューマークラスターでの新しいトランザクションは更新されたデータをすぐにクエリできます。

強力な一貫性により、データの共有中、無効な結果が含まれる可能性がある忠実度の低いビジネスレポートのリスクが排除されます。この要素は、財務分析や、機械学習モデルのトレーニングに使用されるデータセットを準備するために結果を使用する場合に特に重要です。

Amazon Redshift でデータ共有を使用する際の考慮事項

Amazon Redshift のデータ共有を操作する際の考慮事項は次のとおりです。データ共有の制限については、「データ共有に関する制限事項」を参照してください。

- リージョン間のデータ共有には、リージョン間のデータ転送料金が追加されます。これらのデータ転送料金は、同じリージョン内では適用されず、リージョン間の場合にのみ適用されます。詳細については、「クロスリージョンでのデータ共有でコスト管理を行う」を参照してください。
- データ共有からデータを読み取ると、ローカルクラスターデータベースに接続したままになります。データ共有から作成されたデータベースの設定と読み取りの詳細については、「データ共有オブジェクトのクエリ」および「Amazon Redshift Spectrum の外部データレイクテーブルのマテリアライズドビュー」を参照してください。
- プロデューサーのデータをクエリするために必要なすべてのコンピューティング料金とリージョン間のデータ転送料金は、コンシューマーに請求されます。プロデューサーは、プロビジョニングされたクラスターまたはサーバーレス名前空間のデータの基本ストレージに対して課金されます。
- 共有データに対するクエリのパフォーマンスは、コンシューマークラスターの計算能力によって異なります。

データ共有でのクラスター暗号化管理

AWS アカウント間でデータを共有するには、プロデューサークラスターとコンシューマークラスターの両方が暗号化されている必要があります。

Amazon Redshift では、クラスターに対してデータベースの暗号化を有効にして、保管中のデータを保護できます。クラスターに対して暗号化を有効にすると、クラスターとそのスナップショットのデータブロックとシステムメタデータが暗号化されます。クラスターの起動時に暗号化を有効にすることも、暗号化されていないクラスターを AWS Key Management Service (AWS KMS) 暗号化を使用するように変更することもできます。Amazon Redshift データベースの暗号化の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift データベースの暗号化」を参照してください。

転送中のデータを保護するために、すべてのデータは、プロデューサークラスターの暗号化スキーマを介して転送中に暗号化されます。この暗号化スキーマは、データのロード時にコンシューマークラ

スターにより導入されます。この後、コンシューマークラスターは、通常の暗号化クラスタとして動作します。プロデューサとコンシューマー間の通信も、共有されたキースキーマを使用して暗号化されます。転送時の暗号化の詳細については、「送信中の暗号化」を参照してください。

データ共有に関する制限事項

Amazon Redshift でデータ共有を操作する場合、以下の制限事項があります。

- データ共有は、プロビジョニングされたすべての RA3 クラスタータイプおよび Amazon Redshift Serverless でサポートされています。他のクラスタータイプではサポートされていません。
- プロデューサークラスターとコンシューマークラスターの両方とサーバーレス名前空間が同じアカウントにある場合は、同じ暗号化タイプ (両方とも暗号化されていない、または両方とも暗号化されている) である必要があります。Lake Formation のマネージドデータ共有を含む他のすべての場合、コンシューマーとプロデューサーの両方を暗号化する必要があります。これはセキュリティ上の目的です。ただし、同じ暗号キーを共有する必要はありません。
- SQL UDF はデータ共有を通じてのみ共有できます。Python と Lambda UDF はサポートされていません。
- プロデューサデータベースに特定の照合順序がある場合は、コンシューマーデータベースでも同様な照合順序設定を使用します。
- Amazon Redshift は、データ共有に対する外部スキーマやテーブルの追加、または外部テーブルの遅延バインディングビューの追加はサポートしていません。
- Amazon Redshift は、プロデューサークラスターでネストされたユーザー定義の SQL 関数をサポートしません。
- Amazon Redshift では、インターリーブされたソートキーを持つテーブルと、インターリーブされたソートキーを持つテーブルを参照するビューの共有はサポートされていません。
- コンシューマーは、データ共有オブジェクトを別のデータ共有に追加することはできません。さらに、コンシューマーはデータ共有オブジェクトを参照するビューを別のデータ共有に追加することはできません。
- Amazon Redshift は、アクセスの準備と実行の間に同時に DDL が発生した、データ共有オブジェクトへのアクセスをサポートしていません。
- Amazon Redshift は、データ共有によるストアドプロシージャの共有をサポートしていません。
- Amazon Redshift は、メタデータシステムビューとシステムテーブルの共有をサポートしていません。

データ共有が利用可能なリージョン

次の表は、利用可能なデータ共有機能の一覧です。

リージョン	同じリージョンでのデータ共有	リージョン間でのデータ共有	AWS Lake Formation 管理対象のデータ共有
米国東部 (バージニア北部) (us-east-1)	あり	はい	可能
米国東部 (オハイオ) (us-east-2)	あり	はい	可能
米国西部 (北カリフォルニア) (us-west-1)	あり	はい	可能
米国西部 (オレゴン) (us-west-2)	あり	はい	可能
アジアパシフィック (香港) (ap-east-1)	可能	いいえ	不可
アジアパシフィック (ムンバイ) (ap-south-1)	あり	はい	可能
アジアパシフィック (ハイデラバード) (ap-south-2)	可能	いいえ	不可
アジアパシフィック (東京) (ap-northeast-1)	あり	はい	可能
アジアパシフィック (シンガポール) (ap-southeast-1)	あり	はい	可能

リージョン	同じリージョンでのデータ共有	リージョン間でのデータ共有	AWS Lake Formation 管理対象のデータ共有
アジアパシフィック (シドニー) (ap-southeast-2)	あり	はい	可能
アジアパシフィック (ジャカルタ) (ap-southeast-3)	可能	いいえ	不可
アジアパシフィック (メルボルン) (ap-southeast-4)	可能	いいえ	不可
アジアパシフィック (ソウル) (ap-northeast-2)	あり	はい	可能
アジアパシフィック (大阪) (ap-northeast-3)	可能	いいえ	不可
中国 (北京) (cn-north-1)	可能	いいえ	不可
中国 (寧夏) (cn-northwest-1)	可能	いいえ	不可
アフリカ (ケープタウン) (af-south-1)	可能	いいえ	不可
カナダ西部 (カルガリー) (ca-west-1)	可能	いいえ	不可
カナダ (中部) (ca-central-1)	あり	はい	可能

リージョン	同じリージョンでのデータ共有	リージョン間でのデータ共有	AWS Lake Formation 管理対象のデータ共有
ヨーロッパ (フランクフルト) (eu-central-1)	あり	はい	可能
欧州 (チューリッヒ) (eu-central-2)	可能	いいえ	不可
欧州 (アイルランド) (eu-west-1)	あり	はい	可能
ヨーロッパ (ロンドン) (eu-west-2)	あり	はい	可能
欧州 (パリ) (eu-west-3)	あり	はい	可能
欧州 (ミラノ) (eu-south-1)	可能	いいえ	不可
欧州 (スペイン) (eu-south-2)	可能	いいえ	不可
欧州 (ストックホルム) (eu-north-1)	あり	はい	可能
中東 (UAE) (me-central-1)	可能	いいえ	不可
中東 (バーレーン) (me-south-1)	可能	いいえ	不可
イスラエル (テルアビブ) (il-central-1)	可能	いいえ	不可
南米 (サンパウロ) (sa-east-1)	あり	はい	可能

リージョン	同じリージョンでのデータ共有	リージョン間でのデータ共有	AWS Lake Formation 管理対象のデータ共有
AWS GovCloud (米国東部) (us-gov-east-1)	可能	いいえ	可能
AWS GovCloud (米国西部) (us-gov-west-1)	可能	いいえ	可能

データ共有のためのマルチウエアハウスの書き込みのリージョンでの可用性

PREVIEW_2023 トラックでは、データ共有は書き込み操作が可能で、さらに詳細な共有機能も備わっています。これらの設定方法についての詳細は、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。プレビュー機能を利用できるリージョンについて詳しくは、「データ共有が可能なリージョン (プレビュー)」を参照してください。

Amazon Redshift データ共有のデータ共有

データ共有は、Amazon Redshift でデータを共有する単位です。同じ AWS アカウントまたは異なる AWS アカウント間でデータを使用するには、データ共有を使用します。また、異なる Amazon Redshift クラスター間で、読み取りのためにデータを共有することもできます。

各データ共有は、Amazon Redshift クラスター内の特定のデータベースに関連付けられます。

プロデューサークラスター管理者は、データ共有を作成し、データベースオブジェクトを追加して、他のクラスターとデータを共有できます。これはアウトバウンド共有と呼ばれます。コンシューマー管理者は、他のクラスターからデータ共有を受け取ることができます。これはインバウンド共有と呼ばれます。プロデューサーとコンシューマーの詳細については、「データ共有のプロデューサーとコンシューマー」を参照してください。

データ共有オブジェクトは、クラスター上の特定のデータベースからのオブジェクトであり、プロデューサークラスターの管理者がデータ共有に追加してデータコンシューマーと共有することができます。データ共有オブジェクトは、データコンシューマーに対して読み取り専用です。データ共有オブジェクトの例には、テーブル、ビュー、およびユーザー定義関数があります。データ共有の作成中またはデータ共有の編集中に、いつでもデータ共有オブジェクトをデータ共有に追加できます。

クラスターがサイズ変更されるか、プロデューサークラスターが一時停止されても、データ共有は引き続き機能します。

データ共有にはさまざまな種類があります。

トピック

標準データ共有

標準データ共有を使用すると、プロビジョニングされたクラスター、サーバーレスワークグループ、アベイラビリティーゾーン、AWS アカウント、および AWS リージョン の間でデータを共有できます。クラスタータイプ間およびプロビジョニングされたクラスターと Amazon Redshift Serverless 間で共有できます。

データを共有するには、以下のプロビジョニング済みクラスター、サーバーレス名前空間、および AWS アカウント 識別子を書き留めます。

- プロビジョニングされたクラスターの名前空間は、Amazon Redshift でプロビジョニングされたクラスターを識別する識別子です。名前空間のグローバル一意識別子 (GUID) は、プロビジョニングされたクラスターの作成中に自動的に作成され、クラスターにアタッチされます。Amazon リソースネーム (ARN) の名前空間は、arn:{partition}:redshift:{region}:{account-id}:namespace:{namespace-guid} の形式です。プロビジョニングされたクラスターの名前空間は、Amazon Redshift コンソールのクラスター詳細ページで確認できます。

  データ共有ワークフローでは、AWS アカウント内にあるクラスターとのデータの共有に名前空間の GUID 値とクラスター名前空間 の ARN が使用されます。`current_namespace` 関数を使用して、現在のクラスターの名前空間を見つけることもできます。
- サーバーレス名前空間は、Amazon Redshift Serverless を識別する識別子です。名前空間のグローバル一意識別子 (GUID) は、Amazon Redshift Serverless の作成中に自動的に作成され、インスタンスにアタッチされます。サーバーレス名前空間 ARN は、arn:{partition}:redshift-serverless:{region}:{account-id}:namespace/{namespace-guid} の形式です。
- AWS アカウントはデータ共有のコンシューマーにすることができ、それぞれが 12 桁の AWS アカウント ID で表されます。

標準データ共有の場合、以下の点を考慮してください。

- プロデューサークラスターが削除されると、Amazon Redshift はプロデューサークラスターによって作成されたデータ共有を削除します。プロデューサークラスターをバックアップおよび復元しても、作成されたデータ共有は復元されたクラスターに残ります。ただし、他のクラスターに付与されたデータ共有のアクセス許可は、復元されたクラスターでは無効になります。データ共有の使用許可を目的のコンシューマークラスターに再付与します。コンシューマークラスター上のコンシューマデータベースは、スナップショットが作成された元のクラスターからのデータ共有を指します。復元されたクラスターから共有データをクエリするには、コンシューマークラスター管理者は別のデータベースを作成する必要があります。この管理者はまた、既存のコンシューマーデータベースを削除および再作成して、新しく復元されたクラスターのデータ共有を使用することもできます。
- コンシューマークラスターが、削除された後にスナップショットから復元された場合は、このクラスターに共有されていた以前のアクセスは無効になり、表示されなくなります。復元されたコンシューマークラスターで引き続きデータ共有へのアクセスが必要な場合、プロデューサクラスターの管理者は、復元されたコンシューマークラスターに対し、データ共有の使用を再度許可する必要があります。コンシューマークラスター管理者は、無効化されたデータ共有から作成された古いコンシューマデータベースを、すべて削除する必要があります。さらにこの管理者は、プロデューサーがアクセス許可を再付与した後に、データ共有からコンシューマーデータベースを再作成する必要があります。復元されたクラスターのクラスター名前空間 GUID は元のクラスターとは異なるため、コンシューマークラスターまたはプロデューサークラスターがバックアップから復元されたときにデータ共有の許可を再付与します。

AWS Data Exchange データ共有

AWS Data Exchange データ共有を使用して、Amazon Redshift データ共有の請求を管理できます。

AWS Data Exchange データ共有は、AWS Data Exchange 経由でデータを共有するためのライセンス供与の単位です。AWS は、AWS Data Exchange へのサブスクリプションと Amazon Redshift データ共有の使用に関連するすべての請求と支払いを管理します。承認されたデータプロバイダーは、AWS Data Exchange製品に AWS Data Exchange データ共有を追加することができます。御社の顧客が AWS Data Exchange データ共有を使用する製品にサブスクライブした場合、製品内のデータ共有にアクセスできるようになります。

AWS Data Exchange for Amazon Redshift を使用すると、Amazon Redshift データへの AWS Data Exchange を介したアクセスを許可するのに便利です。御社の顧客が AWS Data Exchange データ共有を使用する製品にサブスクライブする場合、AWS Data Exchangeではその顧客を、データ製品に

含まれるすべての AWS Data Exchange データ共有に対し、コンシューマーとして自動的に追加します。請求書は自動的に生成され、AWS Marketplace Entitlement Serviceを介して支払いの一元的な収集と、自動的な分配が行われます。

プロバイダーは、スキーマ、テーブル、ビュー、およびユーザー定義関数などの詳細なレベルで、Amazon Redshift のデータをライセンスできます。複数の AWS Data Exchange 製品間で、同じ AWS Data Exchange データ共有を使用できます。コンシューマーは、AWS Data Exchangeデータ共有に追加された任意のオブジェクトを利用することが可能です。プロデューサーはすべての AWS Data Exchange データ共有を、それらを管理している AWS Data Exchange に代わって表示できます。これには Amazon Redshift API オペレーション、SQL コマンド、および Amazon Redshift コンソールを使用します。製品の AWS Data Exchange データ共有にサブスクライブしている顧客は、そのデータ共有内のオブジェクトに対する、読み取り専用のアクセス権を持ちます。

サードパーティーのプロデューサーデータを利用しようとする顧客は、AWS Data Exchangeカタログを閲覧することで、Amazon Redshift のデータセットを検索しサブスクライブできます。AWS Data Exchange へのサブスクリプションがアクティブ化された後は、クラスター内のデータ共有からデータベースを作成したり、Amazon Redshift のデータに対しクエリを実行したりできます。

AWS Data Exchange データ共有のしくみ

プロデューサー管理者としての AWS Data Exchange データ共有の管理

データプロデューサー (AWS Data Exchange 上ではプロバイダとも呼ばれます) である場合は、Amazon Redshift データベースに接続する AWS Data Exchange データ共有を作成できます。AWS Data Exchange 上の製品に AWS Data Exchange データ共有を追加するには、AWS Data Exchangeのプロバイダーとして登録されている必要があります。

AWS Data Exchange でのデータ共有の使用開始方法については、「AWS Data Exchange でのライセンス付き Amazon Redshift データの共有」を参照してください。

コンシューマーとしてアクティブな AWS Data Exchange サブスクリプションを使用ながらの AWS Data Exchange データ共有の使用

有効な AWS Data Exchange サブスクリプションのあるコンシューマー (AWS Data Exchange 上ではサブスクライバとも呼ばれます) であれば、AWS Data Exchangeコンソールから AWS Data Exchange カタログを閲覧し AWS Data Exchange データ共有が使用可能な製品の検索が行えます。

AWS Data Exchange データ共有が含まれる製品にサブスクライブした後、クラスター内のデータ共有からデータベースを作成します。その後は、抽出、変換、およびロードを行わなくても Amazon Redshift のデータを直接クエリできようになります。

AWS Data Exchange でのデータ共有の使用開始方法については、「AWS Data Exchange でのライセンス付き Amazon Redshift データの共有」を参照してください。

AWS Data Exchange データ共有については、以下の点を考慮してください。

- プロデューサークラスターが削除されると、Amazon Redshift はプロデューサークラスターによって作成されたデータ共有を削除します。プロデューサークラスターをバックアップおよび復元しても、作成されたデータ共有は復元されたクラスターに残ります。データのサブスクライバが引き続きデータにアクセスできるようにするには、AWS Data Exchangeデータ共有を再度実行し、そのデータ共有を製品のデータセットに公開します。コンシューマークラスター上のコンシューマデータベースは、スナップショットが作成された元のクラスターからのデータ共有を指します。復元されたクラスターから共有データをクエリするには、コンシューマークラスターの管理者は別のデータベースを作成するか、既存のコンシューマデータベースを削除して再作成し、その AWS Data Exchange データ共有を新しく復元されたクラスターから使用します。
- コンシューマークラスターが削除された後にスナップショットから復元された場合も、このクラスターに共有されていた以前のアクセス権は有効のまま維持されるので表示が可能になります。コンシューマークラスターの管理者は、非アクティブなデータ共有から作成された古いコンシューマーデータベースをすべて削除し、プロデューサーが許可を再付与した後、データ共有からコンシューマーデータベースを再作成する必要があります。復元されたクラスターのクラスター名前空間 GUID は元のクラスターとは異なるため、プロデューサークラスターがバックアップから復元された際にデータ共有の許可を再付与します。
- AWS Data Exchange データ共有を使用している場合には、クラスターの削除は行わないように推奨します。このタイプの変更を実行すると、AWS Data Exchangeのデータ製品での使用条件に違反する可能性があります。

Amazon Redshift 向け AWS Data Exchange を使用する場合の考慮事項

Amazon Redshift 向け AWS Data Exchange を使用する場合、以下の点を考慮してください。

- プロデューサーとコンシューマーの両方が、Amazon Redshift のデータ共有を使用する際は、RA3 インスタンスタイプを使用する必要があります。プロデューサーの場合は、最新の Amazon Redshift クラスターバージョンで RA3 インスタンスタイプを使用する必要があります。
- プロデューサクラスターとコンシューマークラスターの両方が暗号化されている必要があります。
- AWS Data Exchange のデータ共有が使用可能なものを含む AWS Data Exchange 製品をリストするためには、AWS Data Exchangeプロバイダーとして登録されている必要があります。詳細については、「Getting started as a provider」を参照してください。

- AWS Data Exchange を介して Amazon Redshift データに対し検索、サブスクライブ、およびクエリを行う際は、AWS Data Exchangeプロバイダーとして登録する必要はありません。
- データへのアクセスを制御するには、パブリックなアクセスが可能な設定を有効にして AWS Data Exchange データ共有を作成します。AWS Data Exchange データ共有を変更してパブリックなアクセスが可能な設定をオフにするには、ALTER DATASHARE SET PUBLICACCESSIBLE FALSE を許可するようにセッション変数を設定します。詳細については、「ALTER DATASHARE の使用に関する注意事項」を参照してください。
- データ共有へのアクセス権は、AWS Data Exchangeデータ共有が使用可能な AWS Data Exchange 製品に対し有効なサブスクリプションがある場合に付与されます。そのためプロデューサーは、コンシューマーを手動で AWS Data Exchange データ共有に追加したり、そこから削除したりすることはできません。
- プロデューサは、コンシューマーが実行する SQL クエリを表示できません。プロデューサは、プロデューサーのみがアクセス可能な Amazon Redshift テーブルを介して、クエリの数やコンシューマーがクエリしているオブジェクトなどのメタデータのみを表示できます。詳細については、「Amazon Redshift でのデータ共有のモニタリングと監査」を参照してください。
- データ共有は、パブリックなアクセスが可能なように構成とすることをお勧めします。これを行わない場合、パブリックにアクセス可能なコンシューマークラスターを使用している AWS Data Exchange 上のサブスクライバーが、データ共有を使用できなくなります。
- 他の AWS アカウント と共有している AWS Data Exchange データ共有は、DROP DATASHARE ステートメントを使用して削除しないことをお勧めします。これを行うと、そのデータ共有へのアクセス権を持つ AWS アカウント が、データ共有にアクセスできなくなります。このアクションを元に戻すことはできません。このタイプの変更を実行すると、AWS Data Exchangeのデータ製品での使用条件に違反する可能性があります。AWS Data Exchange のデータ共有を削除する場合は、「DROP DATASHARE の使用に関する注意事項」を参照してください。
- クロスリージョンでのデータ共有の場合、AWS Data Exchangeデータ共有を作成すればライセンスされたデータを共有できます。
- 別のリージョンのデータを使用する場合、コンシューマーはプロデューサーリージョンからコンシューマーリージョンへのクロスリージョンデータ転送料金を支払います。

AWS Lake Formation 管理のデータ共有

Amazon Redshift では、AWS Lake Formation マネージドデータ共有を通じて、AWS アカウントと Amazon Redshift クラスター間でライブデータへのアクセスと共有が可能です。AWS Lake Formation データ共有を使用すると、データプロバイダーは Amazon S3 データレイクのライブデー

タを、他の AWS アカウントや Amazon Redshift クラスターを含む任意のコンシューマーと安全に共有できます。

AWS Lake Formation を使用すると、Amazon Redshift データ共有のデータベース、テーブル、列、行レベルのアクセス許可を一元的に定義して適用し、データ共有内のオブジェクトへのユーザーアクセスを制限できます。Lake Formation を介してデータを共有することで、Lake Formation でアクセス権を定義し、そのアクセス許可を任意のデータ共有とそのオブジェクトに適用できます。例えば、従業員情報を含んだテーブルがある場合、Lake Formation の列レベルのフィルターを使用して、人事部門に属さない従業員が社会保障番号などの個人を特定できる情報 (PII) を表示できないようにすることができます。データフィルターの詳細については、「AWS Lake Formation 開発者ガイド」の「Data Filtering and Cell-Level Security in Lake Formation」(Lake Formation におけるデータフィルタリングとセルレベルのセキュリティ) を参照してください。

Lake Formation でタグを使用して、Lake Formation のリソースに対するアクセス許可を設定することもできます。詳細については、「Lake Formation のタグベースのアクセスコントロール」を参照してください。

Amazon Redshift は現在、同じアカウント内または複数のアカウント間での共有は、Lake Formation 経由でのデータ共有をサポートしています。クロスリージョン共有は現在サポートされていません。

次に、Lake Formation を使用してデータ共有の許可を制御する方法の大まかな概要を示します。

1. Amazon Redshift では、プロデューサークラスターまたはワークグループの管理者は、プロデューサークラスターまたはワークグループ上にデータ共有を作成し、Lake Formation アカウントに対する使用許可の付与を行います。
2. プロデューサークラスターまたはワークグループの管理者は、Lake Formation アカウントにデータ共有にアクセスする承認を与えます。
3. Lake Formation 管理者がデータ共有を見つけて、登録します。また、アクセスできる AWS Glue ARN を見つけて、データ共有を AWS Glue Data Catalog ARN に関連付ける必要があります。AWS CLI を使用している場合、Redshift CLI 操作 `describe-data-shares` と `associate-data-share-consumer` でデータ共有を見つけて承認できます。データ共有を登録するには、Lake Formation CLI 操作 `register-resource` を使用します。
4. Lake Formation 管理者は、AWS Glue Data Catalog にフェデレーティッドデータベースを作成し、データ共有内のオブジェクトへのユーザーアクセスを制御する Lake Formation アクセス許可を設定します。AWS Glue のフェデレーティッドデータベースの詳細については、「Amazon Redshift データ共有でのデータに対するアクセス許可の管理」を参照してください。
5. Lake Formation 管理者は、アクセスできる AWS Glue データベースを見つけて、データ共有を AWS Glue Data Catalog ARN に関連付けます。

6. Redshift 管理者は、アクセスできる AWS Glue データベース ARN を見つけて、AWS Glue データベース ARN を使用して Amazon Redshift コンシューマークラスターに外部データベースを作成し、IAM 認証情報で認証されたデータベースユーザーに Amazon Redshift データベースへのクエリを開始する使用権を付与します。
7. データベースユーザーは、SVV_EXTERNAL_TABLES ビューと SVV_EXTERNAL_COLUMNS ビューを使用して、アクセス許可のある AWS Glue データベース内のすべてのテーブルや列を検索し、AWS Glue データベースのテーブルをクエリできます。
8. プロデューサークラスターまたはワークグループの管理者がコンシューマークラスターとデータを共有しないことを決定した場合、プロデューサークラスター管理者は Redshift からデータ共有について使用を取り消し、承認を解除し、または削除できます。Lake Formation 内に関連付けられたアクセス許可とオブジェクトは自動的には削除されません。

プロデューサークラスターまたはワークグループの管理者として、AWS Lake Formation でデータ共有を行う方法の詳細については、「プロデューサーとして Lake Formation 管理のデータ共有を使用する」を参照してください。プロデューサークラスターまたはワークグループの共有データを使用するには、「コンシューマーとしての Lake Formation 管理のデータ共有を使用する」を参照してください。

Amazon Redshift で AWS Lake Formation を使用する場合の考慮事項と制限

Lake Formation を使用して Amazon Redshift のデータを共有する際の考慮事項と制限事項は次のとおりです。データ共有の考慮事項と制限事項については、「Amazon Redshift でのデータ共有に関する考慮事項」を参照してください。Lake Formation の制限事項については、「Lake Formation での Amazon Redshift データ共有の使用に関する注意事項」を参照してください。

- リージョン間での Lake Formation へのデータ共有は、現在サポートされていません。
- 共有リレーションでユーザーに対して列レベルのフィルターが定義されている場合、`SELECT *` 操作を実行すると、ユーザーがアクセスできる列のみが返されます。
- Lake Formation のセルレベルのフィルターはサポートされていません。
- ビューとそのテーブルを作成して Lake Formation と共有した場合、フィルターを設定してテーブルへのアクセスを管理できますが、Amazon Redshift は、コンシューマークラスターのユーザーが共有オブジェクトにアクセスするとき、Lake Formation が定義したポリシーを適用します。ユーザーが Lake Formation と共有されているビューにアクセスすると、Redshift は定義されている Lake Formation ポリシーをビューにのみ適用し、ビューに含まれるテーブルには適用しません。ただし、ユーザーがテーブルに直接アクセスすると、Redshift は定義済みの Lake Formation ポリシーをテーブルに適用します。

- テーブルに Lake Formation フィルターが設定されている場合、共有テーブルに基づいてコンシューマーのマテリアライズドビューを作成することはできません。
- Lake Formation 管理者は、データレイク管理者アクセス許可と、データ共有を受け入れるために必要なアクセス許可を持っている必要があります。
- Lake Formation を介してデータ共有を行うには、プロデューサーコンシューマークラスターは、最新の Amazon Redshift クラスターバージョンまたはサーバーレスワークグループを持つ RA3 クラスターである必要があります。
- プロデューサクラスターとコンシューマークラスターの両方が暗号化されている必要があります。
- プロデューサークラスターまたはワークグループに実装されている Redshift の行レベルと列レベルのアクセス制御ポリシーは、データ共有が Lake Formation と行われる場合は無視されます。Lake Formation の管理者は、Lake Formation でこれらのポリシーを設定する必要があります。プロデューサークラスターまたはワークグループの管理者は、ALTER TABLE コマンドを使用して、テーブルの RLS を無効にできます。
- Lake Formation によるデータ共有は、Redshift と Lake Formation の両方にアクセスできるユーザーのみが利用できます。

データ共有のプロデューサーとコンシューマー

データプロデューサー (データ共有プロデューサーまたはデータシェアプロデューサーとも言う) は、データを共有するクラスターです。プロデューサークラスターの管理者とデータベースの所有者は、CREATE DATASHARE コマンドを使用してデータ共有を作成できます。そこにスキーマ、テーブル、ビュー、SQL ユーザー定義関数 (UDF) などのオブジェクトをデータベースから追加すると、そのデータベースのプロデューサークラスターとコンシューマークラスターが、データを共有できるようになります。

AWS Data Exchange データ共有のためのデータプロデューサー (AWS Data Exchange 上ではプロバイダーとも呼ばれます) は、AWS Data Exchangeを介してデータをライセンスできます。承認されたプロバイダーは、AWS Data Exchange製品に AWS Data Exchange データ共有を追加できます。

AWS Data Exchange データ共有が含まれる製品を顧客がサブスクライブすると、AWS Data Exchangeはその顧客をデータコンシューマーとして、製品に含まれるすべての AWS Data Exchange データ共有に自動的に追加します。AWS Data Exchange は、顧客のサブスクリプションが終了すると、AWS Data Exchangeデータ共有からそれらの顧客全員を削除します。AWS Data Exchange は、AWS Data Exchangeデータ共有が含まれる有料製品に関する料金の請求、請求書の発行、支払いの回収、および支払いの分配も自動的に管理します。詳細については、「AWS Data

Exchange データ共有」を参照してください。AWS Data Exchange のデータプロバイダとしての登録については、「Getting started as a provider」を参照してください。

データコンシューマ (データ共有コンシューマまたはデータシェアコンシューマとも言う) は、プロデューサクラスターからデータ共有を受信するクラスターです。

データを共有する Amazon Redshift の各クラスターは、同じあるいは別の AWS アカウントに置くことも、異なる AWS リージョンに置くこともできます。これにより、組織間でのデータ共有や、他のチームとの共同作業が可能になります。コンシューマークラスターの管理者は、使用が許可されているデータ共有を受け取り、各データ共有の内容を確認します。共有データを使用するために、コンシューマークラスターの管理者はデータ共有から Amazon Redshift データベースを作成します。次に、管理者はデータベースに対するアクセス許可をコンシューマークラスター内のユーザーおよびロールに割り当てます。アクセス許可が付与されると、ユーザーとロールは、コンシューマークラスター上のローカルデータとともに、標準のメタデータクエリの一部として共有オブジェクトを一覧表示できます。すぐにクエリを開始できます。

アクティブな AWS Data Exchange サブスクリプションを持つコンシューマー (AWS Data Exchange ではサブスクライバーとも呼ばれます) である場合は、データを抽出、変換、ロードすることなく、詳細かつ最新の Amazon Redshift データを検索、サブスクライブ、クエリすることが可能です。詳細については、「AWS Data Exchange データ共有」を参照してください。

Amazon Redshift データ共有の詳細

Amazon Redshift では、Amazon Redshift クラスター間でデータをコピーまたは転送することなく、安全にデータを共有できます。Amazon Redshift データ共有を使用すると、ソースデータの更新を含むライブクエリ結果を、同じまたは異なる AWS アカウントおよび AWS リージョンの Amazon Redshift クラスターと共有できます。

トピック

Amazon Redshift データ共有のステータス値

このトピックでは、データ共有が Amazon Redshift で持つことができるステータスについて説明します。

アカウント間データ共有には、対応が必要となるさまざまなステータスがあります。データ共有では、アクティブ、アクションが必要、または非アクティブの各ステータスを取ります。

以下で、各データシェアのステータスと、それらに対して必要なアクションについて説明します。

- プロデューサークラスターの管理者がデータ共有を作成すると、そのプロデューサークラスターでのデータ共有は、承認保留中のステータスとなります。プロデューサークラスターの管理者は、データコンシューマーに対し、データ共有へのアクセスを許可できます。コンシューマークラスターの管理者が行うアクションはありません。
- プロデューサークラスターの管理者がデータ共有を承認すると、そのプロデューサークラスター上のデータ共有は、ステータスが 承認済み に変わります。プロデューサクラスターの管理者が行うアクションはありません。データ共有に、データコンシューマーとの関連付けが少なくとも 1 つある場合、データ共有のステータスは承認済みからアクティブに遷移します。

  その後、コンシューマークラスター上のデータ共有のステータスは、利用可能 (Amazon Redshift コンソールでのアクションが必要) に変わります。コンシューマークラスターの管理者は、データ共有とデータコンシューマーを関連付けることも、そのデータ共有を拒否することもできます。また、コンシューマークラスターの管理者は、AWS CLIコマンドの `describeDatashareforConsumer` を使用して、データ共有のステータスを表示することもできます。さらに、この管理者は、CLI コマンドの `describeDatashare` を使用して Amazon リソースネーム (ARN) を指定しながら、対象のデータ共有のステータスを表示することも可能です。
- コンシューマークラスターの管理者が、データ共有をデータコンシューマーに関連付けると、プロデューサークラスター上のデータ共有のステータスがアクティブに遷移します。データ共有に、データコンシューマーとの関連付けが少なくとも 1 つある場合、データ共有のステータスは承認済みからアクティブに遷移します。プロデユーサクラスターの管理者に必要なアクションはありません。

  コンシューマークラスター上で、データ共有のステータスがアクティブに変わります。コンシューマークラスターの管理者に必要なアクションはありません。
- コンシューマークラスターの管理者が、データ共有のコンシューマーとの関連付けを削除すると、そのデータ共有のステータスは、アクティブまたは承認済みに変わります。そのデータ共有において、他のデータコンシューマーとの関連付けが少なくとも 1 つ存在する場合、ステータスはア

クティブになります。データ共有のコンシューマーとの関連付けがまったく存在しない場合、プロデューサークラスター上のステータスは承認済みに遷移します。プロデューサークラスターの管理者が行うアクションはありません。

すべての関連付けが削除された場合、コンシューマークラスター上のデータ共有のステータスは、アクションが必要に遷移します。コンシューマークラスターの管理者は、データ共有がコンシューマーに対して有効であれば、データ共有をそのデータコンシューマーに再度関連付けることができます。

- コンシューマークラスターの管理者がデータ共有を拒否すると、プロデューサークラスター上のデータ共有のステータスはアクションが必要に、またコンシューマークラスターでは拒否にそれぞれ遷移します。プロデューサークラスターの管理者は、データ共有を再承認できます。コンシューマークラスターの管理者が行うアクションはありません。
- プロデューサークラスターの管理者が、データ共有の承認を削除すると、プロデューサークラスター上のデータ共有のステータスは、アクションが必要に変化します。プロデューサークラスターの管理者は、必要に応じてデータ共有を再承認することを選択できます。コンシューマークラスターの管理者に必要なアクションはありません。

クラスター内およびクラスター間のデータ共有

データ共有は、異なる Amazon Redshift のプロビジョニングされたクラスター間またはサーバーレスワークグループでデータを共有する場合にのみ必要です。同じクラスター内で、他のデータベース内のオブジェクトに対して必要なアクセス許可を持っている限り、単純な 3 つの部分からなる表記 `database.schema.table` を使用して別のデータベースにクエリを実行できます。

Amazon Redshift でのデータ共有に対する許可の管理

プロデューサークラスターの管理者は、共有するデータセットの制御を保持します。新しいオブジェクトをデータ共有に追加したり、データ共有から削除したりできます。複数のコンシューマークラスター、AWSアカウント、または AWS リージョンに対して、データ共有へのアクセス権をまとめて付与または取り消すこともできます。許可が失効すると、コンシューマークラスターは直ちに共有オブジェクトへのアクセス権を失い、それらのオブジェクトは、SVV_DATASHARES の INBUND データ共有の一覧に表示されなくなります。

次の例では、データ共有 `salesshare` を作成し、スキーマ `public` を追加して、テーブル `public.tickit_sales_redshift` を `salesshare` に追加します。これは、指定したクラスター名前空間に対して `salesshare` の使用許可も付与します。

```
CREATE DATASHARE salesshare;
```

```
ALTER DATASHARE salesshare ADD SCHEMA public;

ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

CREATE DATASHARE の場合、スーパーユーザーとデータベース所有者はデータ共有を作成できます。詳細については、「CREATE DATASHARE」を参照してください。ALTER DATASHARE の場合、追加または削除するデータ共有オブジェクトに対して必要な許可を持つデータ共有の所有者は、データ共有を変更できます。詳細については、ALTER DATASHAREを参照してください。

プロデューサー管理者として、データ共有を削除すると、コンシューマークラスターに一覧表示されなくなります。削除されたデータ共有からコンシューマークラスター上に作成されたデータベースとスキーマリファレンスは、オブジェクトなしで引き続き存在します。コンシューマークラスターの管理者は、これらのデータベースを手動で削除する必要があります。

コンシューマー側では、コンシューマークラスター管理者は、データ共有からデータベースを作成することによって、共有データにアクセスする必要があるユーザーとロールを決定できます。データベースの作成時に選択したオプションに応じて、データベースへのアクセスを次のように制御できます。データ共有からデータセットを作成する方法の詳細については、「CREATE DATABASE」を参照してください。

データ共有の設定とコンシューマーからのデータの読み取りの詳細については、「AWS アカウント内のデータへの読み取りアクセスの共有」を参照してください。

WITH PERMISSIONS 句を使用せずにデータベースを作成する

管理者は、データベースレベルまたはスキーマレベルでアクセスを制御できます。スキーマレベルでアクセスを制御するには、管理者はデータ共有から作成された Amazon Redshift データベースから外部スキーマを作成する必要があります。

以下の例では、データベースレベルおよびスキーマレベルで、共有されたテーブルにアクセスする許可を付与します。

```
GRANT USAGE ON DATABASE sales_db TO Bob;

CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE sales_db SCHEMA 'public';

GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

アクセスをさらに制限するために、共有オブジェクトの上にビューを作成し、必要なデータのみを公開することができます。その後、これらのビューを使用して、ユーザーとロールへのアクセスを許可できます。

ユーザーがデータベースまたはスキーマへのアクセスを許可されると、そのデータベースまたはスキーマ内のすべての共有オブジェクトにアクセスできるようになります。

WITH PERMISSIONS 句を使用してデータベースを作成する

データベースまたはスキーマの使用権限を付与すると、管理者は、ローカルのデータベースまたはスキーマでアクセス許可を付与するのと同じアクセス許可付与プロセスを使用してアクセス制御を追加できます。個別のオブジェクトのアクセス許可がないと、ユーザーは、USAGE アクセス許可を付与された後でも、データ共有データベースまたはスキーマ内のオブジェクトにアクセスできません。

以下の例では、データベースレベルで、共有されたテーブルにアクセスする許可を付与します。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
GRANT USAGE FOR SCHEMAS IN DATABASE sales_db TO Bob;
GRANT SELECT ON sales_db.public.tickit_sales_redshift TO Bob;
```

データベースまたはスキーマへのアクセス許可が付与された後でも、アクセスを認めるデータベースまたはスキーマ内のすべてのオブジェクトについての関連アクセス許可をユーザーに付与する必要があります。

WITH PERMISSIONS を使用した詳細な共有設定

クラスターまたはサーバーレスワークグループによるデータ共有のクエリの有効化

このステップでは、データ共有がアカウント内の別のクラスターまたは Amazon Redshift Serverless 名前空間から送信されているか、別のアカウントから送信され、使用している名前空間に関連付けられていることを前提としています。

1. コンシューマーデータベースの管理者は、データ共有からデータベースを作成できます。

   ```
   CREATE DATABASE my_ds_db [WITH PERMISSIONS] FROM DATASHARE my_datashare OF
    NAMESPACE 'abc123def';
   ```

   WITH PERMISSIONS を使用してデータベースを作成すると、データ共有オブジェクトに対する詳細なアクセス許可をさまざまなユーザーやロールに付与できます。これを行わないと、データ共有データベースの USAGE アクセス許可を付与されたすべてのユーザーとロールに、データ共有データベース内のすべてのオブジェクトに対するすべてのアクセス許可が付与されます。

2. 以下では、Redshift データベースユーザーまたはロールにアクセス許可を付与する方法を示しています。これらのステートメントにローカルデータベースを接続できません。GRANT ステートメントの実行前にデータ共有データベースで USE コマンドを実行すると、これらのステートメントは実行できません。

```
GRANT USAGE ON DATABASE my_ds_db TO ROLE data_eng;
GRANT CREATE, USAGE ON SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;
GRANT ALL ON ALL TABLES IN SCHEMA my_ds_db.my_shared_schema TO ROLE data_eng;

GRANT USAGE ON DATABASE my_ds_db TO bi_user;
GRANT USAGE ON SCHEMA my_ds_db.my_shared_schema TO bi_user;
GRANT SELECT ON my_ds_db.my_shared_schema.table1 TO bi_user;
```

Amazon Redshift データ共有でのビューの使用

プロデューサークラスターは、通常ビュー、遅延バインディングビュー、マテリアライズドビューを共有できます。通常ビューまたは遅延バインディングビューを共有する場合、ベーステーブルを共有する必要はありません。以下のテーブルは、データ共有でビューがどのようにサポートされるかを示しています。

ビュー名	このビューをデータ共有に追加できますか?	コンシューマーは、クラスター全体のデータ共有オブジェクトに対してこのビューを作成できますか?
通常ビュー	はい	不可
遅延バインドビュー	あり	はい
マテリアライズドビュー	はい	はい。ただし、完全に更新した場合に限ります

次のクエリは、データ共有でサポートされている通常のビューの出力を示しています。定期的なビューの定義については、「CREATE VIEW」を参照してください。

```
SELECT * FROM tickit_db.public.myevent_regular_vw
ORDER BY eventid LIMIT 5;

   eventid  |  eventname
  ----------+-------------
     3835   | LeAnn Rimes
     3967   | LeAnn Rimes
     4856   | LeAnn Rimes
     4948   | LeAnn Rimes
     5131   | LeAnn Rimes
```

次のクエリは、データ共有でサポートされている遅延バインディングビューの出力を示しています。遅延バインドビューの定義については、「CREATE VIEW」 を参照してください。

```
SELECT * FROM tickit_db.public.event_lbv
ORDER BY eventid LIMIT 5;

 eventid | venueid | catid | dateid |          eventname            |      starttime
 --------+---------+-------+--------+-------------------------------
+---------------------
     1   |   305   |   8   |  1851  |        Gotterdammerung        | 2008-01-25
 14:30:00
     2   |   306   |   8   |  2114  |           Boris Godunov       | 2008-10-15
 20:00:00
     3   |   302   |   8   |  1935  |              Salome           | 2008-04-19
 14:30:00
     4   |   309   |   8   |  2090  |  La Cenerentola (Cinderella) | 2008-09-21
 14:30:00
     5   |   302   |   8   |  1982  |            Il Trovatore       | 2008-06-05
 19:00:00
```

次のクエリは、データ共有でサポートされているマテリアライズドビューの出力を示しています。マテリアライズドビューの定義については、「CREATE MATERIALIZED VIEW」を参照してください。

```
SELECT * FROM tickit_db.public.tickets_mv;

  catgroup | qtysold
 ----------+---------
  Concerts | 195444
    Shows  | 149905
```

プロデューサークラスター内のすべてのテナントで共通テーブルを維持できます。tenant_id (account_id または namespace_id) などのディメンション列でフィルタリングされたデータのサブセットをコンシューマークラスターと共有することもできます。これを行うには、これらの ID 列 (current_aws_account = tenant_id など) にフィルターを使用して、ベーステーブルにビューを定義できます。コンシューマー側では、ビューをクエリすると、アカウントに適格な行のみが表示されます。これを行うには、Amazon Redshift コンテキスト関数 current_aws_account および current_namespace を使用できます。

次のクエリは、現在の Amazon Redshift クラスターが存在するアカウントの ID を返します。Amazon Redshift に接続している場合は、このクエリを実行できます。

```
select current_user, current_aws_account;

current_user | current_aws_account
-------------+--------------------
dwuser       |    111111111111
(1row)
```

次のクエリは、現在の Amazon Redshift クラスターの名前空間を返します。データベースに接続している場合は、このクエリを実行できます。

```
select current_user, current_namespace;

current_user | current_namespace
-------------+--------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8
(1 row)
```

データ共有内のマテリアライズドビューの増分更新

Amazon Redshift は、ベーステーブルが共有されている場合、コンシューマーデータ共有におけるマテリアライズドビューの増分更新をサポートします。増分更新は、Amazon Redshift が前回の更新後に発生したベーステーブルの変更を特定し、マテリアライズドビューの対応するレコードのみを更新する操作です。この動作の詳細については、「CREATE MATERIALIZED VIEW」を参照してください。

IAM ポリシーを使用したデータ共有 API オペレーションへのアクセスの管理

データ共有 API 操作へのアクセスを制御するには、IAM アクションベースのポリシーを使用します。IAM ポリシーの管理方法については、「IAM ユーザーガイド」の「IAM ポリシーの管理」を参照してください。

データ共有 API オペレーションを使用するために必要な許可については、「Amazon Redshift 管理ガイド」の「データ共有 API オペレーションを使用するために必要な許可」を参照してください。

クロスアカウントデータ共有をより安全にするためには、API オペレーションの `AuthorizeDataShare` および `DeauthorizeDataShare` で条件付きキー `ConsumerIdentifier` が使用できます。これを利用することで、どの AWS アカウント が 2 つの API オペレーションを呼び出すことができるのかを明示的に制御できます。

自身のアカウント以外のコンシューマーに対して、データ共有の承認を拒否したり承認を解除したりできます。これを行うには、IAM ポリシー内で AWS アカウント の数を指定します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringNotEquals": {
                    "redshift:ConsumerIdentifier": "555555555555"
                }
            }
        }
    ]
}
```

DataShareArn **testshare2** でプロデューサーを許可して、IAM ポリシーの AWS アカウント番号が 111122223333 のコンシューマーと明示的に共有することができます。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:666666666666:datashare:af06285e-8a45-4ee9-b598-648c218c8ff1/testshare2",
            "Condition": {
                "StringEquals": {
                    "redshift:ConsumerIdentifier": "111122223333"
                }
            }
        }
    ]
}
```

データ共有のクエリ

Amazon Redshift での共有データへのアクセス

標準の SQL インターフェイス、JDBC または ODBC ドライバー、および Data API を使用して、共有データを検出できます。使い慣れたビジネスインテリジェンス (BI) や分析ツールを使用して、高性能でデータをクエリすることもできます。クエリを実行するには、アクセス許可があるクラスターに対してローカルおよびリモートの両方にある他の Amazon Redshift データベースのオブジェクトを参照します。

これは、クラスター内のローカルデータベースに接続したままにすることで簡単に実行できます。その後、共有データを使用するために、データ共有からコンシューマーデータベースを作成します。

完了したら、コンシューマーデータベースに接続し、部分表記を使用して共有オブジェクトをクエリできます。または、ローカルクラスターデータベースに接続し、3 つの部分からなる表記 (*consumer_database_name*.*schema_name*.*table_name*) を使用してコンシューマーデータベース内のオブジェクトをクエリできます。また、コンシューマーデータベース内にあるスキーマへの外部スキーマリンクを使用して、クエリを実行することもできます。同じクエリの中で、ローカルデータと、他のクラスターから共有されているデータの両方をクエリできます。このクエリでは、現在接続されているデータベースに加え、データ共有から作成されたコンシューマーデータベースなど、接続されていない他のデータベースからのオブジェクトも参照することが可能です。

Amazon Redshift でのデータ共有のメタデータへのアクセス

クラスターの管理者がデータ共有を検出できるように、Amazon Redshift はデータ共有を一覧表示するための一連のメタデータビューを提供します。これらのビューには、クラスターで作成されたデータ共有、同じアカウント内の他のクラスターから受信したデータ共有、さらに他の AWS リージョンから受信したデータ共有が一覧表示されます。これらのビューには、以下の情報が表示されます。

- クラスターによって共有および受信されるデータ共有
- 基本的な共有メタデータ、オブジェクト、コンシューマーなど、データ共有内のデータベースオブジェクトのコンテンツ

SHOW SCHEMAS を使用して、接続されたデータベースに関連付けられたデータ共有内の共有スキーマのリストを表示します。詳細については、「SHOW SCHEMAS」を参照してください。

SHOW TABLES を使用して、接続されたデータベースに関連付けられたデータ共有の共有スキーマ内のテーブルのリストを表示します。詳細については、「SHOW TABLES」を参照してください。

SHOW COLUMNS を使用して、接続されたデータベースに関連付けられたデータ共有内の共有テーブルの列のリストを表示します。詳細については、「SHOW COLUMNS」を参照してください。

SVV_DATASHARES を使用すると、クラスターで作成され (アウトバウンド)、他のクラスターから共有された (インバウンド) すべてのデータ共有のリストを表示します。詳細については、「SVV_DATASHARES」を参照してください。

SVV_DATASHARE_CONSUMERS を使用すると、データコンシューマーのリストを表示できます。詳細については、「SVV_DATASHARE_CONSUMERS」を参照してください。

SVV_DATASHARE_OBJECTS を使用すると、クラスター内に作成され (アウトバウンド)、他のユーザーから共有される (インバウンド) すべてのデータ共有内のオブジェクトのリストを表示できます。詳細については、「SVV_DATASHARE_OBJECTS」を参照してください。

Amazon Redshift データ共有のビジネスインテリジェンスツールとの統合

データ共有をビジネスインテリジェンス (BI) ツールと統合するには、Amazon Redshift JDBC または ODBC ドライバーを使用することをお勧めします。

Amazon Redshift JDBC および ODBC ドライバーは、ドライバーで `GetCatalogs` API オペレーションをサポートします。これにより、データ共有から作成されたデータベースを含むすべてのデータベースのリストが返されます。ドライバーは、`GetCatalogs` が返すすべてのデータベースからデータを返す `GetSchemas`、`GetTables` などのダウンストリームオペレーションもサポートしま

す。ドライバーは、カタログが呼び出しで明示的に指定されていない場合でも、このサポートを提供します。JDBC または ODBC ドライバーの詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift での接続設定」を参照してください。

データ共有から作成されたコンシューマーデータベースには、他のデータベースと同様に直接接続できます。ツールに接続切り替えユーザーインターフェイスがある場合、データベースのリストにはデータ共有から作成されたデータベースが表示されます。これらのコンシューマーデータベースは、SVV_REDSHIFT_DATABASES を使用して検出できます。

Amazon Redshift でのデータ共有のモニタリングと監査

Amazon Redshift では、データ共有アクティビティをモニタリングおよび監査して、コンプライアンスとセキュリティを確保できます。

データ共有を監査することで、プロデューサーはデータ共有の進化を追跡できます。例えば監査は、データ共有の作成、オブジェクトの追加や削除、および許可の付与や取り消しが、Amazon Redshift クラスターや AWS アカウントまたは AWS リージョンに対して行われたタイミングを追跡するために役立ちます。

監査に加えて、プロデューサーとコンシューマーは、アカウントレベル、クラスターレベル、およびオブジェクトレベルなど、さまざまな粒度でデータ共有の使用状況を追跡します。使用状況の追跡とビューの監査に関する詳細については、「SVL_DATASHARE_CHANGE_LOG」および「SVL_DATASHARE_USAGE_PRODUCER」を参照してください。

システムビューをクエリすることで、データ共有をモニタリングできます。

1. データを共有しようとするプロデューサークラスターの管理者は、Amazon Redshift のデータ共有を作成します。その上で、プロデューサークラスター管理者は、必要なデータベースオブジェクトを追加します。これらは、データ共有に対するスキーマ、テーブル、およびビューであり、オブジェクトの共有相手となるコンシューマーのリストを指定しています。

   以下のシステムビューを使用すると、プロデューサークラスターやコンシューマークラスターのデータ共有の変更や使用状況を追跡できる統合ビューが表示されます。

   - SYS_DATASHARE_CHANGE_LOG
   - SYS_DATASHARE_USAGE_CONSUMER
   - SYS_DATASHARE_USAGE_PRODUCER

   次のシステムビューを使用して、アウトバウンドデータ共有のデータ共有オブジェクトおよびデータコンシューマー情報を確認します。

- SVV_DATASHARES
- SVV_DATASHARE_CONSUMERS
- SVV_DATASHARE_OBJECTS

2. コンシューマークラスターの管理者は、使用が許可されているデータ共有を確認し、SVV_DATASHARESを使用してインバウンドデータ共有を表示することにより、各データ共有のコンテンツを確認します。

   共有データを使用するために、各コンシューマークラスターの管理者はデータ共有から Amazon Redshift データベースを作成します。次に、管理者はコンシューマークラスター内の適切なユーザーおよびロールにアクセス許可を割り当てます。ユーザーとロールは、次のメタデータシステムのビューを表示することにより、標準メタデータクエリの一部として共有オブジェクトを一覧表示できます。また、データのクエリをすぐに開始できます。

   - SVV_REDSHIFT_COLUMNS
   - SVV_REDSHIFT_DATABASES
   - SVV_REDSHIFT_FUNCTIONS
   - SVV_REDSHIFT_SCHEMAS
   - SVV_REDSHIFT_TABLES

   Amazon Redshift のローカルスキーマ、共有スキーマ、および外部スキーマの両方のオブジェクトを表示するには、次のメタデータシステムビューを使用してクエリを実行します。

   - SVV_ALL_COLUMNS
   - SVV_ALL_SCHEMAS
   - SVV_ALL_TABLES

コンシューマーデータベースに接続すると、クロスデータベース検出は無効になります。メタデータシステムビューは、接続されたデータベースに関連付けられているデータ共有内の共有オブジェクトのメタデータのみを返します。

AWS CloudTrail との Amazon Redshift データ共有の統合

データ共有は AWS CloudTrail と統合されています。CloudTrail は、Amazon Redshift でユーザー、ロール、または AWS のサービスが実行したアクションの記録を提供するサービスです。CloudTrail は、データ共有に対するすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、AWS CloudTrail コンソールからのコールと、データ共有オペレーションへのコード

コールが含まれます。Amazon Redshift と AWS CloudTrail の統合の詳細については、「CloudTrail によるログ記録」を参照してください。

CloudTrail の詳細については、「CloudTrail の仕組み」を参照してください。

データ共有タスクの管理

Amazon Redshift では、Amazon Redshift クラスター間でデータを安全に共有できるため、データコンシューマーは、ライブデータへのクエリやアクセスをコピーまたは複製することなく行えます。データ共有を使用すると、共有するデータベースオブジェクトを参照するプロデューサー側のオブジェクトであるデータ共有を作成および設定できます。

データ共有を開始するには、SQL インターフェイスまたは Amazon Redshift コンソールのいずれかを使用します。

トピック

- SQL インターフェースによるデータ共有の管理
- コンソールを使用したデータ共有の管理
- AWS CloudFormation によるデータ共有の管理
- コンソールを使った書き込みを伴うデータ共有の管理 (プレビュー)

SQL インターフェースによるデータ共有の管理

Amazon Redshift では、SQL インターフェイスを使用して、データベースとクラスター間のデータアクセスと共有を制御できます。データ共有を管理することで、プロデューサーデータベースと 1 つ以上のコンシューマーデータベースまたはクラスター間のプロデューサー - コンシューマー関係であるデータ共有を作成できます。

AWS アカウントの内部またはその間で、あるいは AWS リージョン間で、異なる Amazon Redshift クラスターに置かれたデータを、読み取り目的で共有することができます。

トピック

- AWS アカウント 内のデータへの読み取りアクセスの共有
- データへの書き込みアクセスの共有 (プレビュー)
- AWS アカウント間のデータの共有
- AWS リージョン間のデータの共有

- AWS Data Exchange でのライセンス付き Amazon Redshift データの共有
- AWS Lake Formation 管理のデータ共有の使用

AWS アカウント 内のデータへの読み取りアクセスの共有

Amazon Redshift では、同じ AWS アカウント内の異なるデータベースユーザーまたはグループ間でデータへの読み取りアクセスを共有できます。この機能を使用すると、データへのアクセス許可をきめ細かなレベルで制御し、許可されたユーザーまたはグループのみが特定のデータセットを読み取ることができます。

プロデューサークラスターの管理者またはデータベース所有者として、読み取り目的でデータを共有するには

1. クラスターにデータ共有を作成します。詳細については、「CREATE DATASHARE」を参照してください。

   ```
   CREATE DATASHARE salesshare;
   ```

   クラスターのスーパーユーザーとデータベースの所有者は、データ共有を作成できます。各データ共有は、作成時にデータベースに関連付けられます。そのデータベースのオブジェクトのみがそのデータ共有で共有できます。同じデータベース上に、同じ粒度または異なる粒度のオブジェクトを使用して、複数のデータ共有を作成できます。クラスターが作成できるデータ共有の数に制限はありません。

   Amazon Redshift コンソールを使用してデータ共有を作成することもできます。詳細については、「データ共有の作成」を参照してください。

2. データ共有を操作するための権限を委任します。詳細については、「GRANT」または「REVOKE」を参照してください。

   次の例では、salesshare上の dbuser にアクセス許可を付与しています。

   ```
   GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
   ```

   クラスターのスーパーユーザーとデータ共有の所有者は、追加のユーザーに対してデータ共有の変更許可の付与または取り消しを実行できます。

3. データ共有にオブジェクトを追加したり、データ共有からオブジェクトを削除したりします。データ共有にオブジェクトを追加するには、オブジェクトを追加する前にスキーマを追加し

ます。スキーマを追加する場合、Amazon Redshift はその下にすべてのオブジェクトを追加するわけではありません。これらは明示的に追加してください。詳細については、「ALTER DATASHARE」を参照してください。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

データ共有にビューを追加することもできます。

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

ALTER DATASHARE を使用して、特定のスキーマ内のスキーマ、テーブル、ビュー、および関数を共有します。スーパーユーザー、データ共有の所有者、またはデータ共有に対する ALTER または ALL 許可を持つユーザーは、データ共有を変更して、それに対するオブジェクトの追加または削除を実行できます。ユーザーは、データ共有に対してオブジェクトの追加または削除を行う許可を持っている必要があります。ユーザーは、オブジェクトの所有者である、またはオブジェクトに対する SELECT、USAGE、もしくは ALL 許可を持っている必要もあります。

また、GRANT を使用してデータ共有にオブジェクトを追加することもできます。次の例では、以下の方法を示します。

```
GRANT SELECT ON TABLE public.tickit_sales_redshift TO DATASHARE salesshare;
```

この構文は機能的には `ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;` と同等です。

スキーマを指定して作成された新しいテーブル、ビュー、または SQL ユーザー定義関数 (UDF) をデータ共有に追加するには、INCLUDENEW 句を使用します。データ共有とスキーマの各ペアについて、このプロパティを変更できるのはスーパーユーザーのみです。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

Amazon Redshift コンソールを使用して、データ共有でオブジェクトを追加または削除することもできます。詳細については、「データ共有へのデータ共有オブジェクトの追加」、「データ共

有からのデータ共有オブジェクトの削除」、および「アカウントで作成されたデータ共有の編集」を参照してください。

4. コンシューマーをデータ共有に追加するか、データ共有からコンシューマーを削除します。次の例では、コンシューマークラスター名前空間を salesshare に追加しています。この名前空間は、アカウント内にあるコンシューマークラスターの名前空間ための、グローバルにユニークな識別子 (GUID) です。詳細については、「GRANT」または「REVOKE」を参照してください。

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

GRANT ステートメントでは、1 つのデータ共有コンシューマーにしか許可を付与できません。

クラスターのスーパーユーザーおよびデータ共有オブジェクトの所有者、またはデータ共有に対する SHARE アクセス許可を持つユーザーは、コンシューマーをデータ共有に追加したり、データ共有からコンシューマーを削除したりできます。そのために、GRANT USAGE または REVOKE USAGE を使用します。

現在表示されているクラスターの名前空間を検索するには、SELECT CURRENT_NAMESPACE コマンドを使用できます。同じ AWS アカウント内の異なるクラスターの名前空間を検索するには、Amazon Redshift コンソールのクラスター詳細ページにアクセスします。そのページで、新しく追加された名前空間フィールドを見つけます。

また、Amazon Redshift コンソールを使用して、データ共有に対しデータコンシューマーを追加したり削除したりできます。詳細については、データ共有へのデータコンシューマーの追加およびデータ共有からのデータコンシューマーの削除を参照してください。

5. (オプション) データ共有にセキュリティ制限を追加します。次の例は、パブリック IP アクセスを持つコンシューマークラスターがデータ共有の読み取りを許可されていることを示しています。詳細については、「ALTER DATASHARE」を参照してください。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE = TRUE;
```

データ共有の作成後に、コンシューマーのタイプに関するプロパティを変更できます。例えば、特定のデータ共有からデータを使用するクラスターは、公開でアクセスできないように定義できます。データ共有で指定されたセキュリティ制限を満たさないコンシューマークラスターからのクエリは、クエリランタイムで拒否されます。

Amazon Redshift コンソールを使用して、データ共有を編集することもできます。詳細については、「アカウントで作成されたデータ共有の編集」を参照してください。

6. クラスターで作成されたデータ共有を一覧表示し、データ共有の内容を調べます。

   次の例は、salesshareという名前のデータ共有の情報を表示します。詳細については、DESC DATASHAREおよびSHOW DATASHARESを参照してください。

```
DESC DATASHARE salesshare;


 producer_account  |          producer_namespace            | share_type | share_name
 | object_type |           object_name            |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+--------------------------------+-------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

   次の例は、プロデューサークラスター内のアウトバウンドデータ共有を表示します。

```
SHOW DATASHARES LIKE 'sales%';
```

   出力は次の例のようになります。

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate      | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
```

```
-----------+-------------+----------------+------------------+------------
+-------------------+--------------------+----------+-----------------
+--------------------------------------
salesshare |    100      | dev            |                  |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |          |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

詳細については、「DESC DATASHARE」および「SHOW DATASHARES」を参照してください。

SVV_DATASHARES、SVV_DATASHARE_CONSUMERS、および SVV_DATASHARE_OBJECTS を使用して、データ共有、そのデータ共有内のオブジェクト、およびデータ共有のコンシューマーを表示することもできます。

7. データ共有を削除します。詳細については、「DROP DATASHARE」を参照してください。

   DROP DATASHARE を使用して、いつでもデータ共有オブジェクトを削除できます。クラスターのスーパーユーザーとデータ共有の所有者は、データ共有を削除できます。

   次の例では、salesshareという名前のデータ共有を削除します。

```
DROP DATASHARE salesshare;
```

   Amazon Redshift コンソールを使用してデータ共有を削除することもできます。詳細については、「アカウント内で作成された データ共有の削除」を参照してください。

8. ALTER DATASHARE コマンドを使用すると、データ共有から任意の時点でオブジェクトを削除できます。REVOKE USAGE ON を使用すると、特定のコンシューマーについて、データ共有への許可を取り消すことができます。このコマンドは、データ共有内にあるオブジェクトの USAGE 許可を取り消し、すべてのコンシューマークラスターへのアクセスを即座に停止します。アクセス許可が取り消されたあとは、データベースやテーブルに関するものを含め、データ共有およびメタデータを一覧表示するためのクエリは、共有されたオブジェクトを返さなくなります。

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

   Amazon Redshift コンソールを使用して、データ共有を編集することもできます。詳細については、「アカウントで作成されたデータ共有の編集」を参照してください。

9. コンシューマーとデータを共有する必要がなくなった場合は、名前空間からデータ共有へのアクセスを取り消します。

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

Amazon Redshift コンソールを使用して、データ共有を編集することもできます。詳細については、「アカウントで作成されたデータ共有の編集」を参照してください。

コンシューマークラスターの管理者として読み取り目的でデータを共有するには

1. 利用可能なデータ共有を一覧表示し、各データ共有のコンテンツを表示します。詳細については、「DESC DATASHARE」および「SHOW DATASHARES」を参照してください。

   次の例では、指定されたプロデューサー名前空間のインバウンドデータ共有の情報を表示します。DESC DATASHARE をコンシューマークラスターの管理者として実行する場合、インバウンドデータ共有を表示するには、NAMESPACE オプションを指定する必要があります。

```
DESC DATASHARE salesshare OF NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';


 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name            |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+----------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift      |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                           |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view   |
```

クラスターのスーパーユーザーのみがこれを行うことができます。SVV_DATASHARES を使用してデータ共有を表示し、SVV_DATASHARE_OBJECTS を使用してデータ共有内のオブジェクトを表示することもできます。

次の例では、コンシューマークラスター内のインバウンドデータ共有を表示します。

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+--------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |          t          |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

2. データベーススーパーユーザーとして、データ共有を参照するローカルデータベースを作成できます。詳細については、「CREATE DATABASE」を参照してください。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

ローカルデータベース内のオブジェクトへのアクセスをより細かく制御する場合は、データベースの作成時に WITH PERMISSIONS 句を使用します。これにより、ステップ 4 でデータベース内のオブジェクトにオブジェクトレベルのアクセス許可を付与できます。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

SVV_REDSHIFT_DATABASES ビューにクエリを実行すると、データ共有から作成したデータベースを確認できます。これらのデータベースに直接接続したり、コンシューマークラスターのローカルデータベースに接続し、データベース間クエリを実行して、データ共有データベースのデータをクエリしたりすることができます。既存のデータ共有から作成されたデータベースオブジェクトの上にデータ共有を作成することはできません。ただし、コンシューマークラスター上の別のテーブルにデータをコピーし、必要な処理を実行してから、作成された新しいオブジェクトを共有できます。

Amazon Redshift コンソールを使用して、データ共有からデータベースを作成することもできます。詳細については、「データ共有からのデータベースの作成」を参照してください。

3. (オプション) 外部スキーマを作成して、コンシューマークラスターにインポートされたコンシューマデータベース内の特定のスキーマを参照し、詳細なアクセス許可を割り当てます。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 必要に応じて、データ共有から作成されたデータベースおよびスキーマ参照に対するアクセス許可を、コンシューマークラスター内のユーザーおよびロールに付与します。詳細については、「GRANT」または「REVOKE」を参照してください。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS を使用することなくデータベースを作成している場合は、データ共有から作成されたデータベース全体に対するアクセス許可のみをユーザーおよびロールに割り当てることができます。場合によっては、データ共有から作成されたデータベースオブジェクトのサブセットに対して、詳細なコントロールが必要になります。その場合は、データ共有内の特定のスキーマを指す外部スキーマ参照を作成し (前出のステップを参照)、スキーマレベルの詳細なアクセス許可を提供できます。

また、共有オブジェクトの上に遅延バインディングビューを作成し、これらを使用して詳細なアクセス許可を割り当てることもできます。また、プロデューサークラスターで必要な粒度で追加のデータ共有を作成するよう検討することもできます。

ステップ 2 で WITH PERMISSIONS を使用してデータベースを作成した場合は、共有データベース内のオブジェクトにオブジェクトレベルのアクセス許可を割り当てる必要があります。USAGE アクセス許可のみを持つユーザーは、追加のオブジェクトレベルのアクセス許可が付与されるまで、WITH PERMISSIONS を使用して作成されたデータベース内のオブジェクトにはアクセスできません。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. データ共有内の共有オブジェクト内のデータをクエリします。

コンシューマークラスター上のコンシューマデータベースおよびスキーマに対するアクセス許可を持つユーザーおよびロールは、任意の共有オブジェクトのメタデータを探索およびナビゲートできます。また、コンシューマークラスター内のローカルオブジェクトを探索およびナビゲートすることもできます。これを行うには、JDBC または ODBC ドライバー、または SVV_ALL および SVV_REDSHIFT ビューを使用します。

プロデューサークラスターには、各スキーマ内のデータベース、テーブル、およびビューに多数のスキーマが含まれる場合があります。コンシューマー側のユーザーは、データ共有を通じて利用可能になったオブジェクトのサブセットのみを表示できます。これらのユーザーは、プロデューサークラスターからのメタデータ全体を表示することはできません。このアプローチは、データ共有による詳細なメタデータセキュリティの制御を提供します。

引き続きローカルクラスターのデータベースに接続します。ただし、3 つの部分からなる database.schema.table 表記を使用して、データ共有から作成されたデータベースとスキーマから読み込むこともできるようになりました。表示されているすべてのデータベースにまたがるクエリを実行できます。これらは、クラスター上のローカルデータベースでも、データ共有から作成されたデータベースでもかまいません。コンシューマークラスターは、データ共有から作成されたデータベースに接続できません。

完全な資格を使用してデータにアクセスできます。詳細については、「クロスデータベースクエリの例」を参照してください。

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
   109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
    11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
    52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
    26.25 | 2008-06-09 08:38:52
       5 |      6 |    47402 |   14115 |    8240 |   2069 |       2 |    154.00 |
    23.10 | 2008-08-31 09:17:02
```

SELECT ステートメントは、共有オブジェクトに対してのみ使用できます。ただし、別のローカルデータベース内の共有オブジェクトからデータをクエリすることで、コンシューマークラスターにテーブルを作成できます。

クエリに加えて、コンシューマーは共有オブジェクトのビューを作成できます。遅延バインディングビューまたはマテリアライズドビューのみがサポートされます。Amazon Redshift は、共有データの通常のビューをサポートしていません。コンシューマーが作成するビューは、複数のローカルデータベースまたはデータ共有から作成されたデータベースにまたがることができます。詳細については、「CREATE VIEW」を参照してください。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

データへの書き込みアクセスの共有 (プレビュー)

読み取りと書き込みの両方のデータベースオブジェクトは、同一 AWS アカウント 内の Amazon Redshift クラスターや Amazon Redshift Serverless ワークグループ間、アカウント間、およびリージョン間で共有できます。このトピックの手順は、書き込みアクセス許可を含むデータ共有を設定する方法を示しています。異なるテーブルごとに SELECT、INSERT、UPDATE などのアクセス許可を付与したり、スキーマごとに USAGE と CREATE などのアクセス許可を付与できます。データは、書き込みトランザクションがコミットされるとすぐにライブになり、すべてのウェアハウスで利用できるようになります。プロデューサーアカウント管理者は、特定の名前空間またはリージョンが、データに対する読み取り専用アクセス、読み取り/書き込みアクセス、または任意のアクセスを許可するかどうかを決定できます。

以下のセクションでは、データ共有の設定方法を説明します。この手順では、プロビジョニングされたクラスターまたは Amazon Redshift Serverless ワークグループ内のデータベースで作業を行っていることを前提としています。

読み取り専用データ共有と読み取り/書き込みデータ共有の相違

以前は、データ共有内のオブジェクトはどのような状況でも読み取り専用でした。データ共有内のオブジェクトへの書き込みは新機能です。データ共有内のオブジェクトは、プロデューサーがデータ共有へのオブジェクトに対する INSERT や CREATE などの書き込み権限を明示的に付与した場合にのみ、書き込み可能になります。さらに、アカウント間の共有の場合、プロデューサーは書き込み用のデータ共有を承認し、コンシューマーは書き込み用に特定のクラスターとワークグループを関連付ける必要があります。詳細については、このトピックの後のセクションで説明します。

データ共有に付与できるアクセス許可 (プレビュー)

データ共有コンテキストにおける、さまざまなオブジェクトタイプと、それらに付与できるさまざまなアクセス許可は、以下のとおりです。

スキーマ:

- USAGE
- CREATE

テーブル:

- SELECT
- INSERT
- UPDATE
- DELETE
- TRUNCATE
- DROP
- REFERENCES

関数:

- EXECUTE

データベース:

- CREATE

プレビューでのデータ共有の要件と制限

- 接続 x – データ共有データベースに書き込むには、データ共有データベースに直接接続するか、USE コマンドを実行する必要があります。ただし、間もなく 3 つの部分からなる表記法が可能になる予定です。
- 可用性 – この機能を使用するには、Serverless ワークグループ、ra3.4xl クラスター、または ra3.16xl クラスターを使用する必要があります。
- メタデータの検出 — Redshift JDBC、ODBC、または Python ドライバーを介してデータ共有データベースに直接接続しているコンシューマーの場合、以下の方法でカタログデータを表示できます。
  - SQL SHOW コマンド。
  - information_schema テーブルとビューのクエリ。
  - SVV メタデータビューのクエリ。
- Data API – Data API を介したデータ共有データベースへの接続はできません。これに対するサポートが間もなく提供される予定です。
- アクセス許可の表示 – コンシューマーはデータ共有に付与されたアクセス許可を表示できません。これは間もなく追加されます。
- 暗号化 – アカウント間のデータ共有の場合、プロデューサクラスターとコンシューマークラスターの両方を暗号化する必要があります。
- 分離レベル – 他のサーバーレスワークグループやクラスターでの書き込みを許可するには、データベースの分離レベルをスナップショット分離にする必要があります。
- 自動操作 – コンシューマーがデータ共有オブジェクトに書き込みを行っても、自動分析操作はトリガーされません。そのため、プロデューサーはテーブルにデータを挿入した後に、手動で分析を実行してテーブル統計を更新する必要があります。このようにしないと、クエリプランが最適にならない可能性があります。
- マルチステートメントのクエリとトランザクション — トランザクションブロック外のマルチステートメントのクエリは、現在サポートされていません。そのため、dbeaver のようなクエリエディターを使用していて、書き込みクエリが複数ある場合は、クエリを明示的な BEGIN... END トランザクションステートメントでラップする必要があります。

サポートされている SQL ステートメント

書き込み機能付きデータ共有のパブリックプレビューリリースでは、以下のステートメントがサポートされます。

- BEGIN | START TRANSACTION
- END | COMMIT | ROLLBACK
- COPY without COMPUPDATE
- { CREATE | DROP } SCHEMA
- { CREATE | DROP | SHOW } TABLE
- CREATE TABLE table_name AS
- DELETE
- { GRANT | REVOKE } privilege_name ON OBJECT_TYPE object_name TO consumer_user
- INSERT
- SELECT
- INSERT INTO SELECT
- TRUNCATE
- UPDATE
- スーパーデータタイプ列

サポートされていないステートメントタイプ — 以下はサポートされていません。

- プロデューサーに書き込む際のコンシューマーウェアハウスへのマルチステートメントクエリ。
- コンシューマーからプロデューサーへのクエリの書き込みについての同時実行スケーリング。
- コンシューマーからプロデューサーへのジョブの書き込みの自動コピー。
- コンシューマーからプロデューサーへのジョブの書き込みのストリーミング。
- コンシューマーによる、プロデューサークラスター上での ゼロ ETL 統合テーブルの作成。ゼロ ETL 統合の詳細については、「ゼロ ETL 統合の開始方法」を参照してください。
- インタリーブソートキーを用いたテーブルへの書き込み。

プロデューサーアカウント管理者として書き込みアクセス許可を持つアカウント内のデータ共有 (プレビュー)

以前は、データ共有内のオブジェクトはどのような状況でも読み取り専用でした。データ共有内のオブジェクトへの書き込みは新機能です。データ共有内のオブジェクトは、プロデューサーがデータ共有へのオブジェクトに対する INSERT や CREATE などの書き込み権限を明示的に付与した場合にのみ、書き込み可能になります。詳細については、このトピックの後のセクションで説明します。

読み取り専用のデータ共有に関する既存のドキュメントをお探しの場合は、「Amazon Redshift のクラスター間でのデータ共有」を参照してください。

データ共有を開始するには、プロデューサーの管理者がデータ共有を作成してオブジェクトを追加します。

1. プロデューサーデータベース所有者またはスーパーユーザーがデータ共有を作成します。データ共有は、データベースオブジェクト、アクセス許可、コンシューマーの論理的なコンテナです。(コンシューマーは、アカウントや他のアカウント内のクラスターまたは Amazon Redshift Serverless 名前空間です。) 各データ共有は、そのデータ共有が作成されたデータベースに関連付けられており、追加できるのはそのデータベースのオブジェクトだけです。次のコマンドでデータ共有が作成されます。

```
CREATE DATASHARE my_datashare [PUBLICACCESSIBLE = TRUE];
```

   PUBLICACCESSIBLE = TRUE に設定すると、コンシューマーは、パブリックにアクセス可能なクラスターやプロビジョニングされたワークグループからデータ共有をクエリできます。許可しない場合は、これを除外するか、明示的に false に設定してください。

   データ共有の所有者は、データ共有に追加するスキームに USAGE を付与する必要があります。GRANT コマンドが新しく追加されました。CREATE や USAGE など、スキーマに対するさまざまなアクションの許可を付与するために使用されます。スキーマには、以下の共有オブジェクトが含まれます。

```
CREATE SCHEMA myshared_schema1;
CREATE SCHEMA myshared_schema2;

GRANT USAGE ON SCHEMA myshared_schema1 TO DATASHARE my_datashare;
GRANT CREATE, USAGE ON SCHEMA myshared_schema2 TO DATASHARE my_datashare;
```

   または、管理者は引き続き ALTER コマンドを実行して、データ共有にスキーマを追加できます。この方法でスキーマを追加した場合、USAGE アクセス許可のみが付与されます。

```
ALTER DATASHARE my_datashare ADD SCHEMA myshared_schema1;
```

2. 管理者がスキーマを追加すると、スキーマ内のオブジェクトに対するデータ共有のアクセス許可を付与できます。読み取りと書き込みの両方のアクセス許可が可能です。GRANT ALL サンプルは、すべてのアクセス許可を付与する方法を示しています。

```
GRANT SELECT, INSERT ON TABLE myshared_schema1.table1, myshared_schema1.table2,
 myshared_schema2.table1
TO DATASHARE my_datashare;

GRANT ALL ON TABLE myshared_schema1.table4 TO DATASHARE my_datashare;
```

ALTER DATASHARE などのコマンドを引き続き実行してテーブルを追加できます。実行すると、追加されたオブジェクトには SELECT アクセス許可のみが付与されます。

```
ALTER DATASHARE my_datashare ADD TABLE myshared_schema1.table1,
 myshared_schema1.table2, myshared_schema2.table1;
```

3. 管理者は、データ共有の使用権限をアカウント内の特定の名前空間に付与します。ARN の一部としての名前空間 ID を確認するには、クラスターの詳細ページ、Amazon Redshift Serverless 名前空間の詳細ページを参照するか、コマンド `SELECT current_namespace;` を実行します。詳細については、「CURRENT_NAMESPACE」を参照してください。

```
GRANT USAGE ON DATASHARE my_datashare TO NAMESPACE '86b5169f-012a-234b-9fbb-
e2e24359e9a8';
```

アカウント間でのデータへの書き込みアクセス許可の共有 (プレビュー)

Amazon Redshift では、AWS アカウント間でデータを共有し、書き込み許可を付与して、チームまたは組織間のコラボレーションとデータ共有を行うことができます。クロスアカウントデータ共有を使用すると、データベース、スキーマ、テーブルを作成および管理するデータプロバイダーアカウントを作成し、データコンシューマーアカウントと安全に共有できます。以下のセクションでは、クロスアカウントデータ共有を設定し、Amazon Redshift で書き込み許可を付与するプロセスについて説明します。

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

PREVIEW_2023 トラックでまだデータ共有を作成していない場合、開始するには、「データへの書き込みアクセスの共有 (プレビュー)」に移動します。

共有データをコンシューマーデータセキュリティ管理者として関連付ける (プレビュー)

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

PREVIEW_2023 トラックでまだデータ共有を作成していない場合、開始するには、「データへの書き込みアクセスの共有 (プレビュー)」に移動します。

前提条件: このセクションの手順は、プロデューサー管理者が共有データベースオブジェクトに対して特定のアクションを許可し、データ共有が別のアカウントと共有されている場合は、プロデューサーのセキュリティ管理者がアクセスを承認した後に実行されます。

コンシューマーセキュリティ管理者は以下を決定します。

- アカウント内のすべての名前空間、アカウント内の特定の地域の名前空間、または特定の名前空間がデータ共有にアクセスできるかどうか。
- 名前空間にデータ共有へのアクセス権がある場合、それらの名前空間に書き込みのアクセス許可があるかどうか。

コンシューマーセキュリティ管理者は、コンソール、CLI、または API を介してデータ共有を関連付けることができます。CLI の場合、管理者は以下のコマンドを使用します。

```
associate-data-share-consumer
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

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

コンシューマーセキュリティ管理者は、データ共有を名前空間に関連付けるときに明示的に `allow-writes` を true に設定して、INSERT コマンドと UPDATE コマンドを使用可能にする必要がありま

す。設定しない場合、ユーザーは SELECT、USAGE、EXECUTE などの読み取り操作しか実行できません。

データ共有の名前空間の関連付けを変更するには、別の値を指定して再度 `associate-data-share-consumer` を呼び出します。以前の関連付けは新しい関連付けで上書きされるため、最初に `allow-writes` の関連付けと設定したものの、`no-allow-writes` の関連付けと指定を行っていた場合や、単に値を指定しなかった場合は、コンシューマーの書き込みのアクセス許可は取り消されます。

プロデューサーセキュリティ管理者としてデータ共有への書き込みを承認する (プレビュー)

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

PREVIEW_2023 トラックでまだデータ共有を作成していない場合、開始するには、「データへの書き込みアクセスの共有 (プレビュー)」に移動します。

> **Note**
>
> これは、データ共有がアカウント間で共有されている場合にのみ適用されます。

プロデューサーセキュリティ管理者は以下を決定します。

- 別のアカウントがデータ共有へのアクセス権を持つことができるかどうか。
- アカウントにデータ共有へのアクセス権がある場合に、そのアカウントに書き込みのアクセス許可があるかどうか。

データ共有を承認するには、以下の IAM アクセス許可が必要です。

redshift:AuthorizeDataShare

CLI 呼び出しまたは API を使用して、使用と書き込みを承認できます。

```
authorize-data-share
```

```
--data-share-arn <value>
--consumer-identifier <value>
[--allow-writes | --no-allow-writes]
```

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

コンシューマー ID は次のいずれかになります。

- 12 桁の AWS アカウント ID。
- 名前空間識別子 ARN。

書き込みのアクセス許可は、承認段階では付与されないことに注意してください。書き込み用のデータ共有の承認によってアカウントに付与されるのは、データ共有管理者によって付与された書き込みアクセス許可のみです。管理者が書き込みを許可しない場合、特定のユーザーが使用できるアクセス許可は SELECT、USAGE、EXECUTE に限られます。

データ共有コンシューマーの承認を変更するには、値を変えて再度 `authorize-data-share` を呼び出します。以前の承認は、新しい承認によって上書きされます。そのため、書き込みに対する承認と許可を当初行った場合でも、その後 `no-allow-writes` の再承認と指定をしなかったり、単に値を指定しなかったりすると、コンシューマーの書き込みのアクセス許可は取り消されます。

データ共有が利用可能なリージョン (プレビュー)

Amazon Redshift では、同じリージョンまたは異なる AWS リージョンの Amazon Redshift クラスター間でデータを共有できるため、リージョン間の分析、データレジデンシー、ディザスタリカバリシナリオが可能になります。データ共有が利用可能なリージョンとは、データ共有からのデータ共有をサポートする AWS リージョンを指します。これは、アカウント間またはクラスター間で共有できる 1 つ以上のデータベースオブジェクトを含む Amazon Redshift オブジェクトです。以下のセクションでは、リージョン間における Amazon Redshift のデータ共有の設定と管理について説明します。

これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、データ共有が利用可能な、同じまたは異なる AWS リージョンの Amazon Redshift クラスター間でライブデータを共有できます。

PREVIEW_2023 トラックでまだデータ共有を作成していない場合、開始するには、「データへの書き込みアクセスの共有 (プレビュー)」に移動します。

次のリージョンでは、プレビューでのデータ共有が利用できます。

- 米国東部 (バージニア北部) (us-east-1)
- 米国東部 (オハイオ) (us-east-2)
- 米国西部 (オレゴン) (us-west-2)
- アジアパシフィック (東京) (ap-northeast-1)
- 欧州 (アイルランド) (eu-west-1)
- 欧州 (ストックホルム) (eu-north-1)

AWS アカウント間のデータの共有

AWS アカウント間で、読み取りのためにデータを共有することができます。AWS アカウント間でのデータの共有は、アカウント内でのデータの共有と同じような仕組みになっていますが、AWS アカウント間でのデータの共有では双方向ハンドシェイクが必要となる点が異なります。プロデューサアカウント管理者は、コンシューマーアカウントにデータ共有へのアクセスを許可するか、アクセスを許可しないかのいずれかを選択できます。承認されたデータ共有を使用するために、コンシューマーアカウントの管理者はデータ共有を関連付けることができます。この管理者は、データ共有を AWS アカウント全体またはコンシューマーアカウント内の特定のクラスターに関連付けることや、データ共有を拒否することができます。アカウント内でのデータ共有の詳細については、AWS アカウント 内のデータへの読み取りアクセスの共有を参照してください。

データ共有には、同じアカウント内のクラスター名前空間、または異なる AWS アカウントであるデータコンシューマーを含めることができます。アカウント内での共有やクロスアカウントの共有用に別々のデータ共有を作成する必要はありません。

クロスアカウントデータ共有の場合、プロデューサクラスターとコンシューマークラスターの両方を暗号化する必要があります。

AWS アカウントとデータを共有する場合、プロデューサークラスターの管理者は、エンティティとして AWS アカウントと共有します。コンシューマークラスターの管理者は、コンシューマアカウントのどのクラスター名前空間がデータ共有にアクセスできるかを決定できます。

トピック

プロデューサークラスター管理者のアクション

Amazon Redshift では、プロデューサークラスターで管理タスクを実行して、データの取り込みとロード処理を管理できます。

プロデューサークラスターの管理者またはデータベース所有者である場合 – 以下のステップに従います。

1. クラスターにデータ共有を作成し、データ共有オブジェクトをデータ共有に追加します。データ共有を作成し、データ共有オブジェクトをデータ共有に追加する方法についての詳細なステップについては、「AWS アカウント 内のデータへの読み取りアクセスの共有」を参照してください。CREATE DATASHARE および ALTER DATASHARE については、「CREATE DATASHARE」および「ALTER DATASHARE」を参照してください。

   次の例では、異なるデータ共有オブジェクトをデータ共有 salesshare に追加しています。

```
-- Add schema to datashare
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;

-- Add table under schema to datashare
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;

-- Add view to datashare
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;

-- Add all existing tables and views under schema to datashare (does not include
 future table)
ALTER DATASHARE salesshare ADD ALL TABLES in schema public;
```

   Amazon Redshift コンソールを使用して、データ共有を作成または編集することもできます。詳細については、データ共有の作成およびアカウントで作成されたデータ共有の編集を参照してください。

2. データ共有を操作するための権限を委任します。詳細については、「GRANT」または「REVOKE」を参照してください。

次の例では、salesshare上の dbuser にアクセス許可を付与しています。

```
GRANT ALTER, SHARE ON DATASHARE salesshare TO dbuser;
```

クラスターのスーパーユーザーとデータ共有の所有者は、追加のユーザーに対してデータ共有の変更許可の付与または取り消しを実行できます。

3. コンシューマーをデータ共有に追加するか、データ共有からコンシューマーを削除します。次の例は、AWS アカウントID を salesshare に追加します。詳細については、「GRANT」または「REVOKE」を参照してください。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012';
```

GRANT ステートメントでは、1 つのデータコンシューマーにしか許可を付与できません。

クラスターのスーパーユーザーおよびデータ共有オブジェクトの所有者、またはデータ共有に対する SHARE 許可を持つユーザーは、コンシューマーをデータ共有に追加したり、データ共有からコンシューマーを削除したりできます。そのために、GRANT USAGE または REVOKE USAGE を使用します。

また、Amazon Redshift コンソールを使用して、データ共有に対しデータコンシューマーを追加したり削除したりできます。詳細については、データ共有へのデータコンシューマーの追加およびデータ共有からのデータコンシューマーの削除を参照してください。

4. (オプション) コンシューマーとデータを共有する必要がなくなった場合は、AWS アカウントからデータ共有へのアクセス権を取り消します。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012';
```

プロデューサーアカウントの管理者である場合 – 以下のステップに従います。

AWS アカウントに使用を許可すると、データ共有のステータスが pending_authorization になります。プロデューサーアカウントの管理者は、Amazon Redshift コンソールを使用してデータ共有を認可し、データコンシューマーを選択する必要があります。

https://console.aws.amazon.com/redshiftv2/ にサインインします。次に、データ共有へのアクセスを許可する、またはその許可を削除するデータコンシューマーを選択します。認可されたデータコンシューマーは、データ共有でアクションを実行するための通知を受け取ります。クラスター名前空

間をデータコンシューマーとして追加する場合は、承認を実行する必要はありません。データコンシューマーが認可されると、データ共有オブジェクトにアクセスし、データをクエリするコンシューマデータベースを作成できます。詳細については、「データ共有の承認と承認の取り消し」を参照してください。

コンシューマーアカウント管理者のアクション

Amazon Redshift では、コンシューマーアカウントの管理、およびデータウェアハウスリソースへのアクセス制御を行えます。

コンシューマーアカウントの管理者である場合 – 以下のステップに従います。

他のアカウントから共有されている 1 つまたは複数のデータ共有を、AWS アカウント全体、またはアカウント内の特定のクラスター名前空間に関連付けるには、Amazon Redshift コンソールを使用します。

https://console.aws.amazon.com/redshiftv2/ にサインインします。次に、他のアカウントから共有されている 1 つまたは複数のデータ共有を、AWS アカウント全体、またはアカウント内の特定のクラスター名前空間と関連付けます。詳細については、「データ共有の関連付け」を参照してください。

AWS アカウント、または特定のクラスター名前空間への関連付けが完了すると、データ共有が利用可能になります。データ共有の関連付けはいつでも変更できます。関連付けを個々のクラスター名前空間から AWS アカウントに変更すると、Amazon Redshift がクラスター名前空間を AWS アカウント情報で上書きします。関連付けを AWS アカウントから特定のクラスター名前空間に変更すると、Amazon Redshift が AWS アカウント情報をクラスター名前空間情報で上書きします。アカウント内のすべてのクラスター名前空間が、データにアクセスします。

コンシューマークラスター管理者のアクション

Amazon Redshift では、コンシューマークラスターで管理タスクを実行して、データの取り込みとロード処理を管理できます。

コンシューマークラスターの管理者である場合 – 以下のステップに従います。

1. 利用可能なデータ共有を一覧表示し、データ共有のコンテンツを表示します。データ共有のコンテンツは、プロデューサークラスターの管理者がデータ共有を認可し、コンシューマークラスターの管理者がデータ共有を受け入れて関連付けた場合にのみ使用できます。詳細については、DESC DATASHAREおよびSHOW DATASHARESを参照してください。

   次の例では、指定されたプロデューサー名前空間のインバウンドデータ共有の情報を表示します。コンシューマークラスターの管理者として DESC DATAHSARE を実行する場合、インバウ

ンドのデータ共有を表示するためには、NAMESPACE とアカウント ID を指定する必要があります。アウトバウンドのデータ共有の場合は、データ共有名を指定します。

```
SHOW DATASHARES LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+---------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012    | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
DESC DATASHARE salesshare OF ACCOUNT '123456789012' NAMESPACE 'dd8772e1-
d792-4fa4-996b-1870577efc0d';


 producer_account |          producer_namespace          | share_type | share_name |
 object_type |           object_name
------------------+--------------------------------------+------------+------------
+-------------+---------------------------------
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_users_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_venue_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_category_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_date_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_event_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_listing_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d | INBOUND    | salesshare |
 schema      | public
(8 rows)
```

クラスターのスーパーユーザーのみがこれを行うことができます。SVV_DATASHARES を使用してデータ共有を表示し、SVV_DATASHARE_OBJECTS を使用してデータ共有内のオブジェクトを表示することもできます。

次の例では、コンシューマークラスター内のインバウンドデータ共有を表示します。

```
SELECT * FROM SVV_DATASHARES WHERE share_name LIKE 'sales%';

share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-----------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+---------------------------------------
salesshare |             |                 |                   | INBOUND    |
            |        t            |           | 123456789012      | 'dd8772e1-
d792-4fa4-996b-1870577efc0d'
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name LIKE 'sales%';
 share_type | share_name | object_type |           object_name           |
 producer_account |          producer_namespace
------------+------------+-------------+---------------------------------
+------------------+--------------------------------------
 INBOUND    | salesshare | table       | public.tickit_users_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_venue_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_category_redshift |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_date_redshift     |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_event_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_listing_redshift  |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | table       | public.tickit_sales_redshift    |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
 INBOUND    | salesshare | schema      | public                          |
 123456789012     | dd8772e1-d792-4fa4-996b-1870577efc0d
(8 rows)
```

2. データ共有を参照するローカルデータベースを作成します。データ共有からデータベースを作成する際に、NAMESPACE とアカウント ID を指定します。詳細については、「CREATE DATABASE」を参照してください。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

ローカルデータベース内のオブジェクトへのアクセスをより細かく制御する場合は、データベースの作成時に WITH PERMISSIONS 句を使用します。これにより、ステップ 4 でデータベース内のオブジェクトにオブジェクトレベルのアクセス許可を付与できます。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE 'dd8772e1-d792-4fa4-996b-1870577efc0d';
```

SVV_REDSHIFT_DATABASES ビューにクエリを実行すると、データ共有から作成したデータベースを確認できます。これらのデータベースに直接接続したり、コンシューマークラスターのローカルデータベースに接続し、データベース間クエリを実行して、データ共有データベースのデータをクエリしたりすることができます。既存のデータ共有から作成されたデータベースオブジェクトの上にデータ共有を作成することはできません。ただし、コンシューマークラスター上の個別のテーブルにデータをコピーし、必要な処理を実行してから、作成された新しいオブジェクトを共有することは可能です。

3. (オプション) 外部スキーマを作成して、コンシューマークラスターにインポートされたコンシューマデータベース内の特定のスキーマを参照し、詳細なアクセス許可を割り当てます。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

4. 必要に応じて、データ共有から作成されたデータベースおよびスキーマ参照に対するアクセス許可を、コンシューマークラスター内のユーザーまたはロールに付与します。詳細については、「GRANT」または「REVOKE」を参照してください。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS を使用することなくデータベースを作成している場合は、データ共有から作成されたデータベース全体に対するアクセス許可のみをユーザーおよびロールに割り当てることができます。場合によっては、データ共有から作成されたデータベースオブジェクトのサブセットに対して、詳細なコントロールが必要になります。その場合は、前のステップでの説明にあるように、データ共有内の特定のスキーマを指す外部スキーマ参照を作成できます。その後、スキーマレベルで詳細なアクセス許可を指定します。また、共有オブジェクトの上に遅延バインディングビューを作成し、これらを使用して詳細なアクセス許可を割り当てることもできます。また、プロデューサークラスターで必要な粒度で追加のデータ共有を作成するよう検討することもできます。データ共有から作成されたデータベースへのスキーマ参照は、必要な数を作成できます。

ステップ 2 で WITH PERMISSIONS を使用してデータベースを作成した場合は、共有データベース内のオブジェクトにオブジェクトレベルのアクセス許可を割り当てる必要があります。USAGE アクセス許可のみを持つユーザーは、追加のオブジェクトレベルのアクセス許可が付与されるまで、WITH PERMISSIONS を使用して作成されたデータベース内のオブジェクトにはアクセスできません。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

5. データ共有内の共有オブジェクト内のデータをクエリします。

   コンシューマークラスター上のコンシューマデータベースおよびスキーマに対するアクセス許可を持つユーザーおよびロールは、任意の共有オブジェクトのメタデータを探索およびナビゲートできます。また、コンシューマークラスター内のローカルオブジェクトを探索およびナビゲートすることもできます。これを行うには、JDBC または ODBC ドライバー、または SVV_ALL および SVV_REDSHIFT ビューを使用します。

   プロデューサークラスターには、各スキーマ内のデータベース、テーブル、およびビューに多数のスキーマが含まれる場合があります。コンシューマー側のユーザーは、データ共有を通じて利用可能になったオブジェクトのサブセットのみを表示できます。これらのユーザーは、プロデューサークラスターからのメタデータ全体を表示することはできません。このアプローチは、データ共有による詳細なメタデータセキュリティの制御を提供します。

   引き続きローカルクラスターのデータベースに接続します。ただし、3 つの部分からなる database.schema.table 表記を使用して、データ共有から作成されたデータベースとスキーマから読み込むこともできるようになりました。表示されているすべてのデータベースにまたがるクエリを実行できます。これらは、クラスター上のローカルデータベースでも、データ共有から作成

されたデータベースでもかまいません。コンシューマークラスターは、データ共有から作成されたデータベースに接続できません。

完全な資格を使用してデータにアクセスできます。詳細については、「クロスデータベースクエリの例」を参照してください。

```
SELECT * FROM sales_db.public.tickit_sales_redshift;
```

SELECT ステートメントは、共有オブジェクトに対してのみ使用できます。ただし、別のローカルデータベース内の共有オブジェクトからデータをクエリすることで、コンシューマークラスターにテーブルを作成できます。

クエリの実行に加えて、コンシューマーは共有オブジェクトのビューを作成することができます。遅延バインディングビューおよびマテリアライズドビューのみがサポートされます。Amazon Redshift は、共有データの通常のビューをサポートしていません。コンシューマーが作成するビューは、複数のローカルデータベースまたはデータ共有から作成されたデータベースにまたがることができます。詳細については、「CREATE VIEW」を参照してください。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

AWS リージョン間のデータの共有

AWS リージョン 内の Amazon Redshift クラスター間で、読み取りのためにデータを共有することができます。クロスリージョンのデータ共有を使用すると AWS リージョン 間でデータを共有できます。データを手動でコピーする必要はありません。データを Amazon S3 にアンロードし、新しい Amazon Redshift クラスターにデータをコピーしたり、クロスリージョンでスナップショットのコピーを実行したりする必要はありません。

クロスリージョンのデータ共有を使用すると、クラスターが異なるリージョンにある場合でも、同じ AWS アカウント内、または異なる AWS アカウント内のクラスター全体でデータを共有できます。同じ AWS アカウント 内の異なる AWS リージョン にある Amazon Redshift クラスターでデータを

共有する際も、従うワークフローは AWS アカウント 内でデータを共有する場合と同じです。詳細については、「AWS アカウント 内のデータへの読み取りアクセスの共有」を参照してください。

データを共有するクラスターが、異なる AWS アカウント や AWS リージョン にある場合には、AWS アカウントの間でデータを共有する場合と同様のワークフローに従います。加えて、コンシューマークラスターにリージョンレベルの関連付けを含めます。クロスリージョンのデータ共有では、AWS アカウントおよび AWS リージョン 全体、そして AWS リージョン 内の特定のクラスター名前空間とのデータ共有の関連付けをサポートしています。AWS アカウント 間でのデータ共有の詳細については、「AWS アカウント間のデータの共有」を参照してください。

別のリージョンのデータを使用する場合、コンシューマーはプロデューサーリージョンからコンシューマーリージョンへのクロスリージョンデータ転送料金を支払います。

データ共有を使用するコンシューマーアカウントの管理者は、下記の 3 つの方法のいずれかでデータ共有を関連付けます。

- AWS リージョン のすべてにまたがった、AWS アカウント全体に対する関連付け
- AWS アカウント 内の特定の AWS リージョン に対する関連付け
- AWS リージョン 内の特定のクラスター名前空間との関連付け

管理者が AWS アカウント 全体でのデータ共有を選択した場合は、アカウント内の異なる AWS リージョン にわたり既存の、あるいは将来作成されるクラスターの名前空間は、そのデータ共有にアクセスできるようになります。コンシューマーアカウントの管理者は、リージョン内の特定の AWS リージョン またはクラスター名前空間を選択し、データストアへのアクセス権を付与することも可能です。

プロデューサークラスターの管理者またはデータベース所有者である場合は、データシェアを作成した上でデータベースオブジェクトとデータコンシューマーを追加し、そのデータコンシューマーにアクセス許可を付与します。詳細については、「プロデューサークラスター管理者のアクション」を参照してください。

プロデューサーアカウントの管理者の場合は、AWS Command Line Interface(AWS CLI) もしくは Amazon Redshift コンソールを使用してデータ共有を認可し、データコンシューマーを選択します。

コンシューマーアカウントの管理者である場合 – 以下のステップに従います。

他のアカウントから共有されている 1 つまたは複数のデータ共有を、AWS アカウント全体、特定の AWS リージョン、または AWS リージョン内のクラスター名前空間に関連付けるには、Amazon Redshift コンソールを使用します。

クロスリージョンのデータ共有では、AWS Command Line Interface (AWS CLI) または Amazon Redshift コンソールを使用して、クラスターを特定の AWS リージョンに追加できます。

1 つ以上の AWS リージョンを指定する場合は、`consumer-region`オプションを指定しながら CLI コマンドの `associate-data-share-consumer` を使用します。

次の CLI による例では、`associate-entire-account` オプションを使用して、AWS アカウント全体に `Salesshare` を関連付けています。一度に関連付けることができるリージョンは 1 つだけです。

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--associate-entire-account
```

次の例では、`Salesshare`を米国東部 (オハイオ) リージョン (`us-east-2`) に関連付けています。

```
aws redshift associate-data-share-consumer
--region {PRODUCER_REGION}
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:0123456789012:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-region 'us-east-2'
```

次の例では、アジアパシフィック (シドニー) リージョン (`ap-southeast-2`) にある、別の AWS アカウントの特定のコンシューマークラスター名前空間内に、`Salesshare` を関連付けています。

```
aws redshift associate-data-share-consumer
--data-share-arn arn:aws:redshift:{PRODUCER_REGION}:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/Salesshare
--consumer-arn 'arn:aws:redshift:ap-southeast-2:{CONSUMER_ACCOUNT}:namespace:
{ConsumerImmutableClusterId}'
```

Amazon Redshift コンソールを使用すると、AWS アカウント全体または特定の AWS リージョン、あるいは AWS リージョン 内のクラスター名前空間に対し、データ共有を関連付けることができます。これを行うには、「https://console.aws.amazon.com/redshiftv2/」にサインインします。その後、他のアカウントから共有されている 1 つ以上のデータ共有を、AWS アカウントや AWS リージョン 全体、あるいは AWS リージョン 内の特定のクラスター名前空間に関連付けます。詳細については、「データ共有の関連付け」を参照してください。

AWS アカウント、または特定のクラスター名前空間への関連付けが完了すると、データ共有が利用可能になります。データ共有の関連付けはいつでも変更できます。関連付けを個々のクラスター名前空間から AWS アカウントに変更すると、Amazon Redshift がクラスター名前空間を AWS アカウント情報で上書きします。関連付けを AWS アカウントから特定のクラスター名前空間に変更すると、Amazon Redshift が AWS アカウント情報をクラスター名前空間情報で上書きします。関連付け先を AWS アカウント 全体から特定の AWS リージョンやクラスター名前空間に変更すると、Amazon Redshift により AWS アカウント 情報が、特定のリージョンもしくはクラスター名前空間の情報で上書きされます。

コンシューマークラスター管理者の場合は、データ共有を参照するローカルデータベースを作成できます。また、データ共有から作成したデータベースに対するアクセス許可を、必要に応じてコンシューマークラスター内のユーザーまたはロールに付与できます。さらに、共有オブジェクトに関するビューや参照するための外部スキーマを作成できます。また、コンシューマークラスターにインポートされたコンシューマデータベース内の特定のスキーマに対しては、詳細なアクセス許可を割り当てることができます。詳細については、「コンシューマークラスター管理者のアクション」を参照してください。

クロスリージョンでのデータ共有でコスト管理を行う

Amazon Redshift では、リージョン間で転送されるデータの量を制限するようにデータ共有を設定することで、AWS リージョン間のデータ共有のコスト制御を管理できます。リージョン間のデータ共有のコストコントロールを管理することで、データ転送制限の設定、データ転送使用状況のモニタリング、および制限に近づいたときまたは超えたときの通知の受信を行うことができます。

別のリージョンのデータを使用する場合、コンシューマーはプロデューサーリージョンからコンシューマーリージョンへのクロスリージョンデータ転送料金を支払います。データ転送の料金は、リージョンによって異なります。この料金は、正常に実行されたクエリごとに、スキャンされたデータのバイト数に基づき決定します。Amazon Redshift 料金の詳細については、Amazon Redshift の料金を参照してください。

このバイト数は、次のメガバイトに切り上げられて課金され、各クエリにつき 10MB の最小数が適用されます。クエリの使用量に関するコストコントロールを設定すると、クラスター上で転送されるデータ量をクエリごとに表示できます。

クロスリージョンでのデータ共有の使用状況と、それに関連するコストをモニタリングし制御するには、日単位、週単位、月単位の使用制限を作成します。使用量がそれらの制限に達した場合に Amazon Redshift が自動的に実行するアクションを定義することで、予算の予測性を維持できます。Amazon Redshift の使用制限の詳細については、「Amazon Redshift での使用制限の管理」を参照してください。

Amazon Redshift が実行するアクションは、設定した使用制限に応じて、システムテーブル対するイベントのログ記録、Amazon SNS を使用しての管理者に対する CloudWatch アラームと通知の送信、そして、他の用途のためのクロスリージョンデータ共有の無効化が行えます。アクションの詳細については、「Amazon Redshift での使用制限の管理」を参照してください。

Amazon Redshift コンソールで使用制限を作成するには、ご使用のクラスターの [Actions] (アクション) から [Configure usage limit] (使用制限の設定) を選択します。[Cluster performance] (クラスターのパフォーマンス) タブ、または [Monitoring] (モニタリング) タブから CloudWatch メトリクスを自動生成すると、使用状況の傾向をモニタリングしたり、定義された制限を使用状況が超える場合にアラートを送信させたりできます。また、AWS CLIまたは Amazon Redshift API オペレーションにより、プログラム的に使用制限の作成、変更、および削除が実行できます。詳細については、「Amazon Redshift での使用制限の管理」を参照してください。

AWS Data Exchange でのライセンス付き Amazon Redshift データの共有

作成した AWS Data Exchange データ共有を AWS Data Exchange 製品に追加する際、プロバイダーは Amazon Redshift のデータにライセンスを付与できます。これにより、AWS Data Exchangeへの有効なサブスクリプションを持つコンシューマーが、Amazon Redshift 内の最新データの検出や、サブスクライブ、およびクエリを行えるようになります。

AWS Data Exchange 製品に追加された AWS Data Exchange データ共有を使用すると、コンシューマーはサブスクリプションの開始時に、自動的に製品のデータ共有にアクセスできるようになり、サブスクリプションが有効である限り、そのアクセス権を保持します。

トピック

- プロデューサーとしての AWS Data Exchange データ共有の使用
- コンシューマーとしての AWS Data Exchange データ共有の使用

プロデューサーとしての AWS Data Exchange データ共有の使用

Amazon Redshift では、プロデューサーは、データ共有を作成および管理することで、ライブデータ製品を AWS Data Exchange と共有できます。

プロデューサクラスターの管理者が、Amazon Redshift コンソールで AWS Data Exchange データ共有を管理する場合は、以下のステップに従います。

1. AWS Data Exchange 上で同じデータを参照するためのデータ共有をクラスター内で作成し、AWS Data Exchangeに対するアクセス権を、作成したデータ共有に付与します。

クラスターのスーパーユーザーとデータベースの所有者は、データ共有を作成できます。各データ共有は、作成時にデータベースに関連付けられます。そのデータベースのオブジェクトのみがそのデータ共有で共有できます。同じデータベース上に、同じ粒度または異なる粒度のオブジェクトを使用して、複数のデータ共有を作成できます。クラスター上に作成できるデータ共有の数に制限はありません。

Amazon Redshift コンソールを使用してデータ共有を作成することもできます。詳細については、「[データ共有の作成](#)」を参照してください。

CREATE DATASHARE ステートメントを実行する際、MANAGEDBY ADX オプションを使用して、データ共有へのアクセスを AWS Data Exchange に暗黙的に付与します。これにより、対象のデータ共有を AWS Data Exchange が管理することを指示します。MANAGEDBY ADX オプションは、新しいデータ共有を作成する場合にのみ使用できます。ALTER DATASHARE ステートメントを使用して、既存のデータ共有を変更しながら MANAGEDBY ADX オプションを追加することはできません。MANAGEDBY ADX オプションを使用して作成したデータ共有に対しては、AWS Data Exchangeのみがアクセスおよび管理できます。

```
CREATE DATASHARE salesshare
[[SET] MANAGEDBY [=] {ADX} ];
```

2. データ共有にオブジェクトを追加します。プロデューサー管理者は、AWS Data Exchangeデータ共有内で使用可能なデータ共有オブジェクトの管理を継続します。

   データ共有にオブジェクトを追加するには、オブジェクトを追加する前にスキーマを追加します。スキーマを追加する場合、Amazon Redshift はその下にすべてのオブジェクトを追加するわけではありません。それらを明示的に追加する必要があります。詳細については、「[ALTER DATASHARE](#)」を参照してください。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

   データ共有にビューを追加することもできます。

```
CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
 public.tickit_sales_redshift;
ALTER DATASHARE salesshare ADD TABLE public.sales_data_summary_view;
```

ALTER DATASHARE を使用して、特定のスキーマ内のスキーマ、テーブル、ビュー、および関数を共有します。スーパーユーザー、データ共有の所有者、またはデータ共有に対する ALTER または ALL 許可を持つユーザーは、データ共有を変更して、それに対するオブジェクトの追加または削除を実行できます。ユーザーは、データ共有に対してオブジェクトの追加または削除を行う許可を持っている必要があります。ユーザーは、オブジェクトの所有者である、またはオブジェクトに対する SELECT、USAGE、もしくは ALL 許可を持っている必要もあります。

スキーマを指定して作成された新しいテーブル、ビュー、または SQL ユーザー定義関数 (UDF) をデータ共有に追加するには、INCLUDENEW 句を使用します。データ共有とスキーマの各ペアについて、このプロパティを変更できるのはスーパーユーザーのみです。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

Amazon Redshift コンソールを使用して、データ共有でオブジェクトを追加または削除することもできます。詳細については、「データ共有へのデータ共有オブジェクトの追加」、「データ共有からのデータ共有オブジェクトの削除」、および「AWS Data Exchange データ共有の編集」を参照してください。

3. データ共有へのアクセスを AWS Data Exchange に対し承認するには、以下のいずれかを実行します。

   • `aws redshift authorize-data-share` API で ADX キーワードを使用して、データ共有へのアクセスを AWS Data Exchange に対し明示的に許可します。これにより、AWS Data Exchangeはサービスアカウント内のデータ共有を認識できるようになり、コンシューマーのデータ共有への関連付けを管理することが可能になります。

   ```
   aws redshift authorize-data-share
   --data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
   {PRODUCER_CLUSTER_NAMESPACE}/salesshare
   --consumer-identifier ADX
   ```

   API の `AuthorizeDataShare` および `DeauthorizeDataShare` で条件付きキー `ConsumerIdentifier` を使用すると、AWS Data Exchangeによる (IAM ポリシー内の) これら 2 つの API への呼び出しを、明示的に許可または拒否できます。

   ```
   {
       "Version": "2012-10-17",
   ```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "redshift:AuthorizeDataShare",
                "redshift:DeauthorizeDataShare"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "redshift:ConsumerIdentifier": "ADX"
                }
            }
        }
    ]
}
```

- Amazon Redshift コンソールを使用して、AWS Data Exchange データ共有に関する認可を許可または削除します。詳細については、「データ共有の承認と承認の取り消し」を参照してください。
- オプションで、データ共有を AWS Data Exchange データセットにインポートするときの AWS Data Exchange データ共有へのアクセスを暗黙的に認可することもできます。

AWS Data Exchange データ共有へのアクセス権限を削除するには、`aws redshift deauthorize-data-share`(API) のオペレーションにおいてキーワード ADX を指定します。これにより、サービスアカウント内のデータ共有を認識することと、データ共有からの関連付け削除の管理を行うことを AWS Data Exchange に許可します。

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier ADX
```

4. クラスターで作成されたデータ共有を一覧表示し、データ共有の内容を調べます。

   次の例は、SalesShare という名前のデータ共有の情報を表示します。詳細については、DESC DATASHAREおよびSHOW DATASHARESを参照してください。

```
DESC DATASHARE salesshare;
```

```
 producer_account  |          producer_namespace           | share_type | share_name
 | object_type |           object_name          |   include_new
-------------------+---------------------------------------+------------
+------------+-------------+---------------------------------+-------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_users_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_venue_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_category_redshift|
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_date_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_event_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_listing_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | table       | public.tickit_sales_redshift   |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | schema      | public                         |  t
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | OUTBOUND   | salesshare
 | view        | public.sales_data_summary_view |
```

次の例は、プロデューサークラスター内のアウトバウンドデータ共有を表示します。

```
SHOW DATASHARES LIKE 'sales%';
```

出力は次の例のようになります。

```
share_name | share_owner  | source_database | consumer_database | share_type |
 createdate        | is_publicaccessible  | share_acl | producer_account |
 producer_namespace
-----------+--------------+-----------------+-------------------+------------
+---------------------+----------------------+-----------+------------------
+---------------------------------------
salesshare |    100       | dev             |                   |  OUTBOUND
 | 2020-12-09 02:27:08 |          True        |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

詳細については、「DESC DATASHARE」および「SHOW DATASHARES」を参照してください。

SVV_DATASHARES、SVV_DATASHARE_CONSUMERS、および SVV_DATASHARE_OBJECTS を使用して、データ共有、そのデータ共有内のオブジェクト、およびデータ共有のコンシューマーを表示することもできます。

5. データ共有を削除します。他の AWS アカウント と共有している AWS Data Exchange データ共有は、DROP DATASHARE ステートメントを使用して削除しないことをお勧めします。これらのアカウントはデータ共有へのアクセス権を失います。このアクションを元に戻すことはできません。これは、AWS Data Exchangeにおけるデータ製品の提供規約に違反する可能性があります。AWS Data Exchange のデータ共有を削除する場合は、「DROP DATASHARE の使用に関する注意事項」を参照してください。

   次の例では、SalesShare という名前のデータ共有を削除します。

```
DROP DATASHARE salesshare;
ERROR:  Drop of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value '620c871f890c49'
```

   AWS Data Exchange データ共有の削除を許可するには、datashare_break_glass_session_var 変数を設定した上で、DROP DATASHARE ステートメントを再度実行します。AWS Data Exchange のデータ共有を削除する場合は、「DROP DATASHARE の使用に関する注意事項」を参照してください。

   Amazon Redshift コンソールを使用してデータ共有を削除することもできます。詳細については、「アカウント内で作成された AWS Data Exchange データ共有の削除」を参照してください。

6. ALTER DATASHARE コマンドを使用すると、データ共有から任意の時点でオブジェクトを削除できます。REVOKE USAGE ON を使用すると、特定のコンシューマーについて、データ共有への許可を取り消すことができます。このコマンドは、データ共有内にあるオブジェクトの USAGE 許可を取り消し、すべてのコンシューマークラスターへのアクセスを即座に停止します。アクセス許可が取り消されたあとは、データベースやテーブルに関するものを含め、データ共有およびメタデータを一覧表示するためのクエリは、共有されたオブジェクトを返さなくなります。

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

   Amazon Redshift コンソールを使用して、データ共有を編集することもできます。詳細については、「AWS Data Exchange データ共有の編集」を参照してください。

7. AWS Data Exchangeデータ共有で、GRANT USAGE を付与または取り消す。AWS Data Exchangeデータ共有では、GRANT USAGE を付与または取り消すことはできません。次に、AWS Data Exchange が管理するデータ共有への GRANT USAGE 許可が AWS アカウントに付与された場合に発生する、エラーの例を示します。

```
CREATE DATASHARE salesshare MANAGEDBY ADX;
```

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910';
ERROR:  Permission denied to add/remove consumer to/from datashare salesshare.
 Datashare consumers are managed by ADX.
```

詳細については、「GRANT」または「REVOKE」を参照してください。

AWS Data Exchange コンソールでデータ共有製品を作成および公開するプロデューサークラスターの管理者は、以下のステップに従います。

- AWS Data Exchange データ共有が作成されると、プロデューサーが新しいデータセットを作成し、アセットのインポートとリビジョンの作成を行って、新しい製品を作成して公開します。

  データセットの作成には、Amazon Redshift コンソールを使用します。詳細については、「AWS Data Exchange でのデータセットの作成」を参照してください。

  詳細については、「Providing data products on AWS Data Exchange」を参照してください。

コンシューマーとしての AWS Data Exchange データ共有の使用

Amazon Redshift では、データのコピーを保存または管理することなく、AWS Data Exchange からデータセットにアクセスして分析できます。

AWS Data Exchange データ共有が使用可能なデータ製品の検出と、Amazon Redshift データへのクエリを行うコンシューマーは、以下のステップに従います。

1. AWS Data Exchange コンソール上で、AWS Data Exchangeデータ共有が含まれるデータ製品を検出しサブスクライブします。

   サブスクリプションが開始されると、AWS Data Exchange データ共有が含まれるデータセットにアセットとしてインポートされたライセンス付き Amazon Redshift データにアクセスできます。

AWS Data Exchange データ共有が含まれるデータ製品の使用開始方法に関する詳細については、「AWS Data Exchange でのデータ製品のサブスクライブ」を参照してください。

2. 必要に応じて、Amazon Redshift コンソールで Amazon Redshift クラスターを作成します。

   クラスターの作成方法については、「クラスターの作成」を参照してください。

3. 利用可能なデータ共有を一覧表示し、各データ共有のコンテンツを表示します。詳細については、「DESC DATASHARE」および「SHOW DATASHARES」を参照してください。

   次の例では、指定されたプロデューサー名前空間のインバウンドデータ共有の情報を表示します。DESC DATASHARE をコンシューマークラスターの管理者として実行する際に、インバウンドデータ共有を表示するには、ACCOUNT および NAMESPACE のオプションを指定する必要があります。

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

 producer_account  |          producer_namespace          | share_type | share_name
 | object_type |           object_name           |   include_new
-------------------+--------------------------------------+------------
+------------+-------------+---------------------------------+------------------
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_users_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_venue_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_category_redshift |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_date_redshift     |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_event_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_listing_redshift  |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | table       | public.tickit_sales_redshift    |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | schema      | public                          |
 123456789012      | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare
 | view        | public.sales_data_summary_view  |
```

クラスターのスーパーユーザーのみがこれを行うことができます。SVV_DATASHARES を使用してデータ共有を表示し、SVV_DATASHARE_OBJECTS を使用してデータ共有内のオブジェクトを表示することもできます。

次の例では、コンシューマークラスター内のインバウンドデータ共有を表示します。

```
SHOW DATASHARES LIKE 'sales%';


 share_name | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace
------------+-------------+-----------------+-------------------+------------
+------------+---------------------+-----------+------------------
+-------------------------------------
 salesshare |             |                 |                   | INBOUND
    |            |          t          |           |   123456789012    |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

4. データ共有を参照するローカルデータベースを作成します。AWS Data Exchange データ共有のためにローカルデータベースを作成するには、ACCOUNT および NAMESPACE オプションを指定する必要があります。詳細については、「CREATE DATABASE」を参照してください。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF ACCOUNT '123456789012'
 NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

ローカルデータベース内のオブジェクトへのアクセスをより細かく制御する場合は、データベースの作成時に WITH PERMISSIONS 句を使用します。これにより、ステップ 6 でデータベース内のオブジェクトにオブジェクトレベルのアクセス許可を付与できます。

```
CREATE DATABASE sales_db WITH PERMISSIONS FROM DATASHARE salesshare OF ACCOUNT
 '123456789012' NAMESPACE '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

SVV_REDSHIFT_DATABASES ビューにクエリを実行すると、データ共有から作成したデータベースを確認できます。これらのデータベースに直接接続したり、コンシューマークラスターのローカルデータベースに接続し、データベース間クエリを実行して、データ共有データベースのデータをクエリしたりすることができます。既存のデータ共有から作成されたデータベースオブジェクトの上にデータ共有を作成することはできません。ただし、コンシューマークラスター上

の別のテーブルにデータをコピーし、必要な処理を実行してから、作成された新しいオブジェクトを共有できます。

Amazon Redshift コンソールを使用して、データ共有からデータベースを作成することもできます。詳細については、「データ共有からのデータベースの作成」を参照してください。

5. (オプション) 外部スキーマを作成して、コンシューマークラスターにインポートされたコンシューマデータベース内の特定のスキーマを参照し、詳細なアクセス許可を割り当てます。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA
 'public';
```

6. 必要に応じて、データ共有から作成されたデータベースおよびスキーマ参照に対するアクセス許可を、コンシューマークラスター内のユーザーまたはロールに付与します。詳細については、「GRANT」または「REVOKE」を参照してください。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

WITH PERMISSIONS を使用することなくデータベースを作成している場合は、データ共有から作成されたデータベース全体に対するアクセス許可のみをユーザーおよびロールに割り当てることができます。場合によっては、データ共有から作成されたデータベースオブジェクトのサブセットに対して、詳細なコントロールが必要になります。その場合は、データ共有内の特定のスキーマを指す外部スキーマ参照を作成し (前出のステップを参照)、スキーマレベルの詳細なアクセス許可を提供できます。

また、共有オブジェクトの上に遅延バインディングビューを作成し、これらを使用して詳細なアクセス許可を割り当てることもできます。また、プロデューサークラスターで必要な粒度で追加のデータ共有を作成するよう検討することもできます。データ共有から作成されたデータベースへのスキーマ参照は、必要な数を作成できます。

ステップ 4 で WITH PERMISSIONS を使用してデータベースを作成した場合は、共有データベース内のオブジェクトにオブジェクトレベルのアクセス許可を割り当てる必要があります。USAGE アクセス許可のみを持つユーザーは、追加のオブジェクトレベルのアクセス許可が付与されるまで、WITH PERMISSIONS を使用して作成されたデータベース内のオブジェクトにはアクセスできません。

```
GRANT SELECT ON sales_db.public.tickit_sales_redshift to Bob;
```

7. データ共有内の共有オブジェクト内のデータをクエリします。

   コンシューマークラスター上のコンシューマデータベースおよびスキーマに対するアクセス許可を持つユーザーおよびロールは、任意の共有オブジェクトのメタデータを探索およびナビゲートできます。また、コンシューマークラスター内のローカルオブジェクトを探索およびナビゲートすることもできます。これを行うには、JDBC または ODBC ドライバー、SHOW コマンド、または SVV_ALL および SVV_REDSHIFT ビューを使用します。

   プロデューサークラスターには、各スキーマ内のデータベース、テーブル、およびビューに多数のスキーマが含まれる場合があります。コンシューマー側のユーザーは、データ共有を通じて利用可能になったオブジェクトのサブセットのみを表示できます。これらのユーザーは、プロデューサークラスターからのメタデータ全体を表示することはできません。このアプローチは、データ共有による詳細なメタデータセキュリティの制御を提供します。

   引き続きローカルクラスターのデータベースに接続します。ただし、3 つの部分からなる database.schema.table 表記を使用して、データ共有から作成されたデータベースとスキーマから読み込むこともできるようになりました。表示されているすべてのデータベースにまたがるクエリを実行できます。これらは、クラスター上のローカルデータベースでも、データ共有から作成されたデータベースでもかまいません。または、これらのコンシューマーデータベースに直接接続し、部分表記で共有オブジェクトに対してクエリを実行できます。

   完全な資格を使用してデータにアクセスできます。詳細については、「クロスデータベースクエリの例」を参照してください。

```
SELECT * FROM sales_db.public.tickit_sales_redshift ORDER BY 1,2 LIMIT 5;

 salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime
---------+--------+----------+---------+---------+--------+---------+-----------
+------------+---------------------
       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |    728.00 |
   109.20 | 2008-02-18 02:36:48
       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |     76.00 |
    11.40 | 2008-06-06 05:00:16
       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |    350.00 |
    52.50 | 2008-06-06 08:26:17
       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |    175.00 |
    26.25 | 2008-06-09 08:38:52
```

```
        5 |        6 |     47402 |    14115 |     8240 |    2069 |        2 |    154.00 |
    23.10 | 2008-08-31 09:17:02
```

SELECT ステートメントは、共有オブジェクトに対してのみ使用できます。ただし、別のローカルデータベース内の共有オブジェクトからデータをクエリすることで、コンシューマークラスターにテーブルを作成できます。

クエリに加えて、コンシューマーは共有オブジェクトのビューを作成できます。遅延バインディングビューまたはマテリアライズドビューのみがサポートされます。Amazon Redshift は、共有データの通常のビューをサポートしていません。コンシューマーが作成するビューは、複数のローカルデータベースまたはデータ共有から作成されたデータベースにまたがることができます。詳細については、「CREATE VIEW」を参照してください。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.
CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

AWS Lake Formation 管理のデータ共有の使用

Amazon Redshift では、AWS Lake Formation データ共有を介して共有されるデータにアクセスして分析できます。AWS Lake Formation データ共有を使用すると、基盤データをコピーまたは移動することなく、AWS アカウントと Amazon Redshift クラスター間で安全なデータ共有が可能になります。

AWS Lake Formation でデータを共有すると、Amazon Redshift データ共有の AWS Lake Formation アクセス許可を一元的に定義し、データ共有内のオブジェクトへのユーザーアクセスを制限できます。

トピック

- プロデューサーとして Lake Formation 管理のデータ共有を使用する
- コンシューマーとしての Lake Formation 管理のデータ共有を使用する

プロデューサーとして Lake Formation 管理のデータ共有を使用する

Amazon Redshift では、 AWS Lake Formation マネージドデータ共有をプロデューサーとして使用して、AWS アカウントと Amazon Redshift クラスター間でライブデータを安全に共有できます。Lake Formation マネージドデータ共有は、Amazon Redshift クラスターのライブデータを他の AWS アカウントやサービスと共有できるオブジェクトです。

プロデューサークラスターまたはワークグループの管理者は、以下のステップに従って、Lake Formation とデータ共有を行います。

1. クラスター内にデータ共有を作成し、AWS Lake Formation にデータ共有に対するアクセス権を付与します。

   クラスターのスーパーユーザーとデータベースの所有者は、データ共有を作成できます。各データ共有は、作成時にデータベースに関連付けられます。そのデータベースのオブジェクトのみがそのデータ共有で共有できます。同じデータベース上に、同じ粒度または異なる粒度のオブジェクトを使用して、複数のデータ共有を作成できます。クラスター上に作成できるデータ共有の数に制限はありません。

   ```
   CREATE DATASHARE salesshare;
   ```

2. データ共有にオブジェクトを追加します。プロデューサークラスターまたはワークグループの管理者は、使用可能なデータ共有オブジェクトの管理を継続します。データ共有にオブジェクトを追加するには、オブジェクトを追加する前にスキーマを追加します。スキーマを追加する場合、Amazon Redshift はその下にすべてのオブジェクトを追加するわけではありません。それらを明示的に追加する必要があります。詳細については、「データ共有の変更」を参照してください。

   ```
   ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
   ```

   データ共有にビューを追加することもできます。標準ビュー、遅延バインディングビューおよびマテリアライズドビューがサポートされます。

   ```
   CREATE VIEW public.sales_data_summary_view AS SELECT * FROM
    public.tickit_sales_redshift;
   ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
   ```

ALTER DATASHARE を使用して、特定のスキーマ内のスキーマ、テーブル、ビューを共有します。スーパーユーザー、データ共有の所有者、またはデータ共有に対する ALTER または ALL 許可を持つユーザーは、データ共有を変更して、それに対するオブジェクトの追加または削除を実行できます。データベースユーザーは、オブジェクトの所有者であるか、またはオブジェクトに対する SELECT、USAGE、もしくは ALL 許可を持っている必要があります。

スキーマを指定して作成された新しいテーブル、ビューをデータ共有に追加するには、INCLUDENEW 句を使用します。データ共有とスキーマの各ペアについて、このプロパティを変更できるのはスーパーユーザーのみです。

```
ALTER DATASHARE salesshare ADD SCHEMA PUBLIC;
ALTER DATASHARE salesshare SET INCLUDENEW = TRUE FOR SCHEMA PUBLIC;
```

3. Lake Formation 管理者アカウントにデータ共有へのアクセスを許可します。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '012345678910' VIA DATA CATALOG;
```

使用を取り消すには、次のコマンドを使用します。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '012345678910' VIA DATA CATALOG;
```

4. `aws redshift authorize-data-share` API オペレーションを使用して Lake Formation のデータ共有へのアクセスを許可します。これにより、Lake Formation はサービスアカウント内のデータ共有を認識して、コンシューマーのデータ共有への関連付けを管理できます。

```
aws redshift authorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

Lake Formation 管理のデータ共有から認可を削除するには、`aws redshift deauthorize-data-share` API オペレーションを使用します。これにより、AWS Lake Formation がサービスアカウント内のデータ共有を認識できるようになり、認可を削除することができます。

```
aws redshift deauthorize-data-share
--data-share-arn arn:aws:redshift:us-east-1:{PRODUCER_ACCOUNT}:datashare:
{PRODUCER_CLUSTER_NAMESPACE}/salesshare
--consumer-identifier {"DataCatalog/<consumer-account-id>"}
```

プロデューサークラスターまたはワークグループの管理者がコンシューマークラスターまたはワークグループとデータを共有する必要がなくなったと判断した場合は、いつでも、DROP DATASHARE を使用してデータ共有を削除し、データ共有の承認を解除し、またはデータ共有アクセス許可を取り消すことができます。Lake Formation 内に関連付けられたアクセス許可とオブジェクトは自動的には削除されません。

```
DROP DATASHARE salesshare;
```

Lake Formation アカウントにデータ共有の管理を許可すると、Lake Formation 管理者はデータ共有を見つけ、データカタログの ARN にデータ共有を関連付け、データ共有にリンクする AWS Glue Data Catalog 内にデータベースを作成できます。AWS CLI を使用してデータ共有を関連付けるには、associate-data-share-consumer コマンドを使用します。AWS リージョン 全体でデータ共有を共有するには、`associate-data-share-consumer` コマンドで `--region` パラメータを指定するか、AWS コンソールを使用してデータコンシューマーを選択します。次の例は、Lake Formation 管理のデータ共有をリージョン間で共有する方法を示します。

```
aws redshift associate-data-share-consumer --region <region-1>
--data-share-arn 'arn:aws:redshift:us-
east-1:12345678912:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/sample_share'
--consumer-arn 'arn:aws:glue:<region-1>:111912345678:catalog'
```

Lake Formation 管理者は、データ共有内のオブジェクトが Lake Formation 内のオブジェクトにどのようにマッピングされるかを定義するローカルリソースを作成する必要もあります。データ共有の検出とローカルリソースの作成の詳細については、「Amazon Redshift データ共有でのデータに対するアクセス許可の管理」を参照してください。

コンシューマーとしての Lake Formation 管理のデータ共有を使用する

Amazon Redshift では、AWS Lake Formation データ共有を通じて共有されたデータにアクセスして分析できます。データ共有は、さまざまなデータソースからのテーブルやデータベースなどのデータオブジェクトのコレクションを含むデータ製品です。

AWS Lake Formation 管理者がデータ共有への招待を発見して、そのデータ共有にリンクする AWS Glue Data Catalog にデータベースを作成した後、コンシューマークラスターまたはワークグループの管理者は、クラスターをデータ共有と AWS Glue Data Catalog 内のデータベースに関連付け、コンシューマークラスターまたはワークグループのローカルデータベースを作成し、Amazon Redshift

コンシューマークラスターまたはワークグループ内のユーザーおよびロールにクエリを開始するアクセス許可を付与できます。以下のステップに従って、クエリのアクセス許可を設定します。

1. 必要に応じて、Amazon Redshift コンソールで、コンシューマークラスターまたはワークグループとして機能する Redshift クラスターを作成します。クラスターの作成方法については、「クラスターの作成」を参照してください。

2. AWS Glue Data Catalog コンシューマークラスターまたはワークグループ内のどのデータベースにユーザーがアクセスできるかを一覧表示するには、SHOW DATABASES コマンドを実行します。

   ```
   SHOW DATABASES FROM DATA CATALOG [ACCOUNT <account-id>,<account-id2>] [LIKE
    <expression>]
   ```

   これにより、AWS Glue データベースの ARN、データベース名、データ共有に関する情報など、データカタログから利用できるリソースが一覧表示されます。

3. SHOW DATABASES から得られた AWS Glue データベース ARN を使用して、コンシューマークラスターまたはワークグループにローカルデータベースを作成します。詳細については、「データベースの作成」を参照してください。

   ```
   CREATE DATABASE lf_db FROM ARN <lake-formation-database-ARN> WITH [NO] DATA CATALOG
    SCHEMA [<schema>];
   ```

4. 必要に応じて、データ共有から作成されたデータベースおよびスキーマ参照に対するアクセス許可を、コンシューマークラスターまたはワークグループ内のユーザーおよびロールに付与します。詳細については、「許可」または「取り消し」を参照してください。CREATE USER コマンドで作成されたユーザーは、Lake Formation と共有されているデータ共有内のオブジェクトにはアクセスできないことに注意してください。Redshift と Lake Formation の両方にアクセスできる IAM ユーザーのみが、Lake Formation と共有されているデータ共有にアクセスできます。

   ```
   GRANT USAGE ON DATABASE sales_db TO IAM:Bob;
   ```

   コンシューマークラスターまたはワークグループの管理者は、データ共有から作成されたデータベース全体に対するアクセス許可のみをユーザーおよびロールに割り当てることができます。場合によっては、データ共有から作成されたデータベースオブジェクトのサブセットに対して、詳細なコントロールが必要になります。

また、共有オブジェクトの上に遅延バインディングビューを作成し、これらを使用して詳細なアクセス許可を割り当てることもできます。また、プロデューサークラスターまたはワークグループで必要な詳細度で追加のデータ共有を作成するよう検討することもできます。データ共有から作成されたデータベースへのスキーマ参照は、いくつでも作成できます。

5. データベースユーザーは、SVV_EXTERNAL_TABLES ビューと SVV_EXTERNAL_COLUMNS ビューを使用して、AWS Glue データベース内のすべての共有テーブルまたは列を検索できます。

```
SELECT * from svv_external_tables WHERE redshift_database_name = 'lf_db';

SELECT * from svv_external_columns WHERE redshift_database_name = 'lf_db';
```

6. データ共有内の共有オブジェクト内のデータをクエリします。

   コンシューマークラスターまたはワークグループ上のコンシューマデータベースおよびスキーマに対するアクセス許可を持つユーザーおよびロールは、任意の共有オブジェクトのメタデータを探索およびナビゲートできます。また、コンシューマークラスターまたはワークグループ内のローカルオブジェクトを探索およびナビゲートすることもできます。これを行うには、JDBC または ODBC ドライバー、または SV_ALL および SV_EXTERNAL ビューを使用します。

```
SELECT * FROM lf_db.schema.table;
```

   SELECT ステートメントは、共有オブジェクトに対してのみ使用できます。ただし、別のローカルデータベース内の共有オブジェクトからデータをクエリすることで、コンシューマークラスターにテーブルを作成できます。

```
// Connect to a local cluster database

// Create a view on shared objects and access it.

CREATE VIEW sales_data
AS SELECT *
FROM sales_db.public.tickit_sales_redshift
WITH NO SCHEMA BINDING;

SELECT * FROM sales_data;
```

コンソールを使用したデータ共有の管理

Amazon Redshift では、書き込みを伴うデータ共有をコンソールを使って管理することで、Amazon Redshift クラスターおよび AWS アカウント間のデータへのアクセスと管理を制御できます。以下のセクションでは、Amazon Redshift コンソールを使用して書き込みを伴うデータ共有を設定および管理するためのステップバイステップの手順について説明します。

Amazon Redshift コンソールを使用して、自分のアカウントで作成された、または他のアカウントから共有されたデータ共有を管理します。

データ共有を作成、編集、削除するには。アクセス許可が必要です。詳細については、Amazon Redshift でのデータ共有に対する許可の管理 を参照してください。

- プロデューサークラスターの管理者である場合は、[CLUSTERS (クラスター)] タブから、データ共有の作成、データコンシューマーの追加、データ共有オブジェクトの追加、データ共有からのデータベースの作成、データ共有の編集、またはデータ共有の削除を行うことができます。

  ナビゲーションメニューから、[Clusters (クラスター)] タブに移動し、クラスターリストからクラスターを選択します。次に、以下のいずれかを行ってください。

  - [Datashares] (データ共有) タブを選択し、[Datashares created in my namespace] (名前空間で作成されたデータ共有) セクションからデータ共有を選択します。次に、以下のいずれかを行ってください。
    - データ共有の作成

      データ共有が作成されると、データ共有オブジェクトまたはデータコンシューマーを追加できます。詳細については、データ共有へのデータ共有オブジェクトの追加およびデータ共有へのデータコンシューマーの追加を参照してください。
    - アカウントで作成されたデータ共有の編集
    - アカウント内で作成された データ共有の削除
  - [Datashares (データ共有)] を選択し、[Datashares from other clusters (他のクラスターのデータ共有)] セクションからデータ共有を選択します。次に、以下のいずれかを行ってください。
    - データ共有の作成
    - データ共有からのデータベースの作成
  - [Databases (データベース)] を選択し、[Databases (データベース)] セクションからデータベースを選択します。次に、[Create datashare (データ共有の作成)] を選択します。詳細については、「データ共有からのデータベースの作成」を参照してください。

> **Note**
>
> データベースおよびデータベース内のオブジェクトを表示したり、クラスター内のデータ共有を表示したりするには、データベースに接続します。詳細については、「データベースへの接続」を参照してください。

データベースへの接続

Amazon Redshift では、データウェアハウスクラスターへの接続を確立し、SQL クエリ、データのロード、または管理タスクを実行できます。データベースへの接続とは、クライアントアプリケーションまたはツールと Amazon Redshift クラスターの間に安全なチャネルを作成するプロセスを指します。以下のセクションでは、Amazon Redshift のデータベースに接続する方法をステップバイステップで説明します。

データベースに接続して、このクラスター内のデータベースおよびデータベース内のオブジェクトを表示したり、データ共有を表示したりします。

指定されたデータベースへの接続に使用されるユーザー認証情報には、すべてのデータ共有を表示するために必要となる許可が必要です。

ローカル接続がない場合は、次のいずれかの操作を行います。

- クラスターの詳細ページの [Databases] (データベース) タブにある [Databases] (データベース) または [Datashare objects] (データ共有オブジェクト) セクションで、[Connect to database] (データベースに接続) をクリックし、クラスター内のデータベースオブジェクトを表示します。
- [cluster details (クラスターの詳細)] ページの [Datashares (データ共有)] タブで、次のいずれかを実行します。
  - [Datashares from other clusters (他のクラスターからのデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択して、他のクラスターからのデータ共有を表示します。
  - [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択して、クラスター内のデータ共有を表示します。
- [Connect to database (データベースに接続)] ウィンドウで、次のいずれかを実行します。
  - [新しい接続を作成] を選択した場合は、[AWS Secrets Manager] を選択し、保存されているシークレットを使用して接続へのアクセスを認証します。

または、データベースの認証情報を使用して接続へのアクセスを認証するには、[Temporary credentials (一時的な認証情報)] を選択します。[Database name (データベース名)] と [Database user (データベースユーザー)] の値を指定します。

[接続] を選択します。

- [Use a recent connection (最近の接続を使用する)] を選択して、必要な許可を持つ別のデータベースに接続します。

  Amazon Redshift が自動的に接続を確立します。

データベース接続が確立されたら、データ共有の作成、データ共有のクエリ、またはデータ共有からのデータベースの作成を開始できます。

データ共有の作成

Amazon Redshift では、データ共有を使用して、Amazon Redshift クラスターまたは AWS アカウント間でライブデータを共有することができます。データ共有は、Amazon Redshift クラスターからのライブデータを他のクラスターまたは AWS アカウントと共有できるようにするコンシューマープロデューサーオブジェクトです。データ共有を作成すると、安全なデータ共有が可能になり、アクセスの制御を維持し、データが最新の状態であることを維持できます。以下のセクションでは、データ共有の作成と、ライブデータを安全に共有するためのスキーマ、テーブル、ビューなどのデータベースオブジェクトの追加について詳しく説明します。

データ共有の作成

プロデューサークラスターの管理者は、[cluster details (クラスターの詳細)] ページの [Database (データベース)] タブまたは [Datashares (データ共有)] タブからデータ共有を作成できます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Cluster Details (クラスターの詳細)] ページで次の操作を実行します。
   - [Databases (データベース)] タブの [Database (データベース)] セクションで、データベースを選択します。データベースの詳細ページが表示されます。

     [データ共有を作成] を選択します。データ共有はローカルデータベースからのみ作成できます。データベースに接続していない場合は、[データベースに接続] ページが表示されま

す。データベースへの接続 の手順に従って、データベースに接続します。最近接続したことがある場合は、[データ共有を作成] ページが表示されます。

- データベース接続がない場合は、[Datashares (データ共有)] タブの [Datashares (データ共有)] セクションで、データベースに接続します。

  [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションで、[Create datashare (データ共有の作成)] を選択します。[データ共有を作成] ページが表示されます。

4. [Datashare information] (データ共有情報) セクションで、以下のいずれかのオプションを選択します。
   - [Datashare] (データ共有) では、作成するデータ共有を選択します。異なる Amazon Redshift クラスター間、または同じ AWS アカウント または異なる AWS アカウント 内での、読み取り目的のデータ共有を指定します。
   - [AWS Data Exchange datashare] (データ共有) を選択し、AWS Data Exchange経由でデータをライセンスするためのデータ共有を作成します。
5. [Datashare name] (データ共有名)、[Database name] (データベース名)、および [Publicly accessible] (パブリックにアクセス可能) の値を指定します。

   データベース名を変更するときは、新しいデータベース接続を作成します。
6. [データ共有オブジェクト] セクションで、[追加] を選択します。[データ共有を追加] ページが表示されます。データ共有にオブジェクトを追加するには、データ共有へのデータ共有オブジェクトの追加 に従います。
7. [Data consumers] (データコンシューマー) セクションでは、Redshift アカウントに公開するか、AWS Glue Data Catalog に公開するかを選択できます。公開すると、Lake Formation を介してデータを共有するプロセスが開始されます。データ共有を Redshift アカウントに公開するということは、コンシューマークラスターとして機能する別の Redshift アカウントとデータを共有することを意味します。

   > **Note**
   >
   > データ共有が作成されると、設定を編集して他のオプションに公開することはできません。
8. [Create datashare (データ共有の作成)] を選択します。

Amazon Redshift がデータ共有を作成します。データ共有が作成されたら、データ共有からデータベースを作成できます。

データ共有へのデータ共有オブジェクトの追加

データ共有に 1 つ以上のオブジェクトを追加します。データ共有オブジェクトは、データコンシューマーに対して読み込み専用です。

データ共有オブジェクトを追加せずにデータ共有を作成し、後でオブジェクトを追加できます。

データ共有は、データ共有に少なくとも 1 つのオブジェクトを追加した場合にのみアクティブになります。

1. データ共有リストから、オブジェクトを追加するデータ共有を選択します。
2. [Add] を選択します。[データ共有オブジェクトを追加] ページが表示されます。
3. 他のデータ共有オブジェクトを追加する前に、少なくとも 1 つのスキーマをデータ共有に追加します。[Add and repeat (追加して繰り返す)] を選択して、複数のスキーマを追加します。
4. 指定したスキーマから選択したオブジェクトタイプの既存のオブジェクトをすべて追加するか、指定したスキーマから特定の個別オブジェクトを追加するかを選択できます。テーブルやビューまたはユーザー定義関数などの [オブジェクトタイプ] を選択します。
5. [Add and repeat (追加して繰り返す)] を選択して、指定したスキーマとデータ共有オブジェクトを追加し、別のオブジェクトとオブジェクトを引き続き追加できます。

データ共有へのデータコンシューマーの追加

1 つ以上のデータコンシューマーをデータ共有に追加できます。データコンシューマーは、一意に識別された Amazon Redshift クラスターであるクラスター名前空間、または AWS アカウントにすることができます。

公開アクセスが設定されているクラスターへのデータ共有は、無効にするか有効にするかを明示的に選択する必要があります。

- [Add cluster namespaces to the datashare (クラスター名前空間をデータ共有に追加)] を選択します。名前空間は Amazon Redshift クラスターのグローバル一意識別子 (GUID) です。
- [Add AWS アカウント] (追加) を選択して、データ共有します。指定された AWS アカウントには、データ共有へのアクセス許可がある必要があります。

データ共有へのデータレイクテーブルの追加

データ共有では、データプロデューサーは、スキーマやテーブルなどのきめ細かなデータベースオブジェクトを、同じまたは異なる AWS アカウントのコンシューマーと安全に共有できます。プロ

デューサーは、リージョン間でオブジェクトを共有することもできます。このトピックでは、データレイク、特に AWS Glue データカタログから、データ共有にオブジェクトを追加する方法について説明します。ここでは、次の 2 つのユースケースを対象としています。

- データレイクからテーブルを参照するデータ共有に遅延バインディングビューを追加する – Lake Formation などの外部ソースデータに対するアクセス許可の定義などの事前設定がすでに完了している可能性があるため、これはコンシューマーにとって便利です。さらに、データ共有に追加されたビューは、データレイクのテーブルを Redshift ネイティブテーブルと結合できるという利点もあります。
- 外部スキーマからデータ共有にテーブルを直接追加する – これにより、コンシューマーは、追加のレイヤーやロジックなしでデータレイクからのオブジェクトを利用できます。コンシューマーは、テーブルをクエリするか、コンシューマーのテーブルに結合できます。

これらのケースは、CREATE EXTERNAL SCHEMA を使用して Redshift の AWS データカタログからテーブルを参照した後に適用されます。AWS データカタログのすべてのテーブルをソースとすることができます。

> **Note**
>
> データ共有に追加するデータレイクテーブルには、Lake Formation に登録されたテーブルと AWS Glue データカタログテーブルを含めることができます。

外部スキーマと外部テーブルの作成

以下のセクションのデータ共有に追加する外部スキーマと外部テーブルを作成します。これらは事前ステップです。すでにこれらのステップを完了している場合、スキップすることができます。

1. プロデューサーで、Amazon S3 に保存されているデータレイクデータを参照する外部スキーマを作成します。外部スキーマは AWS Glue データカタログを参照します。サンプル内のロールとリージョンの例は次のとおりです。

```
CREATE EXTERNAL SCHEMA external_schema_name FROM DATA CATALOG
DATABASE 'glue_database_name'
IAM_ROLE 'arn:aws:iam::123456789012:role/sample-role'
REGION 'us-east-1';
```

2. 外部スキーマにデータレイクテーブルを作成します。

```
CREATE EXTERNAL TABLE external_schema_name.sales(
salesid INTEGER,
sellerid INTEGER,
buyerid INTEGER,
saledate DATE,
pricepaid DECIMAL(8,2))
ROW FORMAT delimited
FIELDS TERMINATED BY '\t'
STORED AS textfile
LOCATION 's3://redshift-downloads/tickit/spectrum/sales/';
```

サンプルは LOCATION を含みます。s3://{bucket_name}/{folder}/ の形式でフォルダを指定する必要があります。フォルダの長さは 1 文字以上である必要があります。オプションで、サブフォルダを含めることができます。データレイクにテーブルを作成する他の例については、CREATE EXTERNAL TABLE の「例」を参照してください。

> **Note**
>
> 共有は、プロデューサーの IAM ロールが SELECT アクセスを持つテーブルでのみサポートされます。

データレイクテーブルを参照する遅延バインディングビューをデータ共有に追加する

AWS データカタログから外部スキーマに基づいてテーブルを作成し、データ共有に追加する最も一般的な方法は、作成したテーブルを参照する Redshift 遅延バインディングビューを追加することです。これには、データレイクからのデータが含まれます。以下にそのステップを示します。

1. 作成済みの外部テーブルを参照する遅延バインディングビューを作成します。

```
CREATE VIEW lbv AS
select * from external_schema_name.sales, other_schema.t1
WITH NO SCHEMA BINDING;
```

2. データ共有にビュースキーマを追加します。これは、遅延バインディングビューを含むローカルスキーマです。

```
ALTER DATASHARE dsx_datashare ADD SCHEMA public;
```

3. 遅延バインディングビューによって参照されるテーブルを含むスキーマをデータ共有に追加します。スキーマの追加は、スキーマにローカルデータベースオブジェクトまたはデータレイクからのオブジェクトが含まれているかどうかにかかわらず、データ共有に追加されるビューで参照されるすべてのベーステーブルに必要です。遅延バインディングビューを追加する前に、このスキーマを追加する必要があることに注意してください。

```
ALTER DATASHARE dsx_datashare ADD SCHEMA external_schema_name;
ALTER DATASHARE dsx_datashare ADD SCHEMA other_schema;
```

4. SQL コマンドを使用して、データ共有にビューを追加します。テーブル名にはスキーマプレフィックスが含まれていることに注意してください。

```
ALTER DATASHARE my_datashare ADD TABLE public.lbv;
```

5. ビューとスキーマがデータ共有に正常に追加されていることを確認します。

```
SELECT * FROM svv_datashare_objects WHERE share_name = 'my_datashare';
```

6. コンシューマー管理者は、データ共有からデータベースを作成し、コンシューマーユーザーに使用許可を付与します。

これらのステップを完了すると、データ共有ビューにアクセスできるデータベースコンシューマーユーザーはデータをクエリできます。

データレイクテーブルをデータ共有に直接追加する

外部スキーマテーブルのデータ共有への追加は、ビューの追加と似ています。これは、コンシューマーがデータレイクテーブルを元の状態でクエリする場合や、テーブルをコンシューマーがコンシューマーデータウェアハウスのテーブルに結合する場合に適しています。以下の手順では、SQL を使用してデータ共有にデータレイクテーブルを追加する方法を示します。

1. このトピックの最初のセクションの説明に従って、外部スキーマと外部テーブルを作成します。
2. 外部スキーマ内の既存のテーブルを検出して、作成したテーブルが使用可能であることを確認します。

```
SELECT * FROM svv_external_tables WHERE schemaname = 'external_schema_name';
```

3. データ共有に外部スキーマを追加します。

```
ALTER DATASHARE my_datashare ADD SCHEMA external_schema_name;
```

4. データ共有に外部テーブルを追加します。テーブル名にはスキーマプレフィックスが含まれていることに注意してください。

```
ALTER DATASHARE my_datashare ADD TABLE external_schema_name.sales;
```

5. テーブルがデータ共有に正常に追加されていることを確認します。

```
SELECT * FROM svv_datashare_objects WHERE share_name = 'my_datashare';
```

   詳細については、「AWS アカウント内のデータへの読み取りアクセスの共有」を参照してください。

6. 共有データを受信するデータベースであるコンシューマーでは、管理者はデータ共有を関連付けて、ユーザーがクエリできるようにします。このステップの実行方法の詳細については、「他のアカウントからのコンシューマーとしてのデータ共有の管理」を参照してください。

管理者がこれらのステップを完了すると、コンシューマーのデータベースユーザーは、共有テーブルからデータを取得するためのクエリを書き込み、コンシューマーの他のテーブルに結合できるようになります。

データ共有にデータレイクオブジェクトを追加する際の注意事項

データ共有でデータレイクからのテーブルとビューを使用する場合、注意すべき項目がいくつかあります。

- AWS CloudTrail でのログ記録 — データプロデューサーアカウントは、AWS CloudTrail ログを使用して、データ共有を介して共有されたデータレイクテーブルがいつアクセスされたかを監査できます。
  - ログデータを使用してデータアクセスを制御する — CloudTrail ログには、Redshift データ共有プロデューサーとコンシューマーの両方を含む、共有テーブルにアクセスするユーザーに関する詳細が記録されます。識別子は、`AssumeRole` CloudTrail ログの `ExternalId` フィールドで確認できます。データ所有者は、アクションを使用して IAM ポリシーのデータアクセスに対する追加の制限を設定できます。ポリシーによるデータアクセスの定義の詳細については、「第三者が所有する AWS アカウントへのアクセス」を参照してください。

- セキュリティとコンシューマーのアクセス許可 – Lake Formation に登録されたテーブルの場合、Amazon S3 リソースは Lake Formation によって保護され、Lake Formation が提供する認証情報によって利用可能になります。

データ共有にデータレイクオブジェクトを追加する際の請求に関する考慮事項

以下は、データ共有内のデータレイクオブジェクトの保存とスキャンにかかるコストの詳細です。

- コンシューマーがデータレイクから共有オブジェクトをクエリすると、スキャンのコストがコンシューマーに請求されます。
  - コンシューマーがプロビジョニングされたクラスターの場合、Redshift は Redshift Spectrum を使用して Amazon S3 データをスキャンします。そのため、Spectrum のコストはコンシューマーアカウントに請求されます。
  - コンシューマーが Amazon Redshift Serverless ワークグループの場合、Spectrum に追加の料金はかかりません。
- バケットの一覧表示などのストレージとオペレーションの Amazon S3 コストは、各 Amazon S3 バケットを所有するアカウントに請求されます。

Amazon Redshift Serverless の請求の詳細については、「Amazon Redshift Serverless での請求」を参照してください。請求と料金の詳細な情報については、「Amazon Redshift の料金」を参照してください。

データ共有の承認と承認の取り消し

プロデューサークラスターの管理者として、データ共有へのアクセスを許可するデータコンシューマー、または許可を削除するデータコンシューマーを選択します。認可されたデータコンシューマーは、データ共有でアクションを実行するための通知を受け取ります。クラスター名前空間をデータコンシューマーとして追加する場合は、承認を実行する必要はありません。

前提条件: データ共有の認可を承認または削除するには、データ共有に少なくとも 1 つのデータコンシューマーが追加されている必要があります。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。データ共有リストページが表示されます。
3. [In my account (アカウント内)] を選択します。

4. [Datashares in my account (アカウントのデータ共有)] セクションで、次のいずれかの操作を行います。
   - 認可するコンシューマークラスターを 1 つ以上選択します。[Authorize data consumers (データコンシューマーの認可)] ページが表示されます。次に、[Authorize] を選択します。

     データ共有の作成時に [AWS Glue Data Catalog に発行] を選択した場合、データ共有の許可は Lake Formation アカウントにのみ付与できます。

     AWS Data Exchange データ共有の場合、一度に 1 つのデータ共有しか認可できません。

     AWS Data Exchange データ共有を承認すると、データ共有を AWS Data Exchange サービスと共有し、ユーザーに代わって AWS Data Exchange がデータ共有へのアクセスを管理することを許可することになります。AWS Data Exchange は、コンシューマーが製品をサブスクライブする際、コンシューマーアカウントを AWS Data Exchange データ共有のデータコンシューマーとして追加することで、コンシューマーがアクセスできるようにします。AWS Data Exchange には、データ共有への読み取りアクセス許可はありません。
   - 認可を削除するコンシューマークラスターを 1 つ以上選択します。次に、[Remove authorization (認可の削除)] を選択します。

データコンシューマーが認可されると、データ共有オブジェクトにアクセスし、データをクエリするコンシューマデータベースを作成できます。

認可が削除された直後から、データコンシューマーはデータ共有にアクセスできなくなります。

他のアカウントからのコンシューマーとしてのデータ共有の管理

Amazon Redshift では、他の AWS アカウントからのデータ共有を消費し、クロスアカウントデータ共有とコラボレーションが可能です。データ共有は、異なる AWS アカウントにある場合でも Amazon Redshift クラスター間でライブデータを共有する安全な方法です。以下のセクションでは、アクセスの設定、共有データのクエリ、コンシューマーとしてのデータ共有アクティビティのモニタリングに関する詳細な手順について説明します。

データ共有の関連付け

コンシューマークラスターの管理者は、他のアカウントから共有されている 1 つ、または複数のデータ共有を、AWSアカウント全体、またはアカウント内の特定のクラスター名前空間に関連付けることができます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。データ共有リストページが表示されます。
3. [From other accounts (その他のアカウント)] を選択します。
4. [Datashares from other accounts] (他のアカウントからのデータ共有) セクションで、関連付けを行うデータ共有を選択し、[Associate] (関連付け) をクリックします。[Associate datashare] (データ共有の関連付け) ページが表示されたら、以下の関連付けタイプのいずれかを選択します。
   - AWS アカウント内の異なる AWS リージョンにわたって、既存および将来に作成されるクラスター名前空間のすべてをデータ共有に関連付けるには、[Entire AWS account] (Amazon アカウント全体) をクリックします。その後、[関連付け] を選択します。

     データ共有が AWS Glue Data Catalog に公開されている場合、データ共有を AWS アカウント全体に関連付けることしかできません。
   - 1 つまたは複数の AWS リージョンと特定のクラスター名前空間を、データ共有と関連付ける場合は、[Specific AWS Regions and cluster namespaces] (特定の Amazon リージョンとクラスター名前空間) をクリックします。
     a. 特定の AWS リージョンとクラスター名前空間をデータ共有に追加するには、[Add Region] (リージョンを追加) を選択します。[Add AWS Region] (Amazon リージョンの追加) ページが表示されます。
     b. [AWS Region] (Amazon リージョン) をクリックします。
     c. 次のいずれかを行ってください。
        - このリージョン内に既存、および将来作成されるすべてのクラスター名前空間をデータ共有に追加するには、[Add all cluster namespaces] (すべてのクラスター名前空間を追加) をクリックします。
        - このリージョンにある 1 つ以上の特定のクラスター名前空間をデータ共有に追加するには、[Add specific cluster namespaces] (特定のクラスター名前空間を追加) をクリックします。
        - 1 つまたは複数のクラスター名前空間を選択し、[Add AWS Region] (Amazon リージョンを追加) をクリックします。
     d. [関連付ける] を選択します。

データ共有を Lake Formation アカウントに関連付ける場合は、Lake Formation コンソールに移動してデータベースを作成し、データベースに対するアクセス許可を定義します。詳細について

は、AWS Lake Formation デベロッパーガイドの「Amazon Redshift データ共有に対するアクセス許可の設定」を参照してください。AWS Glue データベースまたはフェデレーションデータベースを作成したら、クエリエディタ v2 または任意の優先 SQL クライアントをコンシューマークラスターで使用して、データをクエリできます。詳細については、「コンシューマーとしての Lake Formation 管理のデータ共有を使用する」を参照してください。

データ共有の関連付けが完了すると、データを共有することが可能になります。

データ共有の関連付けはいつでも変更できます。関連付けを AWS リージョンやクラスター名前空間から AWS アカウント全体に変更すると、Amazon Redshift は、特定のリージョンおよびクラスター名前空間に関する情報を、AWSカウントの情報で上書きします。これにより、AWSアカウントにあるすべての AWS リージョンとクラスター名前空間が、データ共有にアクセスできるようになります。

関連付け先を、特定のクラスター名前空間から、指定された AWS リージョン内のすべてのクラスター名前空間に変更した場合は、このリージョンにあるすべてのクラスター名前空間が、データ共有にアクセスできます。

データコンシューマーからのデータ共有の関連付けの削除

コンシューマークラスターの管理者は、データコンシューマーからデータ共有の関連付けを削除できます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。データ共有リストページが表示されます。
3. [From other accounts (その他のアカウント)] を選択します。
4. [Datashares from other accounts (他のアカウントからのデータ共有)] セクションでデータ共有を選択して、データコンシューマーから関連付けを削除します。
5. [Data consumers (データコンシューマー)] セクションで、関連付けを削除する 1 つ以上のデータコンシューマーを選択します。次に、[Remove association (関連付けの削除)] を選択します。
6. [Remove association (関連付けの削除)] ページが表示されたら、[Remove association (関連付けの削除)] を選択します。

関連付けが削除されると、データコンシューマーはデータ共有にアクセスできなくなります。データコンシューマーの関連付けはいつでも変更できます。

データ共有の拒否

コンシューマークラスター管理者は、状態が「[使用可能またはアクティブ](#)」のデータ共有を拒否できます。データ共有を拒否した場合、コンシューマークラスターのユーザーはデータ共有へのアクセス権を失います。`DescribeDataSharesForConsumer` API 操作を呼び出しても、Amazon Redshift は拒否されたデータ共有を返しません。プロデューサークラスター管理者が `DescribeDataSharesForProducer` API 操作を実行すると、データ共有が拒否されたことがわかります。データ共有が拒否されると、プロデューサークラスター管理者はコンシューマークラスターにデータ共有を再び承認でき、コンシューマークラスター管理者は自分の AWS アカウントをデータ共有に関連付けるか拒否するかを選択できます。

AWS アカウントにデータ共有との関連付けがあり、Lake Formation が管理するデータ共有との関連付けが保留中の場合、Lake Formation が管理するデータ共有の関連付けを拒否すると、元のデータ共有も拒否されます。特定の関連付けを拒否するには、プロデューサークラスター管理者は、指定したデータ共有から承認を削除できます。このアクションは他のデータ共有には影響しません。

データ共有を拒否するには、AWS コンソール、API 操作 `RejectDataShare`、または AWS CLI で `reject-datashare` を使用します。

AWS コンソールを使用してデータ共有を拒否するには:

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [データ共有] を選択します。
3. [From other accounts (その他のアカウント)] を選択します。
4. [Datashares from other accounts (他のアカウントからのデータ共有)] セクションで、拒否するデータ共有を選択します。[Decline datashare] (データ共有の拒否) ページが表示されたら、[Decline] (拒否) をクリックします。

データ共有を拒否した後は、変更を元に戻すことはできません。Amazon Redshift は、リストからデータ共有を削除します。データ共有を再び表示するには、プロデューサー管理者がデータ共有を再び承認する必要があります。

既存のデータ共有の管理

Amazon Redshift では、既存のデータ共有を管理して、Amazon Redshift クラスター内のデータへのアクセスを制御できます。以下のセクションでは、Amazon Redshift 環境でのデータ共有の管理に関するステップバイステップのガイダンスを提供します。

データ共有の表示

[DATASHARES (データ共有)] または [CLUSTERS (クラスター)] タブからデータ共有を表示します。

- [DATASHARES (データ共有)] タブを使用して、自分のアカウントまたは他のアカウントのデータ共有を一覧表示します。
  - ご使用のアカウントで作成されたデータ共有を表示するには、[In my account (マイアカウントで)] を選択してから、表示するデータ共有を選択します。
  - 他のアカウントから共有されているデータ共有を表示するには、[From other accounts (他のアカウントから)] を選択してから、表示するデータ共有を選択します。
- [CLUSTERS (クラスター)] タブを使用して、クラスター内または他のクラスターからのデータ共有を一覧表示します。

  データベースに接続します。詳細については、「データベースへの接続」を参照してください。

  次に、[Datashares from other clusters (他のクラスターからのデータ共有)] または [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションからデータ共有を選択して、詳細を表示します。

データ共有からのデータ共有オブジェクトの削除

次の手順を使用して、データ共有から 1 つ以上のオブジェクトを削除できます。

データ共有から 1 つ以上のオブジェクトを削除するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。
4. [Datashares created in my account (アカウントで作成されたデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続」を参照してください。
5. 編集するデータ共有を選択して [Edit (編集)] を選択します。データ共有の詳細ページが表示されます。
6. データ共有から 1 つ以上のデータ共有オブジェクトを削除するには、次のいずれかを実行します。

- データ共有からスキーマを削除するには、1 つまたは複数のスキーマを選択します。次に、[Remove (削除)] を選択します。Amazon Redshift は、指定されたスキーマと指定されたスキーマのすべてのオブジェクトをデータ共有から削除します。
- データ共有からテーブルとビューを削除するには、1 つまたは複数のテーブルとビューを選択します。次に、[Remove (削除)] を選択します。または、[Remove by schema (スキーマで削除)] を選択して、指定したスキーマのすべてのテーブルとビューを削除します。
- データ共有からユーザー定義関数を削除するには、1 つまたは複数のユーザー定義関数を選択します。次に、[Remove (削除)] を選択します。または、[Remove by schema (スキーマで削除)] を選択して、指定したスキーマ内のすべてのユーザー定義関数を削除します。

データ共有からのデータコンシューマーの削除

データ共有から 1 つ以上のデータコンコンシューマーを削除できます。データコンシューマーは、Amazon Redshift クラスターまたは AWS アカウントを一意に識別する、クラスター名前空間として使用することができます。

クラスター名前空間の ID または AWS アカウントから 1 つ以上のデータコンシューマーを選択し、[削除] をクリックします。

Amazon Redshift は、指定されたデータコンシューマーをデータ共有から削除します。すぐにデータ共有へのアクセスが失われます。

アカウントで作成されたデータ共有の編集

コンソールを使用して、アカウントで作成されたデータ共有を編集します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。
4. [Datashares created in my account (アカウントで作成されたデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続」を参照してください。
5. 編集するデータ共有を選択して [Edit (編集)] を選択します。データ共有の詳細ページが表示されます。

6. [Datashare objects (データ共有オブジェクト)] または [Data consumers (データコンシューマー)] セクションで変更を加えます。

> **Note**
>
> データ共有を AWS Glue Data Catalog に公開することを選択した場合、そのデータ共有を他の Amazon Redshift アカウントに公開するように設定を編集することはできません。

7. [変更を保存] をクリックします。

Amazon Redshift は、変更を加えてデータ共有を更新します。

アカウント内で作成された データ共有の削除

コンソールを使用して、アカウントで作成されたデータ共有を削除します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。データ共有リストが表示されます。
4. [Datashares created in my account (アカウントで作成されたデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続」を参照してください。
5. 削除するデータ共有を 1 つ以上選択してから、[Delete (削除)] を選択します。[Delete datashares (データ共有の削除)] ページが表示されます。

   Lake Formation と共有されているデータ共有を削除しても、Lake Formation 内の関連付けられているアクセス許可は自動的に削除されません。それらを削除するには、Lake Formation コンソールにアクセスしてください。
6. [Delete (削除)] と入力して、指定したデータ共有の削除を確認します。
7. [削除] を選択します。

データ共有が削除されると、データ共有のコンシューマーはデータ共有にアクセスできなくなります。

データ共有のクエリ

Amazon Redshift では、データ共有間のデータクエリプロデューサークラスターから実行して、ライブデータをコピーまたは転送することなく安全にデータにアクセスできます。以下のセクションでは、Amazon Redshift 環境でのデータ共有の設定とクエリの詳細について説明します。

データ共有からのデータベースの作成

データ共有内のデータのクエリを開始するには、データ共有からデータベースを作成します。指定したデータ共有から作成できるデータベースは 1 つだけです。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。データ共有リストが表示されます。
4. [Datashares from other clusters (他のクラスターからのデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続」を参照してください。
5. データベースを作成するデータ共有を選択してから、[Create database from datashare (データ共有からデータベースを作成)] を選択します。[Create database from datashare (データ共有からデータベースを作成)] ページが表示されます。
6. [Database name (データベース名)] にデータベース名を指定します データベース名は 1～64 文字の英数字 (小文字のみ) にする必要があり、予約語は使用できません。
7. [Create] (作成) を選択します。

データベースが作成されたら、データベース内のデータをクエリできます。

AWS Data Exchange データ共有の管理

Amazon Redshift では、データ抽出やパイプラインを作成および管理することなく、AWS Data Exchange からライブデータを安全に共有および受信できます。AWS Data Exchange データ共有を管理することで、サードパーティーのデータ製品をサブスクライブして、ライブデータストリームを Amazon Redshift データウェアハウスに直接統合できるようになります。以下のセクションでは、Amazon Redshift クラスター内の AWS Data Exchange データ共有の管理について説明します。

AWS Data Exchange でのデータセットの作成

AWS Data Exchange でデータセットを作成します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。
4. [Datashares created in my account] (アカウントで作成されたデータ共有) セクションで、AWS Data Exchangeデータ共有を選択します。
5. [Create data set on AWS Data Exchange] (Amazon Web Services Data Exchange にデータセットを作成) をクリックします。詳細については、「Publishing a new product」を参照してください。

AWS Data Exchange データ共有の編集

コンソールを使用して AWS Data Exchange データ共有を編集します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

AWS Data Exchange データ共有では、データコンシューマーに変更を加えることはできません。

クエリエディタ v2 を使用して、AWS Data Exchangeデータ共有のパブリックにアクセス可能な設定を編集するには、Amazon Redshift は、1 回限り有効なランダム値を生成して、この設定を無効化できるようにセッション変数を設定します。詳細については、「ALTER DATASHARE の使用に関する注意事項」を参照してください。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. ナビゲーターメニューで、[Editor] (エディタ)、[Query Editor v2] (クエリエディタ v2) の順に選択します。
4. クエリエディタ v2 の使用が初めての場合は、AWS アカウントを設定してください。デフォルトでは、AWS 所有キーは、リソースを暗号化するために使用されます。AWS アカウント の設定の詳細については、「Amazon Redshift 管理ガイド」の「AWS アカウント の設定」を参照してください。

5. AWS Data Exchange データ共有が含まれているクラスターに接続するには、[データベース] をクリックした後、ツリービューパネルでクラスター名を指定します。プロンプトが表示されたら、接続パラメータを入力します。
6. 次の SQL 文をコピーします。次の例では、SalesShare データ共有のパブリックにアクセス可能な設定を変更します。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

7. コピーした SQL 文を実行するには、[Queries] (クエリ) をクリックし、対象の SQL 文をクエリ領域に貼り付けます。その後、[Run] (実行) をクリックします。

   エラー表示は次のようになります。

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

   値 'c670ba4db22f4b' は、推奨されないオペレーションが発生した際に、Amazon Redshift が生成するランダムで 1 回限り有効な値です。

8. 以下のサンプル文をコピーし、クエリ領域に貼り付けます。その後に、コマンドを実行します。この `SET datashare_break_glass_session_var` コマンドでは、AWS Data Exchangeのデータ共有で推奨されていないオペレーションに対するアクセス許可が付与されます。

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

9. ALTER DATASHARE ステートメントを再度実行します。

```
ALTER DATASHARE salesshare;
```

Amazon Redshift は、変更を加えてデータ共有を更新します。

アカウント内で作成された AWS Data Exchange データ共有の削除

コンソールを使用して、アカウントで作成された AWS Data Exchange データ共有を削除します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。

2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. ナビゲーターメニューで、[Editor] (エディタ)、[Query Editor v2] (クエリエディタ v2) の順に選択します。
4. クエリエディタ v2 の使用が初めての場合は、AWS アカウントを設定してください。デフォルトでは、AWS 所有キーは、リソースを暗号化するために使用されます。AWS アカウント の設定の詳細については、「Amazon Redshift 管理ガイド」の「AWS アカウント の設定」を参照してください。
5. AWS Data Exchange データ共有が含まれているクラスターに接続するには、[データベース] をクリックした後、ツリービューパネルでクラスター名を指定します。プロンプトが表示されたら、接続パラメータを入力します。
6. 次の SQL 文をコピーします。次の例では、データ共有 SalesShare を削除します。

```
DROP DATASHARE salesshare
```

7. コピーした SQL 文を実行するには、[Queries] (クエリ) をクリックし、対象の SQL 文をクエリ領域に貼り付けます。その後、[Run] (実行) をクリックします。

   エラー表示は次のようになります。

```
ERROR:  Drop of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value '620c871f890c49'
```

   値 '620c871f890c49' は、推奨されないオペレーションが発生した際に、Amazon Redshift が生成するランダムで 1 回限り有効な値です。

8. 以下のサンプル文をコピーし、クエリ領域に貼り付けます。その後に、コマンドを実行します。この `SET datashare_break_glass_session_var` コマンドでは、AWS Data Exchangeのデータ共有で推奨されていないオペレーションに対するアクセス許可が付与されます。

```
SET datashare_break_glass_session_var to '620c871f890c49';
```

9. DROP DATASHARE ステートメントを再度実行します。

```
DROP DATASHARE salesshare;
```

データ共有の削除後は、そのデータ共有のコンシューマーはデータ共有にアクセスできなくなります。

共有されている AWS Data Exchange データ共有を削除することは、AWS Data Exchangeでのデータ製品の使用条件に違反する可能性があります。

AWS CloudFormation によるデータ共有の管理

AWS CloudFormation スタックに AWS リソースのプロビジョニングを実行させせることで、データ共有の設定を自動化できます。CloudFormation スタックは、同じ AWS アカウント内にある 2 つの Amazon Redshift クラスター間でのデータ共有を設定します。したがって、リソースをプロビジョニングする SQL ステートメントを実行することなく、データ共有を開始できます。

このスタックは、指定したクラスター上にデータ共有を作成します。作成されたデータ共有には、テーブルとサンプルの読み取り専用データが含まれています。このデータは、別の Amazon Redshift Redshift クラスターから読み取ることができます。

CloudFormation を使用せずに SQL ステートメントを実行してデータ共有を設定し、アクセス許可を付与しながら AWS アカウント内でデータ共有を開始する場合は、「AWS アカウント 内のデータへの読み取りアクセスの共有」を参照してください。

データ共有 CloudFormation スタックを実行する前に、IAM ロールと Lambda 関数を作成する許可を持つユーザーとしてログインしておく必要があります。また、同じアカウントに 2 つの Amazon Redshift クラスターが必要です。それらの内の 1 つ producer はサンプルデータの共有に使用し、もう一方の consumer はデータの読み取りに使用します。これらのクラスターに対する主な要件は、それぞれが RA3 ノードを使用していることです。追加の要件については、「Amazon Redshift でデータ共有を使用する際の考慮事項」を参照してください。

Amazon Redshift クラスターの設定の開始方法については、「Amazon Redshift のプロビジョニングされたデータウェアハウス」を参照してください。CloudFormation による設定の自動化の詳細については、「AWS CloudFormation とは」を参照してください。

> **Important**
>
> CloudFormation スタックを起動する前に、同じアカウント内に 2 つの Amazon Redshift クラスターが存在し、クラスターが RA3 ノードを使用することを確認します。各クラスターにデータベースとスーパーユーザーが存在することを確認します。詳細については、CREATE DATABASEおよびsuperuserを参照してください。

Amazon Redshift のデータ共有のために CloudFormation スタックを起動するには

1. [Launch CFN stack] (CFN スタックを起動する) をクリックし、AWS Management Console内の CloudFormation サービスを開きます。

   プロンプトが表示されたら、サインインします。

   Amazon S3 に保存されている CloudFormation テンプレートファイルを参照しながら、スタック作成プロセスが開始されます。CloudFormation テンプレートは、JSON 形式で作成されたテキストファイルで、スタックを構成する AWS リソースに関する宣言が記述されています。CloudFormation テンプレートの詳細については、「テンプレートの基礎についての学習」を参照してください。
2. [Next] (次へ) をクリックして、スタックの詳細を入力します。
3. [Parameters] (パラメータ) で、クラスターごとに次のように入力します。

   - Amazon Redshift のクラスター名 (例えば、**ra3-consumer-cluster**)
   - データベース名 (例えば、**dev**)
   - データベースのユーザー名 (例えば、**consumeruser**)

   スタックは複数のデータベースオブジェクトを作成するため、テスト用クラスターの使用が推奨されます。

   [Next] を選択します。
4. スタックに関するオプションが表示されます。

   [Next] (次へ) をクリックして、デフォルト設定を受け入れます。
5. [機能] で、[AWS CloudFormation によって IAM リソースが作成される場合があることを承認します] を選択します。
6. [スタックの作成] を選択します。

CloudFormation は、テンプレートを使用しながら 10 分程度で Amazon Redshift スタックを構築し、`myproducer_share` というデータ共有を作成します。スタックは、スタックの詳細で指定されたデータベースにデータ共有を作成します。このデータベースにあるオブジェクトのみが共有できます。

スタックの作成中にエラーが発生した場合は、以下の手順を実行します。

- 各 Redshift クラスターに、クラスター名、データベース名、およびデータベースのユーザー名を正しく入力していることを確認します。
- クラスターに RA3 ノードがあることを確認します。
- IAM ロールと Lambda 関数を作成するアクセス許可を持っているユーザーとしてログインしていることを確認します。IAM ロールの作成についての詳細は、「IAM ロールの作成」を参照してください。Λ関数作成のポリシーの詳細については、「関数の開発」を参照してください。

作成したデータ共有に対するクエリ

以下の手順を使用するには、説明で示されている各クラスターでクエリを実行するための、アクセス許可が必要です。

データ共有に対しクエリするには

1. CloudFormation スタックの作成時に入力したデータベース上のプロデューサークラスターに、Amazon Redshift クエリエディタ v2 などのクライアントツールを使用して接続します。
2. データ共有へのクエリを実行します。

```
SHOW DATASHARES;

+------------------+-------------+-----------------+-------------------
+------------+------------+---------------------+-----------+------------------
+---------------------------------------+
|    share_name     | share_owner | source_database | consumer_database | share_type
 | createdate | is_publicaccessible | share_acl | producer_account |
 producer_namespace                    |
+------------------+-------------+-----------------+-------------------
+------------+------------+---------------------+-----------+------------------
+---------------------------------------+
| myproducer_share | 100          | sample_data_dev | myconsumer_db      | INBOUND
  | NULL        | true                     | NULL       | producer-acct     |         your-
producer-namespace         |
+------------------+-------------+-----------------+-------------------
+------------+------------+---------------------+-----------+------------------
+---------------------------------------+
```

前述のコマンドでは、スタックによって作成されたデータ共有の名前 (myproducer_share) を返します。同時に、データ共有 myconsumer_db に関連付けられたデータベースの名前も返します。

後述のステップで使用するために、プロデューサーの名前空間識別子をコピーします。

3. データ共有内のオブジェクトの詳細を表示します。

```
DESC DATASHARE myproducer_share;

+------------------+---------------------------------------+------------
+------------------+-------------+-----------------------------------
+-------------+
| producer_account |          producer_namespace           | share_type |
 share_name     | object_type |                object_name              | include_new |
+------------------+---------------------------------------+------------
+------------------+-------------+-----------------------------------
+-------------+
|   producer-acct  |        your-producer-namespace        | OUTBOUND   |
 myproducer_share | schema       | myproducer_schema                      | true
 |
|   producer-acct  |        your-producer-namespace        | OUTBOUND   |
 myproducer_share | table        | myproducer_schema.tickit_sales         | NULL
 |
|   producer-acct  |        your-producer-namespace        | OUTBOUND   |
 myproducer_share | view         | myproducer_schema.ticket_sales_view | NULL
 |
+------------------+---------------------------------------+------------
+------------------+-------------+-----------------------------------
+-------------+
```

データ共有の詳細を表示すると、テーブルとビューに関するプロパティが出力されます。スタックは、サンプルデータを含むテーブルとビュー (例えば tickit_sales と tickit_sales_view) を、プロデューサーデータベースに追加します。TICKIT サンプルデータの詳細については、「サンプルデータベース」を参照してください。

クエリを実行するために、データ共有に対するアクセス許可を委任する必要はありません。スタックが必要な許可を付与します。

4. クライアントツールを使用してコンシューマークラスターに接続します。プロデューサの名前空間を指定しながら、データ共有の詳細を表示します。

```
DESC DATASHARE myproducer_share OF NAMESPACE '<namespace id>'; --specify the unique
 identifier for the producer namespace
```

```
+-------------------+---------------------------------------+------------
+-------------------+-------------+-----------------------------------
+-------------+
| producer_account |          producer_namespace           | share_type |
 share_name      | object_type |              object_name              | include_new |
+-------------------+---------------------------------------+------------
+-------------------+-------------+-----------------------------------
+-------------+
|   producer-acct  |        your-producer-namespace        | INBOUND    |
 myproducer_share | schema      | myproducer_schema                     | NULL
 |
|   producer-acct  |        your-producer-namespace        | INBOUND    |
 myproducer_share | table       | myproducer_schema.tickit_sales        | NULL
 |
|   producer-acct  |        your-producer-namespace        | INBOUND    |
 myproducer_share | view        | myproducer_schema.ticket_sales_view | NULL
 |
+-------------------+---------------------------------------+------------
+-------------------+-------------+-----------------------------------
+-------------+
```

5. データ共有のデータベースとスキーマを指定して、データ共有内のテーブルに対しクエリを実行できます。詳細については、「クロスデータベースクエリの例」を参照してください。次のクエリは、TICKIT サンプルデータベースの SALES テーブルから売上と販売者のデータを返します。詳細については、「サンプルデータベース」を参照してください。

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales_view;

+----------+---------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+----------+---------+----------+---------+---------+--------+---------+-----------
+------------+---------------------+
|       1 |      1 |    36861 |   21191 |    7872 |   1875 |       4 |       728 |
    109.2 | 2008-02-18 02:36:48 |
|       2 |      4 |     8117 |   11498 |    4337 |   1983 |       2 |        76 |
     11.4 | 2008-06-06 05:00:16 |
|       3 |      5 |     1616 |   17433 |    8647 |   1983 |       2 |       350 |
     52.5 | 2008-06-06 08:26:17 |
|       4 |      5 |     1616 |   19715 |    8647 |   1986 |       1 |       175 |
    26.25 | 2008-06-09 08:38:52 |
```

```
|       5 |       6 |     47402 |     14115 |     8240 |    2069 |        2 |        154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+--------------------+
```

> **Note**
>
> クエリは、共有スキーマ内のビューに対して実行されます。データ共有から作成されたデータベースに直接接続することはできません。これらのデータベースは読み取り専用です。

6. 集計を含むクエリを実行するには、次の例を使用します。

```
SELECT * FROM myconsumer_db.myproducer_schema.tickit_sales ORDER BY 1,2 LIMIT 5;

+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+--------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+--------------------+
|       1 |       1 |     36861 |     21191 |     7872 |    1875 |        4 |        728 |
     109.2 | 2008-02-18 02:36:48 |
|       2 |       4 |      8117 |     11498 |     4337 |    1983 |        2 |         76 |
      11.4 | 2008-06-06 05:00:16 |
|       3 |       5 |      1616 |     17433 |     8647 |    1983 |        2 |        350 |
      52.5 | 2008-06-06 08:26:17 |
|       4 |       5 |      1616 |     19715 |     8647 |    1986 |        1 |        175 |
     26.25 | 2008-06-09 08:38:52 |
|       5 |       6 |     47402 |     14115 |     8240 |    2069 |        2 |        154 |
      23.1 | 2008-08-31 09:17:02 |
+---------+--------+----------+---------+---------+--------+---------+-----------
+------------+--------------------+
```

このクエリは、サンプルの TICKIT データから売上データと出品者のデータを返します。

データ共有に対するクエリの他の例については、「AWS アカウント 内のデータへの読み取りアクセスの共有」を参照してください。

コンソールを使った書き込みを伴うデータ共有の管理 (プレビュー)

これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、書き込みを伴うデータ共有をコンソールを使って管理することで、Amazon Redshift クラスターおよび AWS アカウント間のデータへのアクセスと管理を制御できます。以下のセクションでは、Amazon Redshift コンソールを使用して書き込みを伴うデータ共有を設定および管理するためのステップバイステップの手順について説明します。

PREVIEW_2023 トラックのセットアップの詳細については、以下の資料のいずれかを参照してください。

- Amazon Redshift Serverless プレビューについては:「プレビューワークグループの作成」
- Amazon Redshift プロビジョニングクラスターのプレビューについては:「プレビュークラスターの作成」

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

Amazon Redshift コンソールを使用して、自分のアカウントで作成された、または他のアカウントから共有されたデータ共有を管理します。

データベースへの接続 (プレビュー)

これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、データ共有を使用して、AWS アカウントおよび Amazon Redshift クラスターのデータベースに接続できます。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

データベースに接続して、このクラスター内のデータベースおよびデータベース内のオブジェクトを表示したり、データ共有を表示したりします。

指定されたデータベースへの接続に使用されるユーザー認証情報には、すべてのデータ共有を表示するために必要となる許可が必要です。

ローカル接続がない場合は、次のいずれかの操作を行います。

- クラスターの詳細ページの [Databases] (データベース) タブにある [Databases] (データベース) または [Datashare objects] (データ共有オブジェクト) セクションで、[Connect to database] (データベースに接続) をクリックし、クラスター内のデータベースオブジェクトを表示します。
- [cluster details (クラスターの詳細)] ページの [Datashares (データ共有)] タブで、次のいずれかを実行します。
  - [Datashares from other clusters (他のクラスターからのデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択して、他のクラスターからのデータ共有を表示します。
  - [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択して、クラスター内のデータ共有を表示します。
- [Connect to database (データベースに接続)] ウィンドウで、次のいずれかを実行します。
  - [新しい接続を作成] を選択した場合は、[AWS Secrets Manager] を選択し、保存されているシークレットを使用して接続へのアクセスを認証します。

    または、データベースの認証情報を使用して接続へのアクセスを認証するには、[Temporary credentials (一時的な認証情報)] を選択します。[Database name (データベース名)] と [Database user (データベースユーザー)] の値を指定します。

    [接続] を選択します。
  - [Use a recent connection (最近の接続を使用する)] を選択して、必要な許可を持つ別のデータベースに接続します。

    Amazon Redshift が自動的に接続を確立します。

データベース接続が確立されたら、データ共有の作成、データ共有のクエリ、またはデータ共有からのデータベースの作成を開始できます。

データ共有の作成とオブジェクトの追加 (プレビュー)

データ共有の作成

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、データ共有を使用して、Amazon Redshift クラスターまたは AWS アカウント間でライブデータを共有することができます。データ共有は、Amazon Redshift クラスターからのライブデータを他のクラスターまたは AWS アカウントと共有できるようにするコンシューマープロデューサーオブジェクトです。データ共有を作成すると、安全なデータ共有が可能になり、アクセスの制御を維持し、データが最新の状態であることを維持できます。以下のセクションでは、データ共有の作成と、ライブデータを安全に共有するためのスキーマ、テーブル、ビューなどのデータベースオブジェクトの追加について詳しく説明します。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

プロデューサークラスターの管理者は、[cluster details (クラスターの詳細)] ページの [Database (データベース)] タブまたは [Datashares (データ共有)] タブからデータ共有を作成できます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Cluster Details (クラスターの詳細)] ページで次の操作を実行します。
   - [Databases (データベース)] タブの [Database (データベース)] セクションで、データベースを選択します。データベースの詳細ページが表示されます。

     [データ共有を作成] を選択します。データ共有はローカルデータベースからのみ作成できます。データベースに接続していない場合は、[データベースに接続] ページが表示されます。データベースへの接続 (プレビュー) の手順に従って、データベースに接続します。最近接続したことがある場合は、[データ共有を作成] ページが表示されます。

- データベース接続がない場合は、[Datashares (データ共有)] タブの [Datashares (データ共有)] セクションで、データベースに接続します。

  [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションで、[Create datashare (データ共有の作成)] を選択します。[データ共有を作成] ページが表示されます。

4. ここから、さまざまなタイプのデータベースオブジェクトを追加できます。[追加] ボタンを選択してオブジェクトを追加します。ダイアログが表示されます。以下のステップを実行します。

   1. 単一のスキーマ、または複数のスキーマを選択します。これにより、スキーマのオブジェクトを追加できるようになります。
   2. スキーマから [オブジェクトタイプ] を選択します。

      ここから、[オブジェクトを追加] するオプションをいくつか選択できます。

      - [スキーマから特定のオブジェクトを追加] — これを選択すると、個々のオブジェクトが名前別に一覧表示されます。オブジェクトを選択してデータ共有に追加できます。例えば、必要に応じて特定のテーブルやストアドプロシージャを追加できます。これで、選択したスキーマのテーブルとストアドプロシージャがデータ共有に含まれます。アクセス許可の設定については、以降の手順で詳しく説明します。[ビュー] やその他のタイプに進み、追加するオブジェクトを選択します。
      - [選択したオブジェクトタイプの既存オブジェクトをすべてスキーマに追加] — これによりすべてのオブジェクトが追加されます。
   3. [今後のオブジェクトを追加] するかどうかも選択できます。スキーマに追加されたデータ共有オブジェクトを含めることを選択すると、スキーマに追加されたオブジェクトが自動的にデータ共有に追加されます。
   4. [追加] を選択してセクションを完成させ、オブジェクトを追加します。これらは [データ共有オブジェクト] の下に一覧表示されます。
   5. オブジェクトを追加したら、個々のオブジェクトを選択し、そのアクセス許可を編集できます。スキーマを選択すると、スコープ設定アクセス許可を追加するかどうかを尋ねるダイアログが表示されます。これにより、既存の各オブジェクト、またはスキーマに追加された各オブジェクトに、オブジェクトタイプに適したアクセス許可のセットがあらかじめ選択されるようになります。例えば、管理者は、追加されたすべてのテーブルに SELECT アクセス許可と UPDATE アクセス許可を付与するように設定できます。
   6. スキーマのアクセス許可を設定したら、他のオブジェクトタイプを順を追って確認し、そのアクセス許可を選択できます。例えば、特定のテーブルに UPDATE アクセス許可を追加できます。

7. データコンシューマーセクションでは、名前空間を追加したり、データ共有のコンシューマーとして AWS アカウントを追加したりできます。
8. [データ共有を作成] を選択して変更を保存します。

データ共有を作成すると、そのデータ共有は [名前空間で作成されたデータ共有] の下のリストに表示されます。リストからデータ共有を選択すると、そのコンシューマー、オブジェクト、その他のプロパティを表示できます。

データ共有へのデータコンシューマーの追加

1 つ以上のデータコンシューマーをデータ共有に追加できます。データコンシューマーは、一意に識別された Amazon Redshift クラスターであるクラスター名前空間、または AWS アカウントにすることができます。

公開アクセスが設定されているクラスターへのデータ共有は、無効にするか有効にするかを明示的に選択する必要があります。

- [Add cluster namespaces to the datashare (クラスター名前空間をデータ共有に追加)] を選択します。名前空間は Amazon Redshift クラスターのグローバル一意識別子 (GUID) です。
- [Add AWS アカウント] (追加) を選択して、データ共有します。指定された AWS アカウントには、データ共有へのアクセス許可がある必要があります。

データ共有からの認可または認可の削除 (プレビュー)

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、指定されたコンシューマーの承認を付与または削除することで、データ共有へのアクセスを制御できます。データ共有を使用すると、同じまたは異なる AWS アカウントの Amazon Redshiftt クラスター間でライブデータを共有できるため、分析データをシームレスに分散および消費できます。以下のセクションでは、Amazon Redshift のデータ共有へのコンシューマーアクセス許可を付与または削除するための手順をステップバイステップで説明します。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

プロデューサークラスターの管理者として、データ共有へのアクセスを許可するデータコンシューマー、または許可を削除するデータコンシューマーを選択します。認可されたデータコンシューマーは、データ共有でアクションを実行するための通知を受け取ります。クラスター名前空間をデータコンシューマーとして追加する場合は、承認を実行する必要はありません。

前提条件: データ共有の認可を承認または削除するには、データ共有に少なくとも 1 つのデータコンシューマーが追加されている必要があります。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。ここには、[データ共有コンシューマー] というリストが表示されます。認可するコンシューマークラスターを 1 つ以上選択します。次に、[承認] を選択します。
3. [アカウントを承認] ダイアログが表示されます。いくつかの承認タイプから選択できます。
    - [クラスター名またはワークグループ名] で読み取り専用 — データ共有作成者が書き込みアクセス許可を付与したとしても、コンシューマーには書き込みアクセス許可がないということです。
    - [クラスター名またはワークグループ名] への読み取りと書き込み — 作成者が付与したすべてのアクセス許可 (書き込みアクセス許可を含む) をコンシューマーでも利用できるということです。
4. [Save] を選択します。

コンシューマーとして AWS Data Exchange を承認することもできます。

1. データ共有の作成時に [AWS Glue Data Catalog に発行] を選択した場合、データ共有の許可は Lake Formation アカウントにのみ付与できます。

    AWS Data Exchange データ共有の場合、一度に 1 つのデータ共有しか認可できません。

    AWS Data Exchange データ共有を承認すると、データ共有を AWS Data Exchange サービスと共有し、ユーザーに代わって AWS Data Exchange がデータ共有へのアクセスを管理することを許可することになります。AWS Data Exchange は、コンシューマーが製品をサブスクライブする際、コンシューマーアカウントを AWS Data Exchange データ共有のデータコンシューマーと

して追加することで、コンシューマーがアクセスできるようにします。AWS Data Exchange には、データ共有への読み取りアクセス許可はありません。

2. [Save] を選択します。

データコンシューマーが認可されると、データ共有オブジェクトにアクセスし、データをクエリするコンシューマデータベースを作成できます。

承認の削除:

認可を削除するコンシューマークラスターを 1 つ以上選択します。次に、[Remove authorization (認可の削除)] を選択します。

認可が削除された直後から、データコンシューマーはデータ共有にアクセスできなくなります。

コンシューマーとしてのデータ共有の関連付けまたは拒否 (プレビュー)

Amazon Redshift では、他の AWS アカウントと共有されているデータ共有を関連付けたり拒否したりできるため、組織の境界を越えてシームレスで安全なデータ共有が可能になります。データ共有は、1 つ以上の Amazon Redshift データベースからのデータをカプセル化する共有可能なデータベースオブジェクトです。以下のセクションでは、Amazon Redshift 環境内の共有データリソースの可能性を最大限に引き出すために、データ共有の関連付けまたは拒否を行うプロセスについて説明します。

データ共有の関連付け

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

コンシューマークラスターの管理者は、他のアカウントから共有されている 1 つ、または複数のデータ共有を、AWSアカウント全体、またはアカウント内の特定のクラスター名前空間に関連付けることができます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。データ共有リストページが表示されます。[From other accounts (その他のアカウント)] を選択します。
3. [Datashares from other accounts] (他のアカウントからのデータ共有) セクションで、関連付けを行うデータ共有を選択し、[Associate] (関連付け) をクリックします。データ共有の [関連付け] ページが表示されたら、以下の関連付けタイプのいずれかを選択します。

   - AWS アカウント内の異なる AWS リージョンにわたって、既存および将来に作成されるクラスター名前空間のすべてをデータ共有に関連付けるには、[Entire AWS account] (Amazon アカウント全体) をクリックします。

     データ共有が AWS Glue Data Catalog に公開されている場合、データ共有を AWS アカウント全体に関連付けることしかできません。
4. ここから [許可されているアクセス許可] を選択できます。選択肢として、次のものがあります。

   - 読み取り専用 — 読み取り専用を選択した場合、UPDATE や INSERT などの書き込みアクセス許可は、プロデューサー側で付与および承認されていたとしても、コンシューマーでは使用できません。
   - 読み取りと書き込み — コンシューマーデータ共有ユーザーは、プロデューサーによって付与および承認されたすべてのアクセス許可 (読み取りと書き込みの両方) を持ちます。
5. または、1 つまたは複数の AWS リージョンと特定のクラスター名前空間を、データ共有と関連付ける場合は、[特定の AWS リージョンとクラスター名前空間] をクリックします。特定の AWS リージョンとクラスター名前空間をデータ共有に追加するには、[Add Region] (リージョンを追加) を選択します。[Add AWS Region] (Amazon リージョンの追加) ページが表示されます。
6. [AWS Region] (Amazon リージョン) をクリックします。
7. 次のいずれかを行ってください。

   - このリージョン内に既存、および将来作成されるすべてのクラスター名前空間をデータ共有に追加するには、[Add all cluster namespaces] (すべてのクラスター名前空間を追加) をクリックします。
   - このリージョンにある 1 つ以上の特定のクラスター名前空間をデータ共有に追加するには、[Add specific cluster namespaces] (特定のクラスター名前空間を追加) をクリックします。

- 1 つまたは複数のクラスター名前空間を選択し、[Add AWS Region] (Amazon リージョンを追加) をクリックします。

8. [関連付ける] を選択します。

プロデューサーが戻って承認の設定を変更することができますが、この場合、コンシューマーの関連付け設定に影響する可能性があります。

データ共有を Lake Formation アカウントに関連付ける場合は、Lake Formation コンソールに移動してデータベースを作成し、データベースに対するアクセス許可を定義します。詳細については、AWS Lake Formation デベロッパーガイドの「Amazon Redshift データ共有に対するアクセス許可の設定」を参照してください。AWS Glue データベースまたはフェデレーションデータベースを作成したら、クエリエディタ v2 または任意の優先 SQL クライアントをコンシューマークラスターで使用して、データをクエリできます。

データ共有の関連付けが完了すると、データを共有することが可能になります。

データ共有の関連付けはいつでも変更できます。関連付けを AWS リージョンやクラスター名前空間から AWS アカウント全体に変更すると、Amazon Redshift は、特定のリージョンおよびクラスター名前空間に関する情報を、AWSカウントの情報で上書きします。これにより、AWSアカウントにあるすべての AWS リージョンとクラスター名前空間が、データ共有にアクセスできるようになります。

関連付け先を、特定のクラスター名前空間から、指定された AWS リージョン内のすべてのクラスター名前空間に変更した場合は、このリージョンにあるすべてのクラスター名前空間が、データ共有にアクセスできます。

データコンシューマーからのデータ共有の関連付けの削除

コンシューマークラスターの管理者は、データコンシューマーからデータ共有の関連付けを削除できます。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Datashares] (データ共有) を選択します。データ共有リストページが表示されます。
3. [From other accounts (その他のアカウント)] を選択します。
4. [Datashares from other accounts (他のアカウントからのデータ共有)] セクションでデータ共有を選択して、データコンシューマーから関連付けを削除します。

5. [Data consumers (データコンシューマー)] セクションで、関連付けを削除する 1 つ以上のデータコンシューマーを選択します。次に、[Remove association (関連付けの削除)] を選択します。
6. [Remove association (関連付けの削除)] ページが表示されたら、[Remove association (関連付けの削除)] を選択します。

関連付けが削除されると、データコンシューマーはデータ共有にアクセスできなくなります。データコンシューマーの関連付けはいつでも変更できます。

データ共有の拒否

コンシューマークラスター管理者は、状態が「使用可能またはアクティブ」のデータ共有を拒否できます。データ共有を拒否した場合、コンシューマークラスターのユーザーはデータ共有へのアクセス権を失います。`DescribeDataSharesForConsumer` API 操作を呼び出しても、Amazon Redshift は拒否されたデータ共有を返しません。プロデューサークラスター管理者が `DescribeDataSharesForProducer` API 操作を実行すると、データ共有が拒否されたことがわかります。データ共有が拒否されると、プロデューサークラスター管理者はコンシューマークラスターにデータ共有を再び承認でき、コンシューマークラスター管理者は自分の AWS アカウントをデータ共有に関連付けるか拒否するかを選択できます。

AWS アカウントにデータ共有との関連付けがあり、Lake Formation が管理するデータ共有との関連付けが保留中の場合、Lake Formation が管理するデータ共有の関連付けを拒否すると、元のデータ共有も拒否されます。特定の関連付けを拒否するには、プロデューサークラスター管理者は、指定したデータ共有から承認を削除できます。このアクションは他のデータ共有には影響しません。

データ共有を拒否するには、AWS コンソール、API 操作 `RejectDataShare`、または AWS CLI で `reject-datashare` を使用します。

AWS コンソールを使用してデータ共有を拒否するには:

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [データ共有] を選択します。
3. [From other accounts (その他のアカウント)] を選択します。
4. [Datashares from other accounts (他のアカウントからのデータ共有)] セクションで、拒否するデータ共有を選択します。[Decline datashare] (データ共有の拒否) ページが表示されたら、[Decline] (拒否) をクリックします。

データ共有を拒否した後は、変更を元に戻すことはできません。Amazon Redshift は、リストからデータ共有を削除します。データ共有を再び表示するには、プロデューサー管理者がデータ共有を再び承認する必要があります。

既存のデータ共有の管理 (プレビュー)

> これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、既存のデータ共有を管理して、Amazon Redshift クラスター内のデータへのアクセスを制御できます。以下のセクションでは、Amazon Redshift 環境でのデータアクセスを効果的に制御および監査するために、データ共有オブジェクトの変更、データ共有アクセス許可の管理、データ共有プロパティの更新に関するステップバイステップのガイダンスを提供します。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

データ共有の表示

[DATASHARES (データ共有)] または [CLUSTERS (クラスター)] タブからデータ共有を表示します。

- [DATASHARES (データ共有)] タブを使用して、自分のアカウントまたは他のアカウントのデータ共有を一覧表示します。
  - ご使用のアカウントで作成されたデータ共有を表示するには、[In my account (マイアカウントで)] を選択してから、表示するデータ共有を選択します。
  - 他のアカウントから共有されているデータ共有を表示するには、[From other accounts (他のアカウントから)] を選択してから、表示するデータ共有を選択します。
- [CLUSTERS (クラスター)] タブを使用して、クラスター内または他のクラスターからのデータ共有を一覧表示します。

  データベースに接続します。詳細については、「データベースへの接続 (プレビュー)」を参照してください。

次に、[Datashares from other clusters (他のクラスターからのデータ共有)] または [Datashares created in my cluster (クラスターで作成したデータ共有)] セクションからデータ共有を選択して、詳細を表示します。

データ共有からのデータ共有オブジェクトの削除

次の手順を使用して、データ共有から 1 つ以上のオブジェクトを削除できます。

データ共有から 1 つ以上のオブジェクトを削除するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。
4. [Datashares created in my account (アカウントで作成されたデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続 (プレビュー)」を参照してください。
5. 編集するデータ共有を選択して [Edit (編集)] を選択します。データ共有の詳細ページが表示されます。
6. データ共有から 1 つ以上のデータ共有オブジェクトを削除するには、次のいずれかを実行します。

    - データ共有からスキーマを削除するには、1 つまたは複数のスキーマを選択します。次に、[Remove (削除)] を選択します。Amazon Redshift は、指定されたスキーマと指定されたスキーマのすべてのオブジェクトをデータ共有から削除します。
    - データ共有からテーブルとビューを削除するには、1 つまたは複数のテーブルとビューを選択します。次に、[Remove (削除)] を選択します。または、[Remove by schema (スキーマで削除)] を選択して、指定したスキーマのすべてのテーブルとビューを削除します。
    - データ共有からユーザー定義関数を削除するには、1 つまたは複数のユーザー定義関数を選択します。次に、[Remove (削除)] を選択します。または、[Remove by schema (スキーマで削除)] を選択して、指定したスキーマ内のすべてのユーザー定義関数を削除します。

データ共有からのデータコンシューマーの削除

データ共有から 1 つ以上のデータコンコンシューマーを削除できます。データコンシューマーは、Amazon Redshift クラスターまたは AWS アカウントを一意に識別する、クラスター名前空間として使用することができます。

クラスター名前空間の ID または AWS アカウントから 1 つ以上のデータコンシューマーを選択し、[削除] をクリックします。

Amazon Redshift は、指定されたデータコンシューマーをデータ共有から削除します。すぐにデータ共有へのアクセスが失われます。

アカウントで作成されたデータ共有の編集

コンソールを使用して、アカウントで作成されたデータ共有を編集します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。
4. [アカウントで作成されたデータ共有] セクションで、[データベースに接続] を選択します。
5. 編集するデータ共有を選択して [Edit (編集)] を選択します。データ共有の詳細ページが表示されます。
6. [Datashare objects (データ共有オブジェクト)] または [Data consumers (データコンシューマー)] セクションで変更を加えます。

> **Note**
>
> データ共有を AWS Glue Data Catalog に公開することを選択した場合、そのデータ共有を他の Amazon Redshift アカウントに公開するように設定を編集することはできません。

7. [変更を保存] をクリックします。

Amazon Redshift は、変更を加えてデータ共有を更新します。

アカウント内で作成された データ共有の削除

コンソールを使用して、アカウントで作成されたデータ共有を削除します。最初にデータベースに接続して、アカウントで作成されたデータ共有のリストを確認します。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。データ共有リストが表示されます。
4. [アカウントで作成されたデータ共有] セクションで、[データベースに接続] を選択します。
5. 削除するデータ共有を 1 つ以上選択してから、[Delete (削除)] を選択します。[Delete datashares (データ共有の削除)] ページが表示されます。

   Lake Formation と共有されているデータ共有を削除しても、Lake Formation 内の関連付けられているアクセス許可は自動的に削除されません。それらを削除するには、Lake Formation コンソールにアクセスしてください。
6. [Delete (削除)] と入力して、指定したデータ共有の削除を確認します。
7. [削除] を選択します。

データ共有が削除されると、データ共有のコンシューマーはデータ共有にアクセスできなくなります。

データ共有のクエリ (プレビュー)

これは Amazon Redshift のデータ共有機能によるマルチデータウェアハウス書き込みに関するプレリリースドキュメントで、PREVIEW_2023 トラックでパブリックプレビュー版が提供されています。ドキュメントと機能はどちらも変更されることがあります。この特徴は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの契約条件については、AWS のサービス条件の「ベータサービスへの参加」を参照してください。

Amazon Redshift では、データ共有間のデータクエリプロデューサークラスターから実行して、ライブデータをコピーまたは転送することなく安全にデータにアクセスできます。以下のセクションでは、Amazon Redshift 環境でのデータ共有の設定とクエリの詳細について説明します。

データ共有の開始方法の詳細については、「データへの書き込みアクセスの共有 (プレビュー)」を参照してください。

データ共有からのデータベースの作成

データ共有内のデータのクエリを開始するには、データ共有からデータベースを作成します。指定したデータ共有から作成できるデータベースは 1 つだけです。

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Clusters] (クラスター) を選択してから、ご使用のクラスターを選択します。クラスターの詳細ページが表示されます。
3. [Datashares (データ共有)] を選択します。データ共有リストが表示されます。
4. [Datashares from other clusters (他のクラスターからのデータ共有)] セクションで、[Connect to database (データベースに接続)] を選択します。詳細については、「データベースへの接続 (プレビュー)」を参照してください。
5. データベースを作成するデータ共有を選択してから、[Create database from datashare (データ共有からデータベースを作成)] を選択します。[Create database from datashare (データ共有からデータベースを作成)] ページが表示されます。
6. [Database name (データベース名)] にデータベース名を指定します データベース名は 1～64 文字の英数字 (小文字のみ) にする必要があり、予約語は使用できません。
7. [Create] (作成) を選択します。

データベースを作成した後は、コンシューマー管理者によって許可、承認、関連付けがなされている場合は、データベース内のデータのクエリや、書き込み操作の実行ができます。

Amazon Redshift の半構造化データ

Amazon Redshift で半構造化データのサポートを使用することにより、半構造化データを取り込んで Amazon Redshift データウェアハウスに保存できます。Amazon Redshift は、SUPER データ型と PartiQL 言語を使用して、データウェアハウス機能を拡張し、SQL データソースと NoSQL データソースの両方と統合します。このようにして、Amazon Redshift は JSON などのリレーショナルおよび半構造化された保存データの効率的な分析を可能にします。

Amazon Redshift には、SUPER データ型と Amazon Redshift Spectrum という 2 つの形式の半構造化データサポートが用意されています。

低いレイテンシーで JSON データの小さなバッチを挿入または更新する必要がある場合は、SUPER データ型を使用します。また、クエリで強力な一貫性、予測可能なクエリパフォーマンス、複雑なクエリサポート、および進化するスキーマやスキーマレスデータの使いやすさが必要な場合にも SUPER を使用します。

これとは対照的に、他の AWS サービスとの統合や、アーカイブ目的で主に Amazon S3 に保存されているデータとの統合をデータクエリが必要とする場合は、オープンファイル形式で Amazon Redshift Spectrum を使用してください。

SUPER データ型のユースケース

Amazon Redshift の SUPER データ型を使用した半構造化データのサポートは、優れたパフォーマンス、柔軟性、使いやすさを提供します。以下に示すユースケースは、SUPER で半構造化データサポートを使用する方法を示します。

JSON データの迅速かつ柔軟な挿入 – Amazon Redshift は、JSON を解析して SUPER 値として保存できる迅速なトランザクションをサポートします。挿入トランザクションは、従来の列で SUPER の属性をシュレッダー処理したテーブルに同じ挿入を実行するよりも、最大 5 倍高速に動作することができます。例えば、受信 JSON の形式が {“a”:.., “b”:.., “c”“..., ...} であるとします。受信 JSON を列「a」、「b」、「c」などを持つ従来のテーブル TR に保存する代わりに、単一の SUPER 列 S を持つテーブル TJ に保存することで、挿入パフォーマンスを何度も加速できます。JSON に何百もの属性がある場合、SUPER データ型のパフォーマンス上の利点はかなりのものになります。

また、SUPER データ型には通常のスキーマは必要ありません。受信した JSON を保存する前に、確認してクリーンアップする必要はありません。例えば、受信 JSON に文字列「c」属性があり、その他の JSON には整数「c」属性があり、SUPER データ型がないとします。この場合、c_string 列と c_int 列を分離するか、データをクリーンアップする必要があります。これに対し、SUPER デー

タ型では、すべての JSON データが取り込み中に情報を失うことなく保存されます。後で、SQL の PartiQL 拡張機能を使用して情報を分析できます。

検出のための柔軟なクエリ – 半構造化データ (JSON など) を SUPER データ値に保存した後、スキーマを課すことなくクエリを実行できます。PartiQL 動的型付けと lax セマンティクスを使用してクエリを実行し、深くネストされた必要なデータを検出できます。クエリの前にスキーマを課す必要はありません。

抽出、ロード、変換 (ETL) オペレーションで従来のマテリアライズドビューへの柔軟なクエリ – スキーマレスおよび半構造化データを SUPER に保存した後、PartiQL マテリアライズドビューを使用してデータを観察し、マテリアライズドビューでシュレッド処理を行えます。

細分化されたデータを含むマテリアライズドビューは、従来の分析ケースに対するパフォーマンスと使いやすさによる利点の良い例です。細分化されたデータに対して分析を実行すると、Amazon Redshift マテリアライズドビューの列構造により、パフォーマンスが向上します。さらに、取り込まれたデータに対して従来のスキーマを必要とするユーザーおよびビジネスインテリジェンス (BI) ツールは、データの従来のスキーマ表示としてビュー (マテリアライズドまたは仮想) を使用できます。

PartiQL マテリアライズドビューが、JSON または SUPER で見つかったデータを従来の列指向マテリアライズドビューに抽出した後、マテリアライズドビューにクエリを実行できます。SUPER データ型とマテリアライズドビューの詳しい仕組みについては、「SUPER データ型とマテリアライズドビュー」を参照してください。

SUPER 型の列のパス上の `scalar` 値に動的データマスキングポリシーを適用できます。動的データマスキングの詳細については、「動的データマスキング」を参照してください。SUPER データ型での動的データマスキングの使用に関する詳細については、「SUPER データタイプパスでの動的データマスキングの使用」を参照してください (プレビュー)。

SUPER データ型の詳細については、「SUPER タイプ」を参照してください。

SUPER データ型の使用例の詳細については、「SUPER サンプルデータセット」で始まる、このトピックのサブセクションを参照してください。

SUPER データ型の使用概念

以下に、Amazon Redshift SUPER データ型の概念をいくつか示します。

Amazon Redshift での SUPER データ型を理解する – SUPER データ型は Amazon Redshift データ型であり、Amazon Redshift スカラーや、場合によってはネストされた配列や構造体を含むスキーマレ

ス配列や構造体を保存できます。SUPER データ型は、JSON やドキュメント指向のソースから発信されたデータなど、さまざまな形式の半構造化データをネイティブに保存できます。新しい SUPER 列を追加して、半構造化データを保存し、通常のスカラー列とともに SUPER 列にアクセスするクエリを書き込むことができます。SUPER データ型の詳細については、SUPER タイプを参照してください。

スキーマレス JSON を SUPER に取り込む – 柔軟な半構造化 SUPER データ型を使用すると、Amazon Redshift はスキーマレス JSON を受信して SUPER 値に取り込むことができます。例えば、Amazon Redshift は JSON 値 [10.5, ‘first’] を SUPER 値 [10.5, 'first'] に取り込むことができます。これは、Amazon Redshift の 10 進数 10.5 と varchar 'first' を含む配列です。Amazon Redshift は、COPY コマンドまたは json_parse('[10.5, "first"]') などの JSON 解析関数を使用して、JSON を SUPER 値に取り込むことができます。デフォルトでは、COPY と `json_parse` のどちらも、厳密な解析セマンティクスを使用して JSON を取り込みます。また、データベースデータ自体を使用して、配列や構造体を含む SUPER 値を構築することもできます。

SUPER 列は、スキーマレス JSON の不規則な構造を取り込むときに、スキーマを変更する必要はありません。例えば、クリックストリームを分析するときに、最初に「IP」と「時間」属性を持つ「クリック」構造を SUPER 列に保存します。このような変更を取り込むために、スキーマを変更せずに属性「customer id」を追加できます。

SUPER データ型に使用されるネイティブ形式は、テキスト形式の JSON 値よりも少ないスペースを必要とするバイナリ形式です。これにより、クエリでの SUPER 値の取り込みと実行時の処理が高速になります。

PartiQL を使用して SUPER データをクエリする – PartiQL は、SQL-92 の拡張機能として下位互換性を持ち、現在多くの AWS サービスにより使用されています。PartiQL を使用すると、使い慣れた SQL 構造は、従来の表形式の SQL データと SUPER の半構造化データの両方へのアクセスがシームレスに組み合わされます。オブジェクトおよび配列のナビゲーションを実行し、配列のネストを解除できます。PartiQL は、標準の SQL 言語を拡張して、ネストされた複数値を持つデータを宣言的に表現し、処理します。

PartiQL は、SQL の拡張であり、SUPER 列のネストされたスキーマレスデータは第一級市民です。PartiQL では、クエリのコンパイル時にすべてのクエリ式を型チェックする必要はありません。この方法は、SUPER 列内の実際のデータ型にアクセスしたときに、クエリの実行中に動的に型付けされる SUPER データ型を含むクエリ式を有効にします。また、PartiQL は lax モードで動作します。このモードでは、型の不整合によって障害が発生することはありませんが、null が返されます。スキーマレスクエリと lax クエリ処理の組み合わせにより、PartiQL は、SQL クエリが SUPER 列に取り込まれる JSON データを評価する抽出、ロード、転送 (ELT) アプリケーションに最適です。

Redshift Spectrum との統合 – Amazon Redshift は、JSON、Parquet、およびネストされたデータを持つその他の形式で Redshift Spectrum クエリを実行するときに、PartiQL の複数の側面をサポートします。Redshift Spectrum は、スキーマを持つネストされたデータのみをサポートします。例えば、Redshift Spectrum を使用すると、JSON データの属性 nested_schemaful_example がスキーマ ARRAY<STRUCT<a:INTEGER, b:DECIMAL(5,2)>> にあることを宣言できます。この属性のスキーマは、データに常に配列が含まれていることを決定します。配列には、整数 a と小数 b の構造が含まれています。より多くの属性を含むようにデータが変更されると、タイプも変更されます。対照的に、SUPER データ型にはスキーマは必要ありません。異なる属性または型を持つ構造体要素を持つ配列を保存することができます。また、いくつかの値は配列の外側に保存することができます。

SUPER データ型をサポートする関数の詳細については、以下を参照してください。

- ABS 関数
- CEILING (または CEIL) 関数
- FLOOR 関数
- ROUND 関数
- SIGN 関数
- TRUNC 関数

SUPER データに関する考慮事項

SUPER データを使用する際、以下の点を考慮してください。

- JDBC ドライバーのバージョン 1.2.50、ODBC ドライバーのバージョン 1.4.17 以降、および Amazon Redshift Python ドライバーのバージョン 2.0.872 以降を使用します。

  JDBC ドライバーの詳細については、「JDBC 接続の設定」を参照してください。

  ODBC ドライバーの詳細については、「ODBC 接続の設定」を参照してください。
- SUPER サンプルデータセット の次のトピックで使用されているスキーマの例をご覧ください。
- 次のトピックで使用されるすべての SQL コード例は、ダウンロード用に同じ S3 プレフィックスが付いています。これには、データ定義言語 (DDL) と COPY ステートメント、SUPER で動作する特定の TPC-H 変更クエリが含まれます。

  SQL ファイルを表示またはダウンロードするには、次のいずれかを実行します。
  - SUPER チュートリアル SQL ファイルと TPC-H ファイルをダウンロードします。

- Amazon S3 CLI を使用して、次のコマンドを実行します。独自のターゲットパスを使用できます。

```
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/semistructured-
tutorial.sql /target/path
aws s3 cp s3://redshift-downloads/semistructured/tutorialscripts/
super_tpch_queries.sql /target/path
```

SUPER 設定の詳細については、「SUPER 設定」を参照してください。

SUPER サンプルデータセット

SUPER サンプルデータセットを探索して分析できます。このデータセットには、さまざまなカテゴリ、リージョン、期間にわたる架空の製品の販売に関連するデータが含まれています。以下のセクションでは、Amazon Redshift クラスター内の SUPER サンプルデータセットへのアクセス、クエリ、操作について詳しく説明します。

取り込みやクエリの例に使用されるテーブルスキーマとデータモデルは、次のように定義されています。

```
/*customer-orders-lineitem*/
CREATE TABLE customer_orders_lineitem
(c_custkey bigint
,c_name varchar
,c_address varchar
,c_nationkey smallint
,c_phone varchar
,c_acctbal decimal(12,2)
,c_mktsegment varchar
,c_comment varchar
,c_orders super
);

/* Datamodel of documents to be stored in c_orders Super column would be as follows*/
ARRAY < STRUCT < o_orderkey:bigint
                           ,o_orderstatus:string
                           ,o_totalprice:double
                           ,o_orderdate:string
                           ,o_orderpriority:string
                           ,o_clerk:string
```

```
                            ,o_shippriority:int
                            ,o_comment:string
                            ,o_lineitems:ARRAY < STRUCT < l_partkey:bigint
                                                        ,l_suppkey:bigint
                                                        ,l_linenumber:int
                                                        ,l_quantity:double
                                                        ,l_extendedprice:double
                                                        ,l_discount:double
                                                        ,l_tax:double
                                                        ,l_returnflag:string
                                                        ,l_linestatus:string
                                                        ,l_shipdate:string
                                                        ,l_commitdate:string
                                                        ,l_receiptdate:string
                                                        ,l_shipinstruct:string
                                                        ,l_shipmode:string
                                                        ,l_comment:string
                                                        > >
                             > >

/*part*/
CREATE TABLE part
(
 p_partkey bigint
 ,p_name varchar
 ,p_mfgr varchar
 ,p_brand varchar
 ,p_type varchar
 ,p_size int
 ,p_container varchar
 ,p_retailprice decimal(12,2)
 ,p_comment varchar
 );

/*region-nations*/
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
 );

 /* Datamodel of documents to be stored in r_nations Super column would be as follows*/
```

```
 ARRAY < STRUCT < n_nationkey:int,n_name:string,n_comment:string > >

/*supplier-partsupp*/
CREATE TABLE supplier_partsupp
(
 s_suppkey bigint
 ,s_name varchar
 ,s_address varchar
 ,s_nationkey smallint
 ,s_phone varchar
 ,s_acctbal double precision
 ,s_comment varchar
 ,s_partsupps super
 );

 /* Datamodel of documents to be stored in s_partsupps Super column would be as
 follows*/
 ARRAY < STRUCT <
 ps_partkey:bigint,ps_availqty:int,ps_supplycost:double,ps_comment:string > >
```

半構造化データを Amazon Redshift にロードする

SUPER データ型を使用して、Amazon Redshift の階層データと汎用データを永続化およびクエリします。Amazon Redshift は JSON 形式のデータを解析して SUPER 表現に変換する `json_parse` 関数が導入されています。Amazon Redshift は COPY コマンドを使用した SUPER 列のロードもサポートしています。サポートされているファイル形式は、JSON、Avro、テキスト、コンマ区切り値 (CSV) 形式、Parquet、および ORC 形式です。

次の例で使用されるテーブルの詳細については、「SUPER サンプルデータセット」を参照してください。

`json_parse` 関数の詳細については、「JSON_PARSE 関数」を参照してください。

SUPER データ型のデフォルトのエンコードは ZSTD です。

JSON ドキュメントを SUPER 列で解析する

`json_parse` 関数を使用すると、SUPER 列に JSON データを挿入または更新できます。この関数は、データを JSON 形式で解析し、SUPER データ型に変換します。SUPER データ型は INSERT ステートメントまたは UPDATE ステートメントで使用できます。

次の例では、JSON データを SUPER 列に挿入します。`json_parse` 関数がクエリに含まれていない場合、Amazon Redshift は関連する値を、解析する必要がある JSON 形式の文字列ではなく単一の文字列として扱います。

SUPER データ列を更新する場合、Amazon Redshift では、ドキュメント全体を列の値に渡す必要があります。Amazon Redshift は、部分的な更新をサポートしていません。

```
INSERT INTO region_nations VALUES(0,
   'lar deposits. blithely final packages cajole. regular waters are final requests.
 regular accounts are according to',
   'AFRICA',
   JSON_PARSE('{"r_nations":[
      {"n_comment":" haggle. carefully final deposits detect slyly agai",
         "n_nationkey":0,
         "n_name":"ALGERIA"
      },
      {"n_comment":"ven packages wake quickly. regu",
         "n_nationkey":5,
         "n_name":"ETHIOPIA"
      },
      {"n_comment":" pending excuses haggle furiously deposits. pending, express pinto
 beans wake fluffily past t",
         "n_nationkey":14,
         "n_name":"KENYA"
      },
      {"n_comment":"rns. blithely bold courts among the closely regular packages use
 furiously bold platelets?",
         "n_nationkey":15,
         "n_name":"MOROCCO"
      },
      {"n_comment":"s. ironic, unusual asymptotes wake blithely r",
         "n_nationkey":16,
         "n_name":"MOZAMBIQUE"
      }
   ]
}'));
```

COPY を使用して Amazon Redshift で SUPER 列をロードする

以下のセクションでは、COPY コマンドを使用して JSON データを Amazon Redshift にロードするさまざまな方法について説明します。

JSON および Avro からのデータのコピー

Amazon Redshift で半構造化データサポートを使用すると、JSON 構造の属性を複数の列にシュレッダー処理することなく JSON ドキュメントをロードできます。

Amazon Redshift には、完全にまたは部分的に不明な JSON 構造であっても、COPY を使用して JSON ドキュメントを取り込むための方法が 2 つあります。

1. `noshred` オプションを使用して、JSON ドキュメントから派生したデータを単一の SUPER データ列に保存します。この方法は、スキーマが不明であるか、変更が予想される場合に便利です。したがって、このメソッドを使用すると、単一の SUPER 列にタプル全体を容易に保存することができます。
2. `auto` または `jsonpaths` オプションを使用して、JSON ドキュメントを複数の Amazon Redshift 列にシュレッダー処理します。属性には、Amazon Redshift スカラーまたは SUPER 値を指定できます。

これらのオプションは、JSON 形式または Avro 形式で使用できます。

シュレッダー処理前の JSON オブジェクトの最大サイズは 4 MB です。

JSON ドキュメントを単一の SUPER データ列にコピーする

JSON ドキュメントを単一の SUPER データ列にコピーするには、単一の SUPER データ列を持つテーブルを作成します。

```
CREATE TABLE region_nations_noshred (rdata SUPER);
```

Amazon S3 から単一の SUPER データ列にデータをコピーします。JSON ソースデータを単一の SUPER データ列に取り込むには、FORMAT JSON 句で `noshred` オプションを指定します。

```
COPY region_nations_noshred FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'noshred';
```

COPY が JSON を正常に取り込んだ後のテーブルには、JSON オブジェクト全体のデータを含む `rdata` SUPER データ列があります。取り込まれたデータは、JSON 階層のすべてのプロパティを保持します。ただし、効率的なクエリ処理のために、リーフは Amazon Redshift スカラー型に変換されます。

次のクエリを使用して、元の JSON 文字列を取得します。

```
SELECT rdata FROM region_nations_noshred;
```

Amazon Redshift が SUPER データ列を生成すると、JSON シリアル化によって JDBC を文字列として使用してアクセスできるようになります。詳細については、「複雑なネストされた JSON のシリアル化」を参照してください。

JSON ドキュメントを複数の SUPER データ列にコピーする

JSON ドキュメントは、SUPER データ列または Amazon Redshift スカラー型のいずれかである複数の列に細分家できます。Amazon Redshift は、JSON オブジェクトのさまざまな部分を異なる列に分散します。

```
CREATE TABLE region_nations
(
 r_regionkey smallint
 ,r_name varchar
 ,r_comment varchar
 ,r_nations super
 );
```

前の例のデータをテーブルにコピーするには、FORMAT JSON 句で AUTO オプションを指定して、JSON 値を複数の列に分割します。COPY は、最上位の JSON 属性を列名と照合し、ネストされた値を JSON 配列やオブジェクトなどの SUPER 値として取り込むことができます。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';
```

JSON 属性名が大文字と小文字が混在する場合は、FORMAT JSON 句の中で `auto ignorecase` オプションを指定します。COPY コマンドの詳細については、「'auto ignorecase' オプションを使用した JSON データからのロード」を参照してください。

場合によっては、列名と JSON 属性の間に不一致があるか、ロードする属性がレベルより深くネストされていることがあります。その場合は、`jsonpaths` ファイルを使用して JSON 属性を Amazon Redshift 列に手動でマッピングします。

```
CREATE TABLE nations
```

```
(
 regionkey smallint
 ,name varchar
 ,comment super
 ,nations super
 );
```

列名が JSON 属性と一致しないテーブルにデータをロードするとします。次の例では、`nations` テーブルがそのようなテーブルです。`jsonpaths` 配列内の位置によってテーブル列に属性のパスをマップする `jsonpaths` ファイルを作成できます。

```
{"jsonpaths": [
       "$.r_regionkey",
       "$.r_name",
       "$.r_comment",
       "$.r_nations
    ]
}
```

`jsonpaths` ファイルの場所は、FORMAT JSON の引数として使用されます。

```
COPY nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_jsonpaths.json';
```

次のクエリを使用して、複数の列に分散されたデータを表示するテーブルにアクセスします。SUPER データ列は、JSON 形式を使用して印刷されます。

```
SELECT r_regionkey,r_name,r_comment,r_nations[0].n_nationkey FROM region_nations ORDER
 BY 1,2,3 LIMIT 1;
```

Jsonpaths ファイルは、JSON ドキュメント内のフィールドをテーブル列にマップします。SUPER 列として完全なドキュメントをロードしながら、分散キーおよびソートキーなどの追加の列を抽出できます。以下のクエリは、完全なドキュメントを nations 列にロードします。name 列はソートキーで、`regionkey` 列は分散キーです。

```
CREATE TABLE nations_sorted (
    regionkey smallint,
    name varchar,
```

```
    nations super
) DISTKEY(regionkey) SORTKEY(name);
```

ルート jsonpath「$」は、以下のようにドキュメントのルートにマップされます。

```
{"jsonpaths": [
        "$.r_regionkey",
        "$.r_name",
        "$"
    ]
}
```

jsonpaths ファイルの場所は、FORMAT JSON の引数として使用されます。

```
COPY nations_sorted FROM 's3://redshift-downloads/semistructured/tpch-nested/data/json/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 's3://redshift-downloads/semistructured/tpch-nested/data/jsonpaths/
nations_sorted_jsonpaths.json';
```

テキストおよび CSV からのデータのコピー

Amazon Redshift は、シリアルナンバーが付けられた JSON として、テキスト形式および CSV 形式の SUPER 列を表します。SUPER 列が正しい型情報をロードするには、有効な JSON フォーマットが必要です。オブジェクト、配列、数値、ブール値、およびNULL値の引用符を外します。文字列値を二重引用符で囲みます。SUPER 列は、テキストおよび CSV 形式に標準のエスケープ規則を使用します。CSV の場合、区切り文字は CSV 標準に従ってエスケープされます。テキスト形式の場合、選択した区切り文字が SUPER フィールドにも表示される可能性がある場合は、COPY および UNLOAD 中に ESCAPE オプションを使用します。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/csv/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT CSV;
```

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/text/
region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
DELIMITER ','
ESCAPE;
```

列形式の Parquet および ORC からのデータのコピー

半構造化データまたはネストされたデータが Apache Parquet または Apache ORC 形式で既に使用可能な場合は、COPY コマンドを使用して Amazon Redshift にデータを取り込むことができます。

Amazon Redshift テーブル構造は、Parquet ファイルまたは ORC ファイルの列数と列データ型と一致する必要があります。COPY コマンドで SERIALIZETOJSON を指定することにより、テーブルの SUPER 列と整列するファイル内の任意の列タイプを SUPER としてロードできます。これには、構造体型と配列型が含まれます。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/
parquet/region_nation'
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT PARQUET SERIALIZETOJSON;
```

次の例では、ORC 形式を使用します。

```
COPY region_nations FROM 's3://redshift-downloads/semistructured/tpch-nested/data/orc/
region_nation'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT ORC SERIALIZETOJSON;
```

日付または時刻のデータ型の属性が ORC の場合、Amazon Redshift はそれらを SUPER でエンコードするときに varchar に変換します。

半構造化データのアンロード

Amazon Redshift を使用すると、Amazon Redshift クラスターから Amazon S3 に、テキスト、Apache Parquet、Apache ORC、Avro などのさまざまな形式で半構造化データをエクスポートできます。以下のセクションでは、Amazon Redshift で半構造化データのアンロード操作を設定および実行するプロセスについて説明します。

CSV or text formats

SUPER データ列を持つテーブルを、コンマ区切り値 (CSV) 形式またはテキスト形式で Amazon S3 にアンロードできます。navigation 句と unnest 句の組み合わせを使用して、Amazon Redshift は SUPER データ形式の階層データを CSV 形式またはテキスト形式の Amazon S3 にアンロードします。その後、アンロードされたデータに対して外部テーブルを作成し、Redshift Spectrum を使用してクエリを実行できます。UNLOAD の使用と必要な IAM アクセス許可の詳細については、「UNLOAD」を参照してください。

次の例を実行する前に、「半構造化データを Amazon Redshift にロードする」に記載されているプロセスを使用して region_nations テーブルにデータを入力します。次の例で使用されるテーブルの詳細については、「SUPER サンプルデータセット」を参照してください。

次の例では、Amazon S3 にデータをアンロードします。

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
DELIMITER AS '|'
GZIP
ALLOWOVERWRITE;
```

ユーザー定義の文字列が null 値を表す他のデータ型とは異なり、Amazon Redshift は JSON 形式を使用して SUPER データ列をエクスポートし、JSON 形式で決定されるように null として表します。その結果、SUPER データ列では、UNLOAD コマンドで使用される NULL [AS] オプションを無視します。

Parquet format

SUPER データ列を持つテーブルをParquet形式で Amazon S3 にアンロードできます。Amazon Redshift は、JSON データ型として、Parquetで SUPER 列を表します。これにより、半構造化データを Parquet で表すことができます。これらの列は、Redshift Spectrum を使用してクエリするか、またはCOPY コマンドを使用して Amazon Redshift に取り戻すことができます。UNLOAD の使用と必要な IAM アクセス許可の詳細については、「UNLOAD」を参照してください。

次の例では、データをParquet形式でAmazon S3 にアンロードします。

```
UNLOAD ('SELECT * FROM region_nations')
TO 's3://xxxxxx/'
IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3-Write'
FORMAT PARQUET;
```

半構造化データのクエリ

Amazon Redshift を使用すると、JSON、Avro、Ion などの半構造化データを構造化データとともにクエリおよび分析できます。半構造化データとは、階層構造またはネスト構造を可能にする柔軟なスキーマを持つデータを指します。以下のセクションでは、Amazon Redshift のオープンデータ形式の

サポートを使用して半構造化データをクエリする方法について説明します。これにより、複雑なデータ構造から有益な情報を引き出すことができます。

Amazon Redshift は PartiQL 言語を使用して、リレーショナルデータ、半構造化データ、ネストされたデータへの SQL 互換アクセスを提供します。

PartiQL は、動的型で動作します。このアプローチにより、構造化、半構造化、およびネストされたデータセットの組み合わせに対して、直感的なフィルタリングや結合、および集約を行えるようになります。PartiQL 構文では、ネストされたデータにアクセスするときに、パスナビゲーションにドット付き表記と配列添字を使用します。また、FROM 句の項目が配列を反復処理し、不正なオペレーションに使用できるようにします。以下では、パスおよび配列のナビゲーション、ネスト解除、または結合を、SUPER データ型で行う場合の、さまざまなクエリパターンについて説明します。

次の例で使用されるテーブルの詳細については、「SUPER サンプルデータセット」を参照してください。

トピック

ナビゲーション

Amazon Redshift は PartiQL を使用して、それぞれ [...] 角括弧とドット表記を使用して、配列と構造体へのナビゲーションを可能にします。さらに、ドット表記を使用して構造体に、角括弧表記を使用して配列にナビゲーションを混在させることができます。例えば、次の例では、`c_orders` SUPER データ列が構造体を持つ配列であり、属性の名前が `o_orderkey` であると仮定しています。

テーブル `customer_orders_lineitem` にデータを挿入するには、次のコマンドを実行します。IAM ロールを独自の認証情報に置き換えます。

```
COPY customer_orders_lineitem FROM 's3://redshift-downloads/semistructured/tpch-nested/
data/json/customer_orders_lineitem'
```

```
REGION 'us-east-1' IAM_ROLE 'arn:aws:iam::xxxxxxxxxxxx:role/Redshift-S3'
FORMAT JSON 'auto';

SELECT c_orders[0].o_orderkey FROM customer_orders_lineitem;
```

Amazon Redshift では、表記のプレフィックスとしてテーブルエイリアスも使用します。次の例では、前の例と同じクエリを実行します。

```
SELECT cust.c_orders[0].o_orderkey FROM customer_orders_lineitem AS cust;
```

フィルタリング、結合、集約など、すべてのタイプのクエリでドットと角括弧の表記を使用できます。この表記は、通常の場合に列参照を含んでいるクエリで使用します。次の例では、結果をフィルタリングする SELECT ステートメントを使用します。

```
SELECT count(*) FROM customer_orders_lineitem WHERE c_orders[0]. o_orderkey IS NOT
 NULL;
```

次の例では、GROUP BY 句と ORDER BY 句の両方で角括弧とドットのナビゲーションを使用します。

```
SELECT c_orders[0].o_orderdate,
       c_orders[0].o_orderstatus,
       count(*)
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderkey IS NOT NULL
GROUP BY c_orders[0].o_orderstatus,
         c_orders[0].o_orderdate
ORDER BY c_orders[0].o_orderdate;
```

ネストされていないクエリ

クエリをネスト解除するために、Amazon Redshift は PartiQL 構文を使用しながら SUPER 配列を反復処理します。このために、クエリの FROM 句を使用して配列上をナビゲートします。前の例を使用して、次の例では、属性値 `c_orders` を繰り返し処理しています。

```
SELECT c.*, o FROM customer_orders_lineitem c, c.c_orders o;
```

ネスト解除構文は、FROM 句の拡張です。標準 SQL では、FROM 句 x (AS) y は x と関連する各タプルを y が反復処理することを意味します。この場合、x は関連を指し、y はその関連 x のため

のエイリアスを指します。同様に、FROM 句項目 x （AS） y を使用してネスト解除する PartiQL 構文では、y が(SUPER) 配列式 x 内の各 (SUPER) 値を反復処理することを意味します。この場合、x は SUPER 表現であり、y は x のエイリアスです。

左のオペランドは、通常のナビゲーションのためにドットと角括弧の表記を使用することもできます。前の例では、customer_orders_lineitem c は customer_order_lineitem ベースのテーブルに対する反復処理であり、c.c_orders o は c.c_orders 配列に対する反復処理です。配列内の配列である o_lineitems 属性を反復処理するには、複数の句を追加する必要があります。

```
SELECT c.*, o, l FROM customer_orders_lineitem c, c.c_orders o, o.o_lineitems l;
```

Amazon Redshift では、AT キーワードを使用して配列の反復処理を行う際の、配列インデックスもサポートしています。句 x AS y AT z は、配列 x を反復処理し、配列インデックスとしてフィールド z, を生成します。次の例は、配列インデックスがどのように機能するかを示しています。

```
SELECT c_name,
       orders.o_orderkey AS orderkey,
       index AS orderkey_index
FROM customer_orders_lineitem c, c.c_orders AS orders AT index
ORDER BY orderkey_index;

c_name             | orderkey | orderkey_index
-------------------+----------+----------------
Customer#000008251 | 3020007  |        0
Customer#000009452 | 4043971  |        0
  (2 rows)
```

次の例では、スカラー配列の繰り返し処理を行います。

```
CREATE TABLE bar AS SELECT json_parse('{"scalar_array": [1, 2.3, 45000000]}') AS data;

SELECT index, element FROM bar AS b, b.data.scalar_array AS element AT index;

 index | element
-------+----------
     0 | 1
     1 | 2.3
     2 | 45000000
(3 rows)
```

次の例では、複数のレベルの配列を繰り返し処理します。この例では、複数の UNNEST 句を使用して、最も内側の配列を反復処理します。f.multi_level_array AS 配列は multi_level_array を反復処理します。配列 AS 要素は、multi_level_array 内の配列に対する反復処理を表します。

```
CREATE TABLE foo AS SELECT json_parse('[[1.1, 1.2], [2.1, 2.2], [3.1, 3.2]]') AS
 multi_level_array;

SELECT array, element FROM foo AS f, f.multi_level_array AS array, array AS element;

   array    | element
-----------+---------
 [1.1,1.2] | 1.1
 [1.1,1.2] | 1.2
 [2.1,2.2] | 2.1
 [2.1,2.2] | 2.2
 [3.1,3.2] | 3.1
 [3.1,3.2] | 3.2
(6 rows)
```

FROM 句の詳細については、「FROM 句」を参照してください。

オブジェクトのピボット解除

オブジェクトのピボット解除を実行する際、Amazon Redshift は、PartiQL 構文により SUPER オブジェクトを反復処理します。この処理は、UNPIVOT キーワードを指定しながらクエリの FROM 句を使用することで実現されます。この場合、式は c.c_orders[0] オブジェクトです。このクエリ例では、オブジェクトから返された各属性を反復処理します。

```
SELECT attr as attribute_name, json_typeof(val) as value_type
FROM customer_orders_lineitem c, UNPIVOT c.c_orders[0] AS val AT attr
WHERE c_custkey = 9451;

 attribute_name  | value_type
-----------------+------------
 o_orderstatus   | string
 o_clerk         | string
 o_lineitems     | array
 o_orderdate     | string
 o_shippriority  | number
 o_totalprice    | number
```

```
 o_orderkey      | number
 o_comment       | string
 o_orderpriority | string
(9 rows)
```

ネスト解除構文と同様に、ピボット解除構文も FROM 句の拡張の 1 つです。違いは、ピボット解除構文では UNPIVOT キーワードを使用して、配列ではなくオブジェクトを反復処理することを示す点です。これは、AS `value_alias` を使用してオブジェクト内のすべての値を反復処理し、AT `attribute_alias` を使用してすべての属性を反復処理します。次の構文フラグメントを検討してください。

```
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

Amazon Redshift は、次のように、単一の FROM 句でオブジェクトのピボット解除と配列のネスト解除の使用をサポートしています。

```
SELECT attr as attribute_name, val as object_value
FROM customer_orders_lineitem c, c.c_orders AS o, UNPIVOT o AS val AT attr
WHERE c_custkey = 9451;
```

オブジェクトのピボット解除を使用する場合、Amazon Redshift では相関ピボット解除はサポートされません。具体的には、異なるクエリレベルで複数のピボット解除が存在し、内側のピボット解除が外側のピボット解除を参照している場合が考えられます。Amazon Redshift は、このタイプの複数にわたるピボット解除をサポートしていません。

FROM 句の詳細については、「FROM 句」を参照してください。PIVOT および UNPIVOT を使用して構造化データをクエリする方法の例については、「PIVOT と UNPIVOT の例」を参照してください。

動的型付け

動的型付けでは、ドットと角括弧のパスから抽出されたデータを明示的にキャストする必要はありません。Amazon Redshift では、動的型付けを使用して、クエリで使用する前にデータ型を宣言することなく、スキーマレスの SUPER データを処理します。動的型付けでは、明示的に Amazon Redshift 型にキャストすることなく、SUPER データ列に移動した結果が使用されます。動的型付けは、結合および GROUP BY 句で最も便利です。次の例では、通常の Amazon Redshift 型にドット式と角括弧式を明示的にキャストする必要がない SELECT ステートメントを使用しています。タイプの互換性および変換の詳細については、「型の互換性と変換」を参照してください。

```
SELECT c_orders[0].o_orderkey
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus = 'P';
```

c_orders[0].o_orderstatus が文字列「P」の場合、このクエリの等号は `true` と評価されます。他のすべての場合、等号は `false` と評価されます。これには、等号の引数が異なるタイプである場合も含まれます。

動的型と静的型

動的型付けを使用しないと、c_orders[0].o_orderstatus が文字列、整数、または構造体のいずれであるかを判断できません。c_orders[0].o_orderstatus が SUPER データ型であると判断できるのは、Amazon Redshift スカラー、配列、または構造体の場合のみです。c_orders[0].o_orderstatus の静的型は、SUPER データ型です。従来、型は SQL では暗黙的に静的型です。

Amazon Redshift は、スキーマレスデータの処理に動的型付けを使用します。クエリがデータを評価すると、c_orders[0].o_orderstatus は特定のタイプであることが判明します。例えば、customer_orders_lineitem の最初のレコードで c_orders[0].o_orderstatus を評価すると、整数になることがあります。2 番目のレコードを評価すると、文字列になることがあります。これらは、式の動的型です。

動的型を持つドット式と角括弧式で SQL 演算子または関数を使用すると、Amazon Redshift は、それぞれの静的型で標準の SQL 演算子または関数を使用する場合と同様の結果を生成します。この例では、パス式の動的型が文字列である場合、文字列 'P' との比較は意味を持ちます。c_orders[0].o_orderstatus の動的型が文字列以外のデータ型であるときはいつでも、等式が false を返します。誤って入力された引数が使用されると、他の関数は null を返します。

次の例では、静的型付けを使用して前のクエリを書き込みます。

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR = 'P'
           ELSE FALSE END;
```

等価述語と比較述語の次の違いに注意してください。前の例では、等価述語を下回る述語に置き換えると、セマンティクスは false ではなく null を生成します。

```
SELECT c_orders[0]. o_orderkey
```

```
FROM customer_orders_lineitem
WHERE c_orders[0].o_orderstatus <= 'P';
```

この例では、c_orders[0].o_orderstatus が文字列の場合、アルファベット順で「P」以下であれば、Amazon Redshift は true を返します。アルファベット順で 'P' より大きい場合、Amazon Redshift は false を返します。ただし、c_orders[0].o_orderstatus が文字列でない場合、Amazon Redshift では次のクエリに示すように、異なるタイプの値を比較できないため、Amazon Redshift は null を返します。

```
SELECT c_custkey
FROM customer_orders_lineitem
WHERE CASE WHEN JSON_TYPEOF(c_orders[0].o_orderstatus) = 'string'
           THEN c_orders[0].o_orderstatus::VARCHAR <= 'P'
           ELSE NULL END;
```

動的型付けは、最小限の比較を行える型の比較から除外されません。例えば、CHAR と VARCHAR の両方の Amazon Redshift スカラー型を SUPER に変換できます。これらは、Amazon Redshift CHAR および VARCHAR 型に似た末尾の空白文字を無視するなど、文字列と同等です。同様に、整数型、小数型、および浮動小数点値は SUPER 値と同等です。特に 10 進数列の場合、各値は異なるスケールを持つこともできます。Amazon Redshift では、これらを引き続き動的型と見なします。

また、Amazon Redshift は、オブジェクトや配列の深さを評価し、すべての属性を比較するなど、深さが等しいと評価されるオブジェクトや配列の等価性もサポートしています。deep equal を実行するプロセスは時間がかかる可能性があるため、deep equal の使用には注意が必要です。

結合での動的型付けの使用

結合の場合、動的型付けは、長い CASE WHEN 分析を実行して表示される可能性のあるデータ型を見つける代わりに、値をさまざまな動的型と自動的に一致させます。例えば、組織がパートキーに使用していた形式を時間の経過とともに変更したと仮定します。

発行された最初の整数部分キーは、'A55' などの文字列部分キーに置き換えられ、後で文字列と数字を組み合わせた ['X', 10] などの配列部分キーに再び置き換えられます。Amazon Redshift では、パートキーについて長いケース分析を行う必要がなく、次の例に示すように、結合を使用できます。

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
```

```
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE l.l_partkey = ps.ps_partkey
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

次の例では、動的型付けを使用しない場合、同じクエリがどれほど複雑で非効率的であるかを示しています。

```
SELECT c.c_name
    ,l.l_extendedprice
    ,l.l_discount
FROM customer_orders_lineitem c
    ,c.c_orders o
    ,o.o_lineitems l
    ,supplier_partsupp s
    ,s.s_partsupps ps
WHERE CASE WHEN IS_INTEGER(l.l_partkey) AND IS_INTEGER(ps.ps_partkey)
           THEN l.l_partkey::integer = ps.ps_partkey::integer
           WHEN IS_VARCHAR(l.l_partkey) AND IS_VARCHAR(ps.ps_partkey)
           THEN l.l_partkey::varchar = ps.ps_partkey::varchar
           WHEN IS_ARRAY(l.l_partkey) AND IS_ARRAY(ps.ps_partkey)
                AND IS_VARCHAR(l.l_partkey[0]) AND IS_VARCHAR(ps.ps_partkey[0])
                AND IS_INTEGER(l.l_partkey[1]) AND IS_INTEGER(ps.ps_partkey[1])
           THEN l.l_partkey[0]::varchar = ps.ps_partkey[0]::varchar
                AND l.l_partkey[1]::integer = ps.ps_partkey[1]::integer
           ELSE FALSE END
AND c.c_nationkey = s.s_nationkey
ORDER BY c.c_name;
```

Lax のセマンティクス

デフォルトでは、SUPER 値に対するナビゲーションオペレーションは、ナビゲーションが無効であるときにエラーを返す代わりに null を返します。SUPER 値がオブジェクトでない場合、または SUPER 値がオブジェクトであるが、クエリで使用される属性名が含まれていない場合、オブジェクトのナビゲーションは無効です。例えば、次のクエリは、SUPER データ列の cdata で無効な属性名にアクセスします。

```
SELECT c.c_orders.something FROM customer_orders_lineitem c;
```

SUPER 値が配列でない場合、または配列インデックスが範囲外の場合、配列ナビゲーションは null を返します。次のクエリは、c_orders[1][1] が範囲外であるため、null を返します。

```
SELECT c.c_orders[1][1] FROM customer_orders_lineitem c;
```

Lax セマンティクスは、動的型付けを使用して SUPER 値をキャストする場合に特に便利です。SUPER 値を間違ったタイプにキャストすると、キャストが無効な場合、エラーではなく null が返されます。例えば、次のクエリは、オブジェクト属性 o_orderstatus の文字列値 'Good' を INTEGER にキャストできないため、null を返します。Amazon Redshift は、VARCHAR から INTEGER へのキャストではエラーを返しますが、SUPER キャストではエラーを返しません。

```
SELECT c.c_orders.o_orderstatus::integer FROM customer_orders_lineitem c;
```

内観の種類

SUPER データ列は、SUPER 値に関する動的型およびその他の型情報を返す検査関数をサポートします。最も一般的な例は、SUPER 値の動的型に応じて、ブール値、数値、文字列、オブジェクト、配列、または null の値を持つ VARCHAR を返す JSON_TYPEOF スカラー関数です。Amazon Redshift は、SUPER データ列に対して次のブール関数をサポートしています。

- DECIMAL_PRECISION
- DECIMAL_SCALE
- IS_ARRAY
- IS_BIGINT
- IS_CHAR
- IS_DECIMAL
- IS_FLOAT
- IS_INTEGER
- IS_OBJECT
- IS_SCALAR
- IS_SMALLINT
- IS_VARCHAR
- JSON_TYPEOF

入力値が null の場合、これらの関数はすべて false を返します。IS_SCALAR、IS_OBJECT、および IS_ARRAY は相互に排他的であり、null を除くすべての可能な値をカバーします。

データに対応するタイプを推測するために、Amazon Redshift は次の例に示すように、SUPER 値のタイプ (のトップレベル) を返す JSON_TYPEOF 関数を使用します。

```
SELECT JSON_TYPEOF(r_nations) FROM region_nations;
 json_typeof
 -------------
 array
(1 row)
```

```
SELECT JSON_TYPEOF(r_nations[0].n_nationkey) FROM region_nations;
 json_typeof
 -------------
 number
```

Amazon Redshift では、この値を SUPER ではなく VARCHAR 列に挿入するのと同様に、単一の長い文字列として認識します。列が SUPER であるため、単一の文字列は引き続き有効な SUPER 値であり、その差は JSON_TYPEOF に記載されています。

```
SELECT IS_VARCHAR(r_nations[0].n_name) FROM region_nations;
 is_varchar
-------------
 true
(1 row)
```

```
SELECT r_nations[4].n_name FROM region_nations
WHERE CASE WHEN IS_INTEGER(r_nations[4].n_nationkey)
           THEN r_nations[4].n_nationkey::INTEGER = 15
           ELSE false END;
```

順

Amazon Redshift では、動的型が異なる値間の SUPER 比較を定義していません。文字列である SUPER 値は、数値である SUPER 値より小さくも大きくもありません。SUPER 列で ORDER BY 句を使用するために、Amazon Redshift では、ORDER BY 句を使用して SUPER 値をランク付けするときに観察されるさまざまなタイプ間の全順序を定義します。動的型間の順序は、ブール値、数値、文字列、配列、オブジェクトです。次の例では、さまざまなタイプの順序を示しています。

```
INSERT INTO region_nations VALUES
(100,'name1','comment1','AWS'),
(200,'name2','comment2',1),
(300,'name3','comment3',ARRAY(1, 'abc', null)),
(400,'name4','comment4',-2.5),
(500,'name5','comment5','Amazon');

SELECT r_nations FROM region_nations order by r_nations;

r_nations
----------------
 -2.5
 1
 "Amazon"
 "AWS"
 [1,"abc",null]
(5 rows)
```

ORDER BY 句の詳細については、「ORDER BY 句」を参照してください。

演算子と関数

Amazon Redshift では、SUPER データを使用して、演算子と関数を駆使して大規模なデータセットに対して高度な分析を実行できます。SUPER データの演算子と関数は、Amazon Redshift テーブルに保存されている半構造化データの複雑な分析と操作を可能にする SQL コンストラクトです。

以下のセクションでは、Amazon Redshift で SUPER データに演算子と関数を使用して、半構造化データの可能性を最大限に引き出すための構文、例、ベストプラクティスについて説明します。

算術演算子

SUPER 値は、動的型付けを使用して、すべての基本的な算術演算子 +、-、*、/、% をサポートしています。結果として得られるオペレーションのタイプは SUPER のままです。バイナリ演算子 + を除くすべての演算子について、入力オペランドは数値でなければなりません。それ以外の場合、Amazon Redshift は null を返します。小数点値と浮動小数点値の区別は、Amazon Redshift がこれらの演算子を実行し、動的型が変更されない場合でも保持されます。ただし、乗算と除算を使用すると、小数点以下の縮尺が変わります。算術オーバーフローは依然としてクエリエラーを引き起こしますが、null に変更されません。バイナリ演算子 + 入力が数字の場合は加算を行い、入力が文字列の場合は連結を行います。一方のオペランドが文字列で、もう一方のオペランドが数値の場合、結果は

null になります。次の例に示すように、SUPER 値が数値でない場合、単項接頭演算子 + および - は null を返します。

```
SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0]. o_orderkey / 10 AS math FROM
 customer_orders_lineitem;
            math
----------------------------
 1757958232200.1500
(1 row)
```

動的型付けにより、SUPER の小数値が異なるスケールを変えることができます。Amazon Redshift では、10 進数値が異なる静的型であるかのように扱われ、すべての数学演算が許可されます。Amazon Redshift は、オペランドのスケールに基づいて、結果のスケールを動的に計算します。オペランドの 1 つが浮動小数点数の場合、Amazon Redshift はもう 1 つのオペランドを浮動小数点数に昇格し、結果を浮動小数点数として生成します。

算術関数

Amazon Redshift は、SUPER 列に対して次の算術関数をサポートしています。入力が数値でない場合、null を返します。

- FLOOR。詳細については、「FLOOR 関数」を参照してください。
- CEIL および CEILING。詳細については、「CEILING (または CEIL) 関数」を参照してください。
- ROUND。詳細については、「ROUND 関数」を参照してください。
- TRUNC。詳細については、「TRUNC 関数」を参照してください。
- ABS。詳細については、「ABS 関数」を参照してください。

次の例では、算術関数を使用してデータをクエリします。

```
SELECT x, FLOOR(x), CEIL(x), ROUND(x)
FROM (
    SELECT (c_orders[0]. o_orderkey + 0.5) * c_orders[0].o_orderkey / 10 AS x
    FROM customer_orders_lineitem
    );

         x           |     floor      |     ceil       |     round
--------------------+----------------+----------------+---------------
 1389636795898.0500  | 1389636795898  | 1389636795899  | 1389636795898
```

ABS 関数は、FLOOR、CEIL が実行される間、入力小数のスケールを保持します。ROUND は、入力小数点のスケールを削除します。

配列関数

Amazon Redshift は、次の配列構成とユーティリティ関数の配列、array_concat、subarray。array_flatten、get_array_length、および split_to_array をサポートしています。

他の SUPER 値を含む ARRAY 関数を使用して、Amazon Redshift データ型の値から SUPER 配列を構築できます。次の例では、可変関数 ARRAY を使用しています。

```
SELECT ARRAY(1, c.c_custkey, NULL, c.c_name, 'abc') FROM customer_orders_lineitem c;
                                array
 -------------------------------------------------------
[1,8401,null,""Customer#000008401"",""abc""]
[1,9452,null,""Customer#000009452"",""abc""]
[1,9451,null,""Customer#000009451"",""abc""]
[1,8251,null,""Customer#000008251"",""abc""]
[1,5851,null,""Customer#000005851"",""abc""]
(5 rows)
```

次の例では、ARRAY_CONCAT 関数で配列の連結を使用しています。

```
SELECT ARRAY_CONCAT(JSON_PARSE('[10001,10002]'),JSON_PARSE('[10003,10004]'));

             array_concat
------------------------------------
 [10001,10002,10003,10004]
(1 row)
```

次の例では、入力配列のサブセットを返す SUBARRAY 関数で配列操作を使用しています。

```
SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);

   subarray
---------------
 ["c","d","e"]
(1 row))
```

次の例では、ARRAY_FLATTEN を使用して、複数のレベルの配列を単一の配列にマージします。

```
SELECT x, ARRAY_FLATTEN(x) FROM (SELECT ARRAY(1, ARRAY(2, ARRAY(3, ARRAY()))) AS x);

     x           | array_flatten
 ----------------+---------------
 [1,[2,[3,[]]]]  | [1,2,3]
(1 row)
```

配列関数 ARRAY_CONCAT および ARRAY_FLATTEN は、動的型付けのルールを使用します。入力が配列でない場合、エラーの代わりに null を返します。GET_ARRAY_LENGTH 関数は、オブジェクトまたは配列パスが与えられた SUPER 配列の長さを返します。

```
SELECT c_name
FROM customer_orders_lineitem
WHERE GET_ARRAY_LENGTH(c_orders) = (
    SELECT MAX(GET_ARRAY_LENGTH(c_orders))
    FROM customer_orders_lineitem
    );
```

次の例では、SPLIT_TO_ARRAY を使用して、文字列を文字列の配列に分割します。この関数は、任意のパラメータとして区切り文字を使用します。区切り文字がない場合、デフォルトはコンマです。

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');

   split_to_array
---------------------
 ["12","345","6789"]
(1 row)
```

SUPER 設定

特定のシナリオ用に SUPER データを設定できます。以下のセクションでは、データ形式の要件に基づいて適切な SUPER 設定を選択して適用する方法について詳しく説明します。

トピック

- SUPER の lax および strict モード
- 大文字、および大小文字が混在するフィールド名または属性を持つ JSON フィールドへのアクセス
- SUPER の解析オプション

SUPER の lax および strict モード

SUPER データをクエリすると、パス式が実際の SUPER データ構造と一致しないことがあります。オブジェクトまたは配列の要素が存在しないメンバーにアクセスしようとすると、クエリがデフォルトの lax モードで実行されると、Amazon Redshift は NULL 値を返します。strict モードでクエリを実行すると、Amazon Redshift はエラーを返します。次のセッションパラメータを設定して、lax モードをオンまたはオフに設定できます。

次の例では、セッションパラメータを使用して lax モードを有効にします。

```
SET navigate_super_null_on_error=ON;  --default lax mode for navigation

SET cast_super_null_on_error=ON;  --default lax mode for casting

SET parse_super_null_on_error=OFF;  --default strict mode for ingestion
```

大文字、および大小文字が混在するフィールド名または属性を持つ JSON フィールドへのアクセス

JSON 属性名が大文字、または大小文字が混在している場合は、大文字と小文字を区別して SUPER 型構造をナビゲートできるようにする必要があります。そのためには、`enable_case_sensitive_identifier` を TRUE に設定して、大文字、および大小文字が混在する属性名を二重引用符で囲みます。`enable_case_sensitive_super_attribute` を TRUE に設定することもできます。この場合、クエリで大文字および大小文字が混在する属性名を二重引用符で囲まずに使用することができます。

次の例では、`enable_case_sensitive_identifier` にクエリデータを設定する方法を説明しています。

```
SET enable_case_sensitive_identifier to TRUE;

-- Accessing JSON attribute names with uppercase and mixedcase names
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 Name | price
------+-------
 "TV" | 345
```

```
(1 row)

RESET enable_case_sensitive_identifier;

-- After resetting the above configuration, the following query accessing JSON
 attribute names with uppercase and mixedcase names should return null (if in lax
 mode).
SELECT json_table.data."ITEMS"."Name",
       json_table.data."price"
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
------+-------
      | 345
(1 row)
```

次の例では、enable_case_sensitive_super_attribute にクエリデータを設定する方法を説明しています。

```
SET enable_case_sensitive_super_attribute to TRUE;
 -- Accessing JSON attribute names with uppercase and mixedcase names

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;

 name | price
 -----+-------
 "TV" | 345
(1 row)

 RESET enable_case_sensitive_super_attribute;

 -- After resetting enable_case_sensitive_super_attribute, the query now returns NULL
 for ITEMS.Name (if in lax mode).

SELECT json_table.data.ITEMS.Name,
       json_table.data.price
FROM
  (SELECT json_parse('{"ITEMS":{"Name":"TV"}, "price": 345}') AS data) AS json_table;
```

```
 name | price
 -----+-------
      | 345
(1 row)
```

SUPER の解析オプション

JSON_PARSE 関数を使用して JSON 文字列を SUPER 値に解析する場合、特定の制限が適用されます。

- 同じ属性名を同じオブジェクトに表示することはできませんが、ネストされたオブジェクトには表示できます。`json_parse_dedup_attributes` 設定オプションを使用すると、JSON_PARSE はエラーを返す代わりに、重複属性の最後の出現のみを保持できます。
- 文字列値は、システムの最大 varchar サイズである 65535 バイトを超えることはできません。`json_parse_truncate_strings` 設定オプションを使用すると、JSON_PARSE() は、エラーを返さずに、この制限より長い文字列を自動的に切り捨てることができます。この動作は文字列値のみに影響し、属性名には影響しません。

JSON_PARSE 関数の詳細については、「JSON_PARSE 関数」を参照してください。

次の例は、`json_parse_dedup_attributes` 設定オプションを重複する属性に対してエラーを返すデフォルトの動作に設定する方法を示しています。

```
SET json_parse_dedup_attributes=OFF;  --default behavior of returning error instead of
 de-duplicating attributes
```

次の例は、`json_parse_truncate_strings` 設定オプションをこの制限を超える長さの文字列に対してエラーを返すデフォルトの動作に設定する方法を示しています。

```
SET json_parse_truncate_strings=OFF;  --default behavior of returning error instead of
 truncating strings
```

制約事項

Amazon Redshift では、SUPER データ型を使用して、JSON、Avro、Ion などの半構造化データを保存およびクエリできます。SUPER データ型の制限とは、Amazon Redshift でこのデータ型を使用する場合の制約と境界を指します。以下のセクションでは、最大サイズ、ネストレベル、半構造化データでサポートされるデータ型など、SUPER データ型の特定の制限について詳しく説明します。

- SUPER 列をディストリビューションキーまたはソートキーとして定義することはできません。
- 各 SUPER オブジェクトは、最大 16 MB のデータを保持できます。
- SUPER オブジェクト内の個々の値は、Amazon Redshift 内で対応しているデータ型での最大長に制限されます。たとえば、SUPER にロードされる 1 つの文字列値は、VARCHAR の最大長 65535 バイトに制限されます。
- SUPER 列に対して部分的な更新または変換オペレーションを実行することはできません。
- 右結合または完全外部結合では、SUPER データ型とそのエイリアスを使用することはできません。
- SUPER データ型は、インバウンドまたはアウトバウンドのシリアル化形式として XML をサポートしていません。
- ネストを解除するためにテーブル変数を参照するサブクエリ (相関関係があるかどうかにかかわらず) の FROM 句では、クエリはその親テーブルのみを参照でき、他のテーブルは参照できません。
- キャストの制限事項

  SUPER 値は、次の例外を除いて、他のデータ型との間でキャストできます。

  - Amazon Redshift は、スケール 0 の整数と小数点を区別しません。
  - スケールがゼロでない場合、SUPER データ型は他の Amazon Redshift データ型と同じ動作をします。ただし、Amazon Redshift は、次の例に示すように、スーパー関連のエラーを null に変換する点が異なります。

```
SELECT 5::bool;
 bool
-------
 True
(1 row)

SELECT 5::decimal::bool;
ERROR:  cannot cast type numeric to boolean

SELECT 5::super::bool;
 bool
-------
 True
(1 row)

SELECT 5.0::bool;
ERROR:  cannot cast type numeric to boolean
```

```
SELECT 5.0::super::bool;
 bool
-------
(1 row)
```

- Amazon Redshift は、日付と時刻の型を SUPER データ型にキャストしません。Amazon Redshift では、次の例に示すように、SUPER データ型から日付と時刻のデータ型のみをキャストできます。

```
SELECT o.o_orderdate FROM customer_orders_lineitem c,c.c_orders o;
  order_date
------------------
 "2001-09-08"
(1 row)


SELECT JSON_TYPEOF(o.o_orderdate) FROM customer_orders_lineitem c,c.c_orders o;
 json_typeof
-----------------
 string
(1 row)


SELECT o.o_orderdate::date FROM customer_orders_lineitem c,c.c_orders o;
 order_date
----------------
 2001-09-08
(1 row)


--date/time cannot be cast to super
SELECT '2019-09-09'::date::super;
ERROR:  cannot cast type date to super
```

- 非スカラー値 (オブジェクトと配列) から文字列にキャストすると、NULL が返されます。これらの非スカラー値を適切にシリアル化するには、それらをキャストしないでください。代わりに、`json_serialize` を使用して非スカラー値をキャストします。`json_serialize` 関数は varchar を返します。次の最初の例に示すように、Amazon Redshift は暗黙的にシリアル化されるため、通常、非スカラー値を varchar にキャストする必要はありません。

```
SELECT r_nations FROM region_nations WHERE r_regionkey=300;
```

```
   r_nations
----------------
 [1,"abc",null]
(1 row)

SELECT r_nations::varchar FROM region_nations WHERE r_regionkey=300;
 r_nations
-------------
(1 row)

SELECT JSON_SERIALIZE(r_nations) FROM region_nations WHERE r_regionkey=300;
 json_serialize
-----------------
 [1,"abc",null]
(1 row)
```

  - 大文字と小文字を区別しないデータベースの場合、Amazon Redshift は SUPER データ型をサポートしません。大文字と小文字を区別しない列の場合、Amazon Redshift はそれらを SUPER 型にキャストしません。したがって、Amazon Redshift では、大文字と小文字が区別されずキャストをトリガーしている列との間で、やり取りを行う SUPER 列はサポートされません。
- Amazon Redshift は、サブクエリで IN 関数の外部テーブルまたは左側 (LHS) を検出しないサブクエリでは、RANDOM ( ) や TIMEOFDAY ( ) などの揮発性関数をサポートしません。

SUPER データ型とマテリアライズドビュー

Amazon Redshift では、SUPER データ型を使用して、マテリアライズドビューのパフォーマンスと柔軟性を強化できます。SUPER データ型を使用すると、ベーステーブルからの列のスーパーセットをマテリアライズドビューに保存し、ベーステーブルを結合せずにマテリアライズドビューを直接クエリできます。以下のセクションでは、Amazon Redshift で SUPER データ型を使ってマテリアライズドビューを作成し、使用する方法を説明します。

Amazon Redshift は、マテリアライズドビューの機能を拡張し、マテリアライズドビューで SUPER データ型および PartiQL と連携します。SQL クエリと PartiQL クエリは、増分マテリアライズドビューを使用して事前計算できます。マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

スキーマレスおよび半構造化データを SUPER に保存したら、PartiQL マテリアライズドビューを使用してデータを確認し、マテリアライズドビューでシュレッディング処理を行えます。

PartiQL クエリの高速化

マテリアライズドビューを使用して、SUPER 列内の階層データをナビゲートするか、またはネストを解除する PartiQL クエリを高速化できます。1 つ以上のマテリアライズドビューを作成して、SUPER 値を複数の列に分割し、Amazon Redshift 分析クエリの列構造を利用します。したがって、クエリはマテリアライズドビューを使用します。

マテリアライズドビューは、基本的にネストされたデータを抽出して正規化します。正規化のレベルは、SUPER データを従来の列指向データに変換するためにどれだけの労力を費やしたかによって異なります。

マテリアライズドビューを使用した SUPER 列のシュレッダー処理

Amazon Redshift では、マテリアライズドビューを使用してデータを SUPER 列にシュレッダー処理することで、クエリのパフォーマンスを向上させ、ストレージ要件を減らすことができます。シュレッダー処理とは、半構造化 JSON や XML などの複雑なデータ型を、より小さくフラットな列に分割するプロセスを指します。SUPER 列は、シュレッダー処理されたデータを高速スキャンできるよう最適化された特殊な形式の列型ストレージです。

以下のセクションでは、Amazon Redshift でマテリアライズドビューを使用してデータを SUPER 列にシュレッダー処理する手順と考慮事項について説明します。

次の例は、ネストされたデータをシュレッディングするマテリアライズドビューを示しています。結果の列は引き続き SUPER データ型です。

```
SELECT c.c_name, o.o_orderstatus
FROM customer_orders_lineitem c, c.c_orders o;
```

以下の例は、シュレッドされたデータから従来の Amazon Redshift スカラー列を作成するマテリアライズドビューを示しています。

```
SELECT c.c_name, c.c_orders[0].o_totalprice
FROM customer_orders_lineitem c;
```

単一のマテリアライズドビュー super_mv を作成して、両方のクエリを高速化できます。

最初のクエリに応答するには、属性 o_orderstatus を実体化する必要があります。ネストされたナビゲーションやネスト解除を伴わないため、属性 c_name を省略できます。また、ベーステーブルを

マテリアライズドビューと結合できるようにするには、customer_orders_lineitem の属性 c_custkey をマテリアライズドビューに含める必要があります。

2 番目のクエリに答えるには、c_orders の属性 o_totalprice と配列インデックス o_idx も実体化する必要があります。したがって、c_orders のインデックス 0 にアクセスすることができます。

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey) AS (
  SELECT c_custkey, o.o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

マテリアライズドビュー super_mv の属性 o_orderstatus および o_totalprice は SUPER です。

マテリアライズドビュー super_mv は、ベーステーブル customer_orders_lineitem が変更されると、増分的に更新されます。

```
REFRESH MATERIALIZED VIEW super_mv;
INFO: Materialized view super_mv was incrementally updated successfully.
```

最初の PartiQL クエリを通常の SQL クエリとして書き換えるには、次のように customer_orders_lineitem と super_mv を結合します。

```
SELECT c.c_name, v.o_orderstatus
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey;
```

同様に、2 番目の PartiQL クエリを書き換えることができます。次の例では、o_idx = 0 のフィルターを使用します。

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_idx = 0;
```

CREATE MATERIALIZED VIEW コマンドで、ディストリビューションキーとして c_custkey を指定し、super_mv のソートキーを指定します。Amazon Redshift は、c_custkey が customer_orders_lineitem のディストリビューションキーおよびソートキーでもあると仮定して、効率的なマージ結合を実行します。そうでない場合は、c_custkey を customer_orders_lineitem のソートキーおよびディストリビューションキーとして次のように指定できます。

```
ALTER TABLE customer_orders_lineitem
ALTER DISTKEY c_custkey, ALTER SORTKEY (c_custkey);
```

EXPLAIN ステートメントを使用して、書き換えられたクエリに対して Amazon Redshift がマージ結合を実行することを確認します。

```
EXPLAIN
      SELECT c.c_name, v.o_orderstatus
      FROM customer_orders_lineitem c JOIN super_mv v ON c.c_custkey = v.c_custkey;

      QUERY PLAN

 ------------------------------------------------------------------------------------------------------
      XN Merge Join DS_DIST_NONE  (cost=0.00..34701.82 rows=1470776 width=27)
      Merge Cond: ("outer".c_custkey = "inner".c_custkey)
      ->  XN Seq Scan on mv_tbl__super_mv__0 derived_table2  (cost=0.00..14999.86
 rows=1499986 width=13)
      ->  XN Seq Scan on customer_orders_lineitem c  (cost=0.00..999.96 rows=99996
 width=30)
      (4 rows)
```

細断されたデータからの Amazon Redshift スカラー列の作成

SUPER に保存されているスキーマレスデータは、Amazon Redshift のパフォーマンスに影響を与える可能性があります。例えば、範囲が制限されたスキャンではゾーンマップを効果的に使用できないため、述語や結合条件をフィルタリングします。ユーザーと BI ツールは、データの従来の表示としてマテリアライズドビューを使用し、分析クエリのパフォーマンスを向上させることができます。

次のクエリは、マテリアライズドビュー super_mv をスキャンし、o_orderstatus でフィルタリングします。

```
SELECT c.c_name, v.o_totalprice
FROM customer_orders_lineitem c
JOIN super_mv v ON c.c_custkey = v.c_custkey
WHERE v.o_orderstatus = 'F';
```

stl_scan を調べて、Amazon Redshift が o_orderstatus の範囲制限スキャンでゾーンマップを効果的に使用できないことを確認します。

```
SELECT slice, is_rrscan FROM stl_scan
```

```
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | f
     1 | f
     5 | f
     4 | f
     2 | f
     3 | f
(6 rows)
```

次の例では、マテリアライズドビュー super_mv を使用して、細断されたデータからスカラー列を作成します。この場合、Amazon Redshift は o_orderstatus を SUPER から VARCHAR にキャストします。さらに、super_mv のソートキーとして o_orderstatus を指定します。

```
CREATE MATERIALIZED VIEW super_mv distkey(c_custkey) sortkey(c_custkey, o_orderstatus)
 AS (
  SELECT c_custkey, o.o_orderstatus::VARCHAR AS o_orderstatus, o.o_totalprice, o_idx
  FROM customer_orders_lineitem c, c.c_orders o AT o_idx
);
```

クエリを再実行した後、Amazon Redshift がゾーンマップを使用できることを確認します。

```
SELECT v.o_totalprice
FROM super_mv v
WHERE v.o_orderstatus = 'F';
```

次のように、範囲が制限されたスキャンでゾーンマップが使用されるようになったことを確認できます。

```
SELECT slice, is_rrscan FROM stl_scan
WHERE query = pg_last_query_id() AND perm_table_name LIKE '%super_mv%';

 slice | is_rrscan
-------+-----------
     0 | t
     1 | t
     2 | t
     3 | t
     4 | t
```

```
      5 | t
(6 rows)
```

マテリアライズドビューでの SUPER データ型の使用に関する制限事項

Amazon Redshift では、SUPER データ型を使用してマテリアライズドビューを作成し、複雑なクエリを事前に計算して、クエリのパフォーマンスを向上させることができます。SUPER データ型を使用すると、クエリの結果セットをフラットなデータ構造として保存できるため、アクセスと処理が高速化されます。

以下のセクションでは、Amazon Redshift でマテリアライズドビューを使用して、SUPER データ型を使用するための制限とベストプラクティスについて詳しく説明します。

Amazon Redshift のマテリアライズドビューには、PartiQL または SUPER に関する特定の制限はありません。

マテリアライズドビュー作成時の一般的な SQL 制約事項については、「制約事項」を参照してください。

マテリアライズドビューの増分更新に関する一般的な SQL の制限については、「増分更新の制約事項」を参照してください。

機械学習

Amazon Redshift 機械学習 (Amazon Redshift ML) は、どの技術レベルのアナリストやデータサイエンティストでも、機械学習のテクノロジーを簡単に使用できる堅牢なクラウドベースのサービスです。Amazon Redshift ML はモデルを使用して結果を生成します。モデルは以下のように使用できます。

- Amazon Redshift へのデータ入力に関連付けられた、モデルとメタデータをトレーニングするデータを指定します。次に、Amazon Redshift ML は、入力データのパターンをキャプチャするモデルを Amazon SageMaker に作成します。モデルに独自のデータを使用することで、Amazon Redshift ML を使用して、顧客の離反予測、ライフタイムバリュー、収益予測などのデータの傾向を特定できます。これらのモデルを使用して、追加のコストを発生させることなく、新しい入力データの予測を生成できます。
- Claude や Amazon Titan など、Amazon Bedrock が提供する基盤モデル (FM) のいずれかを使用できます。Amazon Bedrock を使用すると、大規模言語モデル (LLM) のパワーと Amazon Redshift の分析データを少ない手順で組み合わせることができます。外部の大規模言語モデル (LLM) を使用すると、Amazon Redshift を使用して、データに対して自然言語処理 (NLP) を実行できます。NLP は、テキスト生成、感情分析、翻訳などのアプリケーションに使用できます。Amazon Bedrock と Amazon Redshift の連携については、「Amazon Redshift ML と Amazon Bedrock の統合」を参照してください。

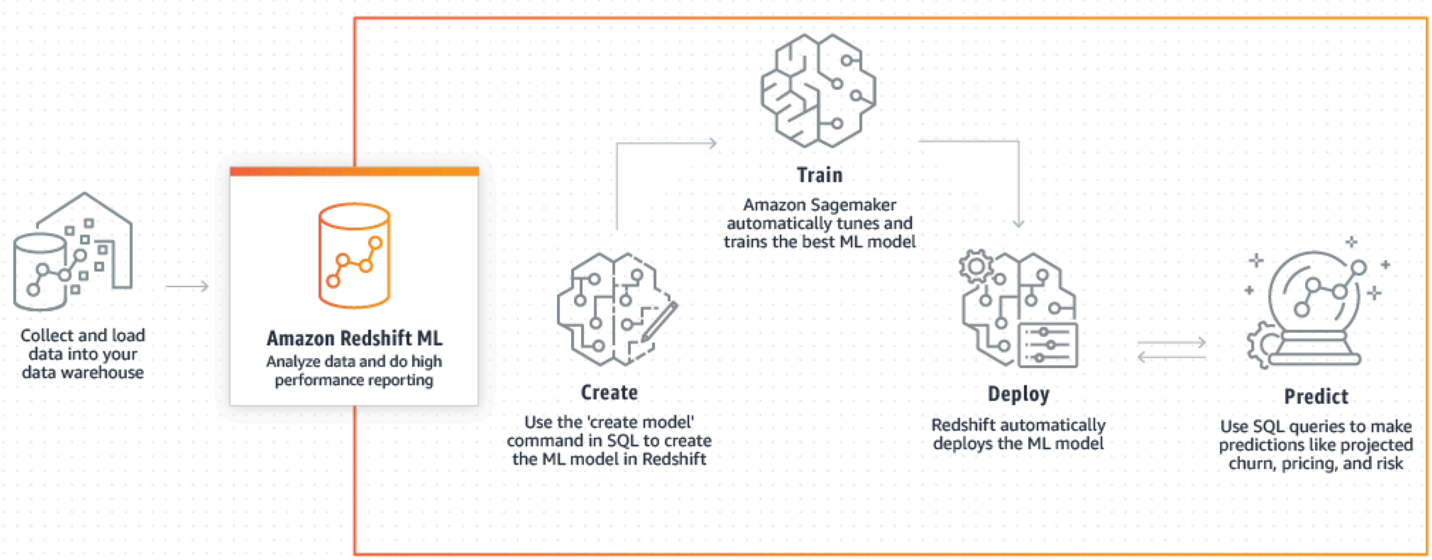
> **Note**
>
> サービス改善のためのデータ使用をオプトアウトする
> Amazon Bedrock モデルを使用していて、サービス改善のために AWS でデータを処理しない場合は、Amazon Bedrock のオプトアウトポリシーを有効にする必要があります。

> **Note**
>
> LLM は、不正確または不完全な情報を生成する可能性があります。LLM が生成する情報を検証して、正確性と完全性を確認することをお勧めします。

Amazon Redshift 機械学習と Amazon SageMaker の連携方法

Amazon Redshift は Amazon SageMaker Autopilot と連携して、最適なモデルを自動的に取得し、Amazon Redshift で予測機能を利用できるようにします。

次の図は、Amazon Redshift 機械学習の仕組みを示しています。



一般的なワークフローは次のとおりです。

1. Amazon Redshift は、トレーニングデータを Amazon S3 にエクスポートします。
2. Amazon SageMaker Autopilot は、トレーニングデータを前処理します。前処理では、欠損値の補完などの重要な機能を担います。特定の列がカテゴリ (郵便番号など) であることを認識し、トレーニング用に適切にフォーマットし、他の多くのタスクを実行します。トレーニングデータセットに適用するのに最適なプリプロセッサを選択すること自体が問題であり、Amazon SageMaker Autopilot はそのソリューションを自動化します。
3. Amazon SageMaker Autopilot は、最も正確な予測でモデルを提供するアルゴリズムとアルゴリズムのハイパーパラメータを見つけます。
4. Amazon Redshift は、予測関数を SQL 関数として Amazon Redshift クラスターに登録します。
5. CREATE MODEL ステートメントを実行すると、Amazon Redshift はトレーニングに Amazon SageMaker を使用します。したがって、モデルのトレーニングには関連するコストがかかります。これは、Amazon SageMaker に関する個別の 1 項目として、AWS 請求書上に記載されています。また、トレーニングデータを保存するために Amazon S3 で使用されるストレージについてもお支払いいただきます。Redshift クラスター上でコンパイルと実行が可能な、CREATE MODEL で作成されたモデルを使用した推論に対しては課金されません。Amazon Redshift ML を使用する場合 Amazon Redshift の追加料金は発生しません。

トピック

Machine Learning の概要

Amazon Redshift では、機械学習機能を活用して、データから貴重なインサイトを得ることができます。この機械学習 (ML) の概要では、ML モデルのトレーニングとデプロイのためにデータを調査、視覚化、準備する方法について説明します。以下のセクションでは、Amazon Redshift ML を使用し、機械学習を通じてデータの可能性を引き出すプロセスについて説明します。

Amazon Redshift ML を使用すると、SQL ステートメントを使用して機械学習モデルをトレーニングし、予測のために SQL クエリでそれらを呼び出すことができます。

Amazon Redshift ML の使用方法を学ぶために、動画 Amazon Redshift ML をご覧ください。

Redshift クラスターまたは Serverless ワークグループをセットアップするための前提条件、アクセス許可、Amazon Redshift ML を使用するための所有権については、以下のセクションを参照してください。これらのセクションでは、Amazon Redshift ML での簡単なトレーニングと予測の仕組みについても説明します。

機械学習による問題解決のしくみ

機械学習モデルは、トレーニングデータでパターンを見つけ、これらのパターンを新しいデータに適用することで予測を生成します。機械学習では、データを最もよく説明するパターンを学習することで、これらのモデルをトレーニングします。その後、モデルを使用して新しいデータを予測します (推論とも呼ぶ)。機械学習は通常、パラメータを変更し、トレーニングデータを改善することによって予測の精度を継続的に向上させることができる反復的なプロセスです。データが変更されると、新しいデータセットを使用して新しいモデルを再トレーニングします。

さまざまなビジネス目標に対応するために、さまざまな基本的な機械学習アプローチがあります。

Amazon Redshift 機械学習での教師あり学習

Amazon Redshift は、高度なエンタープライズ分析への最も一般的なアプローチである教師あり学習をサポートしています。教師あり学習は、確立されたデータセットがあり、特定の入力データがさまざまなビジネス成果を予測する方法を理解している場合に、好ましい機械学習アプローチです。これらの結果は、ラベルと呼ばれることもあります。特に、データセットは、特徴 (入力) とターゲット (出力) で構成される属性を持つテーブルです。たとえば、過去と現在の顧客の年齢と郵便番号を記載したテーブルがあるとします。また、現在の顧客に対しては true、メンバーシップを停止した顧客に対しては false という「アクティブ」フィールドもあるとします。教師あり機械学習の目的は、ターゲットが「False」である顧客によって表される、カスタマーチャーンにつながる年齢と郵便番号のパターンを見つけることです。このモデルを使用して、メンバーシップの一時停止など、解約する可能性が高く、保持インセンティブを提供する可能性のある顧客を予測できます。

Amazon Redshift は、回帰、バイナリ分類、およびマルチクラス分類を含む教師あり学習をサポートします。回帰とは、顧客の総支出など、連続値を予測する問題を指します。バイナリ分類は、顧客が解約するかどうかを予測するなど、2 つの結果のうちの 1 つを予測する問題を指します。マルチクラス分類とは、顧客が関心を持つ可能性のある項目を予測するなど、多くの結果の 1 つを予測する問題を指します。データアナリストとデータサイエンティストは、これを使用して教師あり学習を実行し、予測、パーソナライゼーション、顧客解約の予測などの問題に取り組むことができます。また、締結しそうな売買の予測、収益予測、不正検出、顧客の生涯価値予測などの問題のためにも、教師あり学習を使用できます。

Amazon Redshift 機械学習での教師なし学習

教師なし学習では、機械学習アルゴリズムを使用して、ラベル付けされていないトレーニングデータの分析とグループ化を行います。このアルゴリズムでは、隠れたパターンやグループを検出します。教師なし学習の目標は、データの基礎となる構造や分布をモデル化して、データの詳細を明らかにすることです。

Amazon Redshift は、教師なし学習での問題に対応するために、K-Means によるクラスタリングアルゴリズムをサポートしています。このアルゴリズムは、データ内でグループを検出する際にクラスタリングで発生する問題を解決します。K-Means アルゴリズムは、データ内にある離散群の検出を試みます。未分類のデータについては、その類似点と相違点に基づいたグループ分けと分割が行われます。グループ化によって、K-Means アルゴリズムは最適な重心を反復的に決定し、各メンバーを最も近い重心に割り当てます。最も近い重心が同じである各メンバーは、同じグループに属します。1 つのグループには、可能な限り互いに類似した (かつ、他のグループのメンバーとは可能な限り相違する) メンバーが集められます。例えば、パンデミックの影響を受けた都市を分類したり、消

費者製品の人気に基づいて都市を分類したりするために、K-Means クラスタリングアルゴリズムを使用できます。

K-Means アルゴリズムを使用する場合は、入力として k を指定します。これにより、データ内で検索しようとするクラスターの数を指定します。このアルゴリズムは k 個の重心を出力します。各データポイントは、k 個のクラスターの内で最も近方の 1 つに属します。各クラスターは、その重心によって表現されます。1 つの重心は、クラスターの多次元平均と考えることができます。K-Means アルゴリズムは、クラスター間の距離を比較して、互いにどの程度異なっているかを評価します。一般に、距離が大きいほどクラスター間の相違が大きいことを意味します。

K-Means では、モデルの特徴が同じスケールにとどまることや、信頼性の高い結果が求められるため、データの前処理は重要です。Amazon Redshift では CREATE MODEL ステートメント用として、StandardScaler、MinMax、NumericPassthrough など、K-Means プリプロセッサをいくつかサポートしています。K-Means に前処理を適用しない場合は、トランスフォーマーとして明示的に NumericPassthrough を選択します。K-Means の詳細については、「K-MEANS パラメータによる CREATE MODEL」を参照してください。

K-Means クラスタリングを使用して教師なしトレーニングを実行する方法を学ぶには、動画「Unsupervised training with K-Means clustering」(K-Means クラスタリングを使用した教師なしトレーニング) を視聴することができます。

Amazon Redshift 機械学習の用語と概念

以下の用語は、Amazon Redshift ML の概念の一部を説明するために使用されます。

- Amazon Redshift の機械学習は、1 つの SQL コマンドでモデルをトレーニングします。Amazon Redshift ML と Amazon SageMaker は、すべてのデータ変換、アクセス許可、リソース使用量、および適切なモデルの検出を管理します。
- トレーニングは、Amazon Redshift が指定されたデータのサブセットをモデルに実行することにより、機械学習モデルを作成するフェーズです。Amazon Redshift は、Amazon SageMaker でトレーニングジョブを自動的に起動し、モデルを生成します。
- 予測 (推論とも言う) とは、Amazon Redshift SQL クエリでモデルを使用して結果を予測することです。推論時に、Amazon Redshift はより大きなクエリの一部としてモデルベースの予測関数を使用して予測を生成します。予測は Amazon Redshift クラスターでローカルに計算されるため、高いスループットと低レイテンシーを実現し、追加コストなしで利用できます。
- 独自のモデルを持参 (BYOM) すると、Amazon Redshift の外部でトレーニングされたモデルを Amazon SageMaker で使用して、Amazon Redshift のローカルでデータベース内の推論を行うこ

とができます。Amazon Redshift ML では、ローカル推論での BYOM の使用がサポートされています。

- ローカル推論は、モデルが Amazon SageMaker で事前トレーニングされ、Amazon SageMaker Neo によってコンパイルされ、Amazon Redshift ML でローカライズされるときに使用します。ローカル推論でサポートされているモデルを Amazon Redshift にインポートするには、CREATE MODEL コマンドを使用します。Amazon Redshift は Amazon SageMaker Neo を呼び出して、事前にトレーニングされた SageMaker モデルをインポートします。そこでモデルをコンパイルし、コンパイルされたモデルを Amazon Redshift にインポートします。ローカル推論を使用して、高速化と低コストを実現します。
- リモート推論は、Amazon Redshift が SageMaker にデプロイされたモデルエンドポイントを呼び出すときに使用されます。リモート推論では、すべてのタイプのカスタムモデルと深層学習モデル (Amazon SageMaker で構築およびデプロイした TensorFlow モデルなど) を呼び出すための柔軟性が提供されます。

また、次の点も重要です。

- Amazon SageMaker は、フルマネージド型の機械学習サービスです。Amazon SageMaker では、データサイエンティストとデベロッパーが簡単にモデルの構築とトレーニングを行うことができ、稼働準備が整ったホスト環境に直接デプロイできます。Amazon SageMaker の詳細については、Amazon SageMaker デベロッパーガイドの Amazon SageMaker とはを参照してください。
- Amazon SageMaker Autopilot は、データに基づいて、分類または回帰に最適な機械学習モデルを自動的にトレーニングおよびチューニングする機能セットです。ユーザーは、このための完全な制御性と可視性を維持します。Amazon SageMaker Autopilot では、表形式の入力データがサポートされます。Amazon SageMaker Autopilot では、自動的なデータのクリーニングと前処理に加え、線形回帰、バイナリ分類、マルチクラス分類などからのアルゴリズムの自動選択機能が利用できます。また、ハイパーパラメータの自動最適化 (HPO)、ディストリビューショントレーニング、自動インスタンス、クラスターサイズの選択もサポートしています。Amazon SageMaker Autopilot の詳細については、Amazon SageMaker デベロッパーガイドから Amazon SageMaker Autopilot を使用したモデル開発の自動化を参照してください。
- Amazon Bedrock は、AI21 Labs、Anthropic、Cohere、Meta、Mistral AI、Stability AI、Amazon などの主要な AI 企業から、生成 AI アプリケーションの構築に必要な幅広い機能とともに、高性能の基盤モデル (FM) を単一の API で提供するフルマネージドサービスです。

初心者やエキスパート向けの Machine Learning

Amazon Redshift では、機械学習 (ML) 機能を活用して、ユーザーが初心者であるか ML の専門家であるかにかかわらず、データからインサイトを得ることができます。機械学習は、広範な ML の専門知識や複雑なデータエンジニアリングを必要とすることなく、SQL コマンドを使用して ML モデルを作成、トレーニング、デプロイできる Amazon Redshift の機能です。

以下のセクションでは、Amazon Redshift でデータが持つすべての可能性を引き出すための、機械学習の活用プロセスについて説明します。

Amazon Redshift ML では、1 つの SQL CREATE MODEL コマンドを使用してモデルをトレーニングできます。CREATE MODEL コマンドは、使い慣れた SQL 構造を使用してモデルベースの予測を生成するために Amazon Redshift が使用するモデルを作成します。

Amazon Redshift ML は、機械学習、ツール、言語、アルゴリズム、API の専門知識がない場合に特に役立ちます。Amazon Redshift ML を使用すると、外部の機械学習サービスとの統合に必要な、差別化されていない重労働を実行する必要がありません。Amazon Redshift を使用すると、データのフォーマットと移動および許可コントロールの管理、ならびにカスタム統合やワークフローそしてスクリプトの構築などにかかる時間を節約できます。一般的な機械学習アルゴリズムを簡単に使用して、トレーニングから予測まで頻繁に繰り返す必要があるトレーニングのニーズを簡素化できます。Amazon Redshift は自動的に最適なアルゴリズムを検出し、問題に最適なモデルをチューニングします。Amazon Redshift からデータを移動したり、別のサービスとやり取りして料金を支払ったりすることなく、Amazon Redshift クラスター内から予測を行うことができます。

Amazon Redshift ML は、データアナリストとデータサイエンティストによる機械学習の使用をサポートします。また、知識を持つ機械学習の専門家であれば、指定した側面のみを使用するように、CREATE MODEL ステートメントに指示することもできます。これにより、CREATE MODEL が最適な候補を検索するためや、モデルの精度を向上させるために必要な時間も短縮できます。

CREATE MODEL ステートメントは、トレーニングジョブにパラメータを指定する方法に柔軟性を加えます。この柔軟性により、機械学習の初心者やエキスパートのどちらでも、好みのプリプロセッサ、アルゴリズム、問題の種類、またはハイパーパラメータを選択できるようになります。たとえば、顧客離れに関心のあるユーザーは、顧客離れに適した問題の種類として、バイナリ分類を CREATE MODEL ステートメントに対し指定できます。次に、CREATE MODEL ステートメントは、最適なモデルの検索をバイナリ分類モデルに絞り込みます。ユーザーが問題の種類を選択した場合でも、CREATE MODEL ステートメントで使用できるオプションはたくさんあります。たとえば、CREATE MODEL は最適な前処理変換を検出して適用し、最適なハイパーパラメータの設定を検出します。

Amazon Redshift 機械学習は、Amazon SageMaker Autopilot を使用して最適なモデルを自動的に見つけることで、トレーニングを容易にします。その背後で Amazon SageMaker Autopilot は、提供されたデータに基づいて、最適な機械学習モデルのを自動的なにトレーニングとチューニングを実行します。その後、Amazon SageMaker Neo はトレーニングモデルをコンパイルし、Redshift クラスターで予測が行えるようにします。トレーニング済みのモデルを使用して機械学習推論クエリを実行する場合には、クエリで Amazon Redshift の超並列処理の機能が使用可能になります。同時に、このクエリでは機械学習ベースの予測を使用できます。

- 機械学習の初心者であっても、プリプロセッサ、アルゴリズム、ハイパーパラメータなどの機械学習のさまざまな側面に関する一般的な知識がある場合には、適用する側面を指定しながら CREATE MODEL ステートメントを使用することができます。次に、CREATE MODEL が最適な候補を見つけるために必要な時間を短縮したり、モデルの精度を向上させたりすることができます。また、問題の種類や目的などの追加のドメイン知識を導入することで、予測のビジネス価値を高めることもできます。たとえば、顧客解約シナリオでは、「顧客がアクティブではない」という結果がまれである場合、精度目標よりも F1 目標が優先されることがよくあります。高精度モデルでは、常に「顧客がアクティブ」であると予測される可能性があるため、精度は高くなりますが、ビジネス価値はほとんどありません。F1 目標の詳細については、Amazon SageMaker API リファレンスの AutoMLJobObjective を参照してください。

  CREATE MODEL ステートメントの基本オプションの詳細については、「単純な CREATE MODEL」を参照してください。
- 機械学習の上級者として、特定の (すべてではない) 特徴による問題の種類とプリプロセッサを指定できます。この場合 CREATE MODELは、指定された側面に関し与えられた提案に従います。同時に CREATE MODEL では、その他の特徴の処理に最適なプリプロセッサや、最適なハイパーパラメータの検出も継続して実行します。トレーニングパイプラインの 1 つ以上の側面を制限するための詳細方法については、「ユーザーガイダンス付きの CREATE MODEL」を参照してください。
- 機械学習のエキスパートとして、トレーニングとハイパーパラメータのチューニングを完全に制御できます。この場合ユーザーがすべての選択を行うため、 CREATE MODEL ステートメントでは、最適なプリプロセッサ、アルゴリズム、ハイパーパラメータの検出は試みられません。AUTO OFF で CREATE MODEL ステートメントを使用する方法については、「AUTO OFF 付きの CREATE XGBoost モデル」を参照してください。
- データエンジニアは、事前にトレーニングされた XGBoost モデルを Amazon SageMaker に取り込み、ローカル推論を行うために Amazon Redshift にインポートできます。独自のモデルを持参 (BYOM) すると、Amazon Redshift の外部でトレーニングされたモデルを Amazon SageMaker で使用して、Amazon Redshift のローカルでデータベース内の推論を行うことができます。Amazon Redshift 機械学習は、ローカルまたはリモートの推論で BYOM の使用をサポートしています。

CREATE MODEL ステートメントを使用してローカルまたはリモートの推論を行う方法の詳細については、「独自のモデルを持参 (BYOM) - ローカル推論」を参照してください。

Amazon Redshift ML ユーザーは、次のオプションのいずれかを選択して、モデルをトレーニングおよびデプロイできます。

- 問題の種類については、「ユーザーガイダンス付きの CREATE MODEL」を参照してください。
- 目的については、「ユーザーガイダンス付きの CREATE MODEL」または「AUTO OFF 付きの CREATE XGBoost モデル」を参照してください。
- モデルタイプについては、「AUTO OFF 付きの CREATE XGBoost モデル」を参照してください。
- プリプロセッサについては、「ユーザーガイダンス付きの CREATE MODEL」を参照してください。
- ハイパーパラメータについては、「AUTO OFF 付きの CREATE XGBoost モデル」を参照してください。
- 独自のモデル持参 (BYOM) については、「独自のモデルを持参 (BYOM) - ローカル推論」を参照してください。

Amazon Redshift 機械学習を使用するためのコスト

Amazon Redshift では、広範なデータエンジニアリングや機械学習の専門知識を必要とせずに、機械学習機能を活用してデータからインサイトを得ることができます。以下のセクションでは、Amazon Redshift ML の使用に関連するコストについて説明し、強力な機械学習統合を活用しながら、コストを計画し最適化する方法について説明します。

Amazon Redshift ML と SageMaker を使用するためのコスト

Amazon Redshift ML と SageMaker を併せて使用する場合、予測に既存のクラスターリソースを使用するため、Amazon Redshift で追加料金が発生することを回避できます。モデルの作成または使用に対しては、追加の Amazon Redshift の料金は発生しません。予測は Redshift クラスター内でローカルに実行されるため、クラスターのサイズを変更する必要がない限り追加料金は発生しません。Amazon Redshift ML はモデルのトレーニングに Amazon SageMaker を使用します。これには追加費用がかかります。

Amazon Redshift クラスター内で実行される予測関数については、追加料金は発生しません。CREATE MODEL ステートメントは Amazon SageMaker を使用し、追加料金が発生します。

コストは、トレーニングデータのセル数とともに増加します。セル数は、(トレーニングクエリまたはテーブルの時間) レコード数と列数の積です。たとえば、CREATE MODEL ステートメントの SELECT クエリが 10,000 レコードと 5 列を作成する場合、作成されるセルの数は 50,000 です。

場合によっては、CREATE MODEL の SELECT クエリによって生成されるトレーニングデータが、指定した MAX_CELLS 制限 (指定しなかった場合はデフォルトの 100 万) を超えことがあります。このような場合、CREATE MODEL はランダムに概算の MAX_CELLS (つまり、トレーニングデータセットからの「列数」レコード) を選択します。その後 CREATE MODEL は、このランダムに選択されたタプルを使用してトレーニングを実行します。ランダムなサンプリングの使用により、削減されたトレーニングデータセットにはバイアスが存在しないことが保証されます。したがって、MAX_CELLS を設定することで、トレーニングコストを制御できます。

CREATE MODEL ステートメントを使用する場合、MAX_CELLS オプションと MAX_RUNTIME オプションを使用して、コスト、時間、およびモデルに見込まれる精度を制御できます。

MAX_RUNTIME は、AUTO ON または OFF オプションを使用した場合に SageMaker でトレーニングにかかる最大時間を指定します。トレーニングジョブは、データセットのサイズに応じて、MAX_RUNTIME よりも早く完了することがよくあります。モデルのトレーニングが完了すると、Amazon Redshift はバックグラウンドで追加の作業を行い、クラスターにモデルをコンパイルしてインストールします。したがって、CREATE MODEL が完了するまで MAX_RUNTIME よりも時間がかかる場合があります。ただし、MAX_RUNTIME では、モデルをトレーニングするために SageMaker で使用される計算量と時間を制限します。モデルのステータスは、SHOW MODEL を使用していつでも確認できます。

AUTO ON で CREATE MODEL を実行すると、Amazon Redshift ML は SageMaker Autopilot を使用して、さまざまなモデル (または候補) を自動的かつインテリジェントに探索し、最適なモデルを見つけます。MAX_RUNTIME は、費やされる時間と計算の量を制限します。MAX_RUNTIME の設定が低すぎると、候補が 1 つでも探索するのに十分な時間がない可能性があります。「Autopilot の候補にモデルがありません」というエラーが表示された場合は、MAX_RUNTIME の値を大きくして CREATE MODEL を再実行してください。このパラメータの詳細については、Amazon SageMaker API リファレンスの [MaxAutoMLJobRuntimeInSeconds](#) を参照してください。

AUTO OFF で CREATE MODEL を実行すると、MAX_RUNTIME は SageMaker でトレーニングジョブが実行される時間の制限に対応します。多くの場合、トレーニングジョブは、データセットのサイズや、MODEL_TYPE XGBOOST の num_rounds など、使用されるその他のパラメータに応じてより早く完了します。

また、CREATE MODEL を実行するときに小さな MAX_CELLS 値を指定することで、コストを管理したり、トレーニング時間を短縮したりすることもできます。セルはデータベースのエン

トリです。各行は、列の数と同じ数のセルに対応します。列は、固定幅でも可変幅でもかまいません。MAX_CELLS はセルの数を制限するため、モデルのトレーニングに使用するトレーニング例の数を制限します。デフォルトでは、MAX_CELLS は 100 万個のセルに設定されています。MAX_CELLS を減らすと、Amazon Redshift がエクスポートしてモデルをトレーニングするために SageMaker に送信する CREATE MODEL の SELECT クエリの結果から、行数が減ります。したがって、MAX_CELLS を減らすと、AUTO ON と AUTO OFF の両方でモデルをトレーニングするために使用されるデータセットのサイズが小さくなります。このアプローチは、モデルのトレーニングにかかるコストと時間を削減する場合に役に立ちます。特定のトレーニングジョブのトレーニングと請求時間に関する情報を表示するには、Amazon SageMaker で [トレーニングジョブ] を選択します。

MAX_RUNTIME と MAX_CELLS を増やすと、SageMaker がより多くの候補を探索できるようになるため、モデルの品質が向上することがよくあります。この手法を取ると、SageMaker は各候補をトレーニングするためにより多くの時間を消費でき、また、より多くのデータを使用してより良いモデルのトレーニングが行えるようになります。データセットの反復または探索を高速化したい場合は、MAX_RUNTIME および MAX_CELLS を小さくしてください。モデルの精度を向上させたい場合は、より高い MAX_RUNTIME および MAX_CELLS を使用してください。

さまざまなセル番号に関連するコストと無料トライアルの詳細については、Amazon Redshift の料金を参照してください。

Amazon Bedrock で Amazon Redshift ML を使用するためのコスト

Amazon Bedrock で Amazon Redshift ML を使用すると、追加コストが発生します。詳細については、「Amazon Bedrock の料金体系」ページを参照してください。

Amazon Redshift 機械学習の開始方法

Amazon Redshift ML を使用すると、SQL ユーザーは、使い慣れた SQL コマンドを使用して、機械学習モデルを簡単に作成、トレーニング、デプロイできます。Amazon Redshift ML により Redshift クラスター内のデータを使用することで、Amazon SageMaker のモデルをトレーニングできます。その後、モデルはローカライズされ、Amazon Redshift データベース内で予測が行われます。Amazon Redshift ML は現在、機械学習アルゴリズム XGBoost (AUTO ON および AUTO OFF) と多層パーセプトロン (AUTO ON)、K-Means (AUTO OFF)、線形学習をサポートしています。

トピック

- Amazon Redshift ML 管理者によるクラスターと設定のセットアップ
- Amazon Redshift 機械学習でのモデルの説明可能性の使用

- Amazon Redshift ML 確率メトリクス

Amazon Redshift ML 管理者によるクラスターと設定のセットアップ

Amazon Redshift ML で作業する前に、クラスターの設定を完了し、Amazon Redshift ML を使用するためのアクセス許可を設定します。

Amazon Redshift ML を使用するためのクラスターの設定

Amazon Redshift ML を使用する前に、以下の前提条件を満たしてください。

Amazon Redshift でプロビジョニングされたクラスターを使用するには、Amazon Redshift の管理者は、次の 1 回限りのセットアップを行う必要があります。Amazon Redshift Serverless で Amazon Redshift ML を使用するには、「Amazon Redshift Serverless データウェアハウスの使用を開始」を参照してください。

Amazon Redshift ML のワンタイムクラスターセットアップを実行するには

1. AWS Management Console または AWS Command Line Interface (AWS CLI) を使用して、Redshift クラスターを作成します。クラスターの作成時には、必ずAWS Identity and Access Management (IAM) ポリシーをアタッチするようにしてください。Amazon SageMaker で Amazon Redshift ML を使用するために必要なアクセス許可の詳細については、「Amazon Redshift でのアイデンティティベースのポリシー (IAM ポリシー) の使用」を参照してください。
2. 以下のいずれかの方法で Amazon Redshift ML を使用するために必要な IAM ロールを作成します。

    - Amazon Redshift ML で SageMaker を使用するには、`AmazonS3FullAccess` ポリシーと `AmazonSageMakerFullAccess` ポリシーを持つ IAM ロールを作成します。Forecast モデルも作成する予定がある場合は、ロールにも `AmazonForecastFullAccess` ポリシーを添付してください。
    - Amazon Redshift ML で Amazon Bedrock を使用するには、`AmazonS3FullAccess` ポリシーと `AmazonBedrockFullAccess` ポリシーを持つ IAM ロールを作成します。
    - IAM ロールを作成する際は、CREATE MODEL など SQL コマンドの実行が許可された `AmazonRedshiftAllCommandsFullAccess` ポリシーを持つ、Amazon Redshift コンソールを使用することをお勧めします。Amazon Redshift は、シームレスな API ベースのメカニズムを使用して、AWS アカウント 内でユーザーに代わりプログラム的に IAM ロールを作成します。Amazon Redshift は、既存の AWS 管理ポリシーを IAM ロールに自動的にアタッチ

します。このアプローチにより、ロール作成のために IAM コンソールに切り替える必要はなくなり、作業を Amazon Redshift コンソール内で完了できます。詳細については、「[Creating an IAM role as default for Amazon Redshift]」を参照してください。

IAM ロールをクラスターのデフォルトとして作成する場合は、`redshift` をリソース名の一部として含めるか、RedShift 固有のタグを使用してそれらのリソースをタグ付けします。

クラスターで拡張 Amazon VPC ルーティングが有効になっている場合は、Amazon Redshift コンソールで作成された IAM ロールを使用できます。この IAM ロールには `AmazonRedshiftAllCommandsFullAccess` ポリシーがアタッチされており、このポリシーに以下の許可を追加します。これらの追加のアクセス許可により、Amazon Redshift は、お客様のアカウントで Elastic Network Interface (ENI) を作成および削除できます。さらにこの ENI を、Amazon EC2 または Amazon ECS で実行されているコンパイルタスクにアタッチします。これにより、Amazon S3 バケット内のオブジェクトに対しては、インターネットアクセスがブロックされた仮想プライベートクラウド (VPC) 内からのアクセスのみが許可されます。

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateNetworkInterface",
        "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "*"
}
```

Amazon Bedrock 基盤モデルを使用するには、次のセクションを追加します。

```
// Required section if you use Bedrock models.
{
   "Effect": "Allow",
```

```
    "Action": "bedrock:InvokeModel",
    "Resource": [
        "arn:aws:bedrock:<region>::foundation-model/*"
    ]
}
```

- より制限の厳しいポリシーを持つ IAM ロールを作成する場合、以下のポリシーを使用できます。必要に応じてこのポリシーを変更することもできます。

  Amazon S3 バケット `redshift-downloads/redshift-ml/` は、他のステップや例で使用されるサンプルデータが保存される場所です。Amazon S3 からデータをロードする必要がない場合は、削除できます。または、Amazon Redshift にデータをロードするために使用する他の Amazon S3 バケットに置き換えます。

  `your-account-id`、*`your-role`*、および `amzn-s3-demo-bucket` 値は、CREATE MODEL コマンドの一部として指定する値です。

  (オプション) Amazon Redshift ML の使用中に AWS KMS キーを指定する場合は、サンプルポリシーの AWS KMS キーセクションを使用します。*`your-kms-key`* 値は、CREATE MODEL コマンドの一部として使用するキーです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "sagemaker:*Job*",
                "sagemaker:AddTags",
                "sagemaker:CreateModel",
                "sagemaker:CreateEndpoint",
```

```
                "sagemaker:CreateEndpointConfig",
                "sagemaker:DeleteEndpoint",
                "sagemaker:DeleteEndpointConfig,
                "sagemaker:DeleteModel"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "s3:AbortMultipartUpload",
                "s3:GetObject",
                "s3:DeleteObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:iam::<your-account-id>:role/<your-role>",
                "arn:aws:s3:::amzn-s3-demo-bucket/*",
                "arn:aws:s3:::redshift-downloads/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket,
                "arn:aws:s3:::redshift-downloads"
            ]
        }
        // Optional section needed if you use AWS KMS keys.
        ,{
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:Encrypt",
                "kms:GenerateDataKey*"
             ],
             "Resource": [
```

```
                    "arn:aws:kms:<your-region>:<your-account-id>:key/<your-kms-key>"
                ]
            }
        ]
}
```

3. Amazon Redshift と SageMaker が、他のサービスとやり取りするロールを引き受けることを許可するには、IAM ロールに以下の信頼ポリシーを追加します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "redshift.amazonaws.com",
          "sagemaker.amazonaws.com",
          "forecast.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. (オプション) Amazon S3 バケットと AWS KMS キーを作成します。これらは、Amazon SageMaker に送信されたトレーニングデータを保存し、また、トレーニング済みモデルを Amazon SageMaker から受信するために、Amazon Redshift が使用するものです。
5. (オプション) さまざまなユーザーグループへのアクセスを制御するために、IAM ロールと Amazon S3 バケットのさまざまな組み合わせを作成します。
6. (オプション) Redshift クラスターで VPC ルーティングを有効にした場合は、この Redshift クラスターが置かれている VPC 用に、Amazon S3 VPC エンドポイントならびに SageMaker エンドポイントを作成します。これにより、CREATE MODEL の実行中、トラフィックがサービス間にある VPC を通過できるようになります。VPC ルーティングの詳細については、「Amazon Redshift の拡張 VPC ルーティング」を参照してください。

   ハイパーパラメータ調整ジョブにプライベート VPC を指定するために必要なアクセス許可の詳細については、Amazon SageMakerでのAmazon Redshift 機械学習を使用するために必要なアクセス許可を参照してください。

CREATE MODEL ステートメントを使用して、さまざまなユースケースモデルの作成を開始する方法については、「モデルを作成する」を参照してください。

許可と所有権の管理

Amazon Redshift はテーブルや関数などの他のデータベースオブジェクトと同様に、機械学習モデルの作成と使用をバインドし、制御メカニズムにアクセスします。予測関数を実行するモデルの作成に関しては、個別の許可があります。

次の例では、retention_analyst_grp(モデルクリエーター) とmarketing_analyst_grp(モデルユーザー)の 2 つのユーザーグループを使用して、Amazon Redshift がアクセス制御を管理する方法を示します。この retention analyst は、他のユーザーセットが取得した許可を通じて使用できる機械学習モデルを作成します。

スーパーユーザーは、次のステートメントを使用して、USER または GROUP に機械学習モデルを作成するための許可を GRANT (付与) できます。

```
GRANT CREATE MODEL TO GROUP retention_analyst_grp;
```

この許可を持つユーザーまたはグループは、クラスター内で任意のスキーマのモデルを作成できます (ユーザーが SCHEMA に対する通常の CREATE 許可を持っている場合)。機械学習モデルは、テーブル、ビュー、プロシージャ、およびユーザー定義関数と同様にスキーマ階層の一部です。

スキーマ demo_ml がすでに存在すると仮定して、次のように 2 つのユーザーグループにスキーマに対する許可を付与します。

```
GRANT CREATE, USAGE ON SCHEMA demo_ml TO GROUP retention_analyst_grp;
```

```
GRANT USAGE ON SCHEMA demo_ml TO GROUP marketing_analyst_grp;
```

他のユーザーが機械学習の推論機能を使用できるようにするには、EXECUTE 許可を付与します。次の例では、EXECUTE 許可を使用して、モデルを使用する許可を marketing_analst_grp GROUP に付与します。

```
GRANT EXECUTE ON MODEL demo_ml.customer_churn_auto_model TO GROUP
 marketing_analyst_grp;
```

ユーザーまたはグループからこれらの許可を取り消すには、CREATE MODEL および EXECUTE とともに REVOKE ステートメントを使用します。アクセス許可のコントロールコマンドの詳細については、「GRANT」および「REVOKE」を参照してください。

Amazon Redshift 機械学習でのモデルの説明可能性の使用

Amazon Redshift 機械学習 のモデルの説明可能性では、特徴量の重要度の値を使用して、トレーニングデータの各属性が予測結果にどのように寄与するかを理解することができます。

モデルの説明可能性は、モデルの予測を説明することで、機械学習 (ML) モデルの改善に役立ちます。モデルの説明可能性は、これらのモデルが特徴属性アプローチを使用してどのように予測するかを説明するのに役立ちます。

Amazon Redshift機械学習 には、モデルの説明機能が組み込まれており、モデルの説明機能が Amazon Redshift 機械学習 ユーザーに提供されます。モデルの説明可能性の詳細については、Machine Learning 予測における公平性とモデル説明可能性とはのAmazon SageMaker デベロッパーガイド。

また、モデルの説明可能性は、モデルが稼働時に行う特徴属性のドリフトに対する推論を監視します。また、リスクとコンプライアンスのチーム、および外部の規制当局への通知に使用できるモデルガバナンスレポートの作成に役立つツールも提供します。

CREATE MODEL ステートメントの使用時に AUTO ON または AUTO OFF オプションを指定すると、モデルトレーニングジョブが終了した後、SageMaker によって説明の出力が作成されます。EXPLAIN_MODEL 関数を使用すると、JSON形式で説明可能性に関するレポートのクエリを行えます。詳細については、「機械学習機能」を参照してください。

Amazon Redshift ML 確率メトリクス

教師あり学習問題では、クラスラベルは入力データを使用した予測の結果です。たとえば、モデルを使用して顧客がストリーミングサービスに再登録するかどうかを予測する場合、考えられるラベルは可能性が高いラベルと可能性の低いラベルです。Redshift ML には、各ラベルに確率を割り当ててその可能性を示す確率メトリクスの機能があります。これにより、予測された結果に基づいて、より多くの情報に基づいた意思決定を行うことができます。Amazon Redshift ML では、問題タイプが二項分類または多クラス分類の AUTO ON モデルを作成するときに、確率メトリクスを使用できます。AUTO ON パラメータを省略すると、Redshift ML はモデルが AUTO ON になっているはずだと仮定します。

モデルを作成する

モデルを作成すると、Amazon Redshift はモデルタイプと問題タイプを自動的に検出します。分類の問題の場合、Redshift は自動的に 2 つ目の推論関数を作成します。これを使用して、各ラベルに対する確率を出力できます。この 2 番目の推論関数の名前は、指定した推論関数名の後に文字列 `_probabilities` が続くものです。たとえば、推論関数に `customer_churn_predict` という名前を付けると、2 番目の推論関数の名前は `customer_churn_predict_probabilities` になります。次に、この関数にクエリを実行して、各ラベルの確率を取得できます。

```
CREATE MODEL customer_churn_model
FROM customer_activity
    PROBLEM_TYPE BINARY_CLASSIFICATION
TARGET churn
FUNCTION customer_churn_predict
IAM_ROLE {default}
AUTO ON
SETTINGS ( S3_BUCKET 'amzn-s3-demo-bucket'
```

確率を取得する

確率関数が準備できたら、コマンドを実行すると、返された確率の配列とそれに関連するラベルを含む SUPER 型が返されます。たとえば、結果 `"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]` は False ラベルの確率が 0.7、True ラベルの確率が 0.3 であることを意味します。

```
SELECT customer_churn_predict_probabilities(Account_length, Area_code,
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls)
FROM customer_activity;

customer_churn_predict_probabilities
 --------------------
 {"probabilities" : [0.7, 0.3], "labels" : ["False.", "True."]}
 {"probabilities" : [0.8, 0.2], "labels" : ["False.", "True."]}
 {"probabilities" : [0.75, 0.25], "labels" : ["True.", "False"]}
```

確率とラベルの配列は、常に確率の降順でソートされます。確率関数の SUPER で返された結果をネスト解除することで、最も高い確率で予測されたラベルのみを返すクエリを作成できます。

```
SELECT prediction.labels[0], prediction.probabilities[0]
```

```
            FROM (SELECT customer_churn_predict_probabilities(Account_length,
 Area_code,
            VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
            Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins, Intl_calls,
            Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);

  labels   | probabilities
-----------+--------------
 "False."  | 0.7
 "False."  | 0.8
 "True."   | 0.75
```

クエリを簡単にするために、予測関数の結果をテーブルに保存できます。

```
CREATE TABLE churn_auto_predict_probabilities AS
             (SELECT customer_churn_predict_probabilities(Account_length, Area_code,
             VMail_message, Day_mins, Day_calls, Day_charge,Eve_mins, Eve_calls,
             Eve_charge, Night_mins, Night_calls, Night_charge,Intl_mins,
             Intl_calls, Intl_charge, Cust_serv_calls) AS prediction
FROM customer_activity);
```

結果を含むテーブルをクエリすると、0.7 を超える確率を持つ予測のみを返すことができます。

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7;

  labels   | probabilities
-----------+--------------
 "False."  | 0.8
 "True."   | 0.75
```

インデックス表記を使うと、特定のラベルの確率を求めることができます。次の例では、すべての `True.` ラベルの確率を返します。

```
SELECT label, index, p.prediction.probabilities[index]
FROM churn_auto_predict_probabilities p, p.prediction.labels AS label AT index
WHERE label='True.';

  label  | index | probabilities
---------+-------+---------------
```

```
 "True." |      0 | 0.3
 "True." |      0 | 0.2
 "True." |      0 | 0.75
```

次の例では、顧客が解約する可能性が高いことを示す 0.7 を超える確率の `True` ラベルが付いたすべての行が返されます。

```
SELECT prediction.labels[0], prediction.probabilities[0]
FROM churn_auto_predict_probabilities
WHERE prediction.probabilities[0] > 0.7 AND prediction.labels[0] = "True.";

labels     | probabilities
-----------+--------------
 "True."   | 0.75
```

Amazon Redshift ML のチュートリアル

Amazon Redshift ML を使用すると、SQL ステートメントを使用して機械学習モデルをトレーニングし、予測のために SQL クエリでそれらのモデルを呼び出すことができます。Amazon Redshift の機械学習は、1 つの SQL コマンドでモデルをトレーニングします。Amazon Redshift は、Amazon SageMaker でトレーニングジョブを自動的に起動し、モデルを生成します。モデルが作成されると、そのモデルの予測関数を使用して Amazon Redshift で予測を実行できます。

これらのチュートリアルのステップに従って、Amazon Redshift ML の特徴について学習します。

- チュートリアル: カスタマーチャーンモデルの構築 - このチュートリアルでは、Amazon Redshift ML を使用して CREATE MODEL コマンドでカスタマーチャーンモデルを作成し、ユーザーシナリオの予測クエリを実行します。次に、CREATE MODEL コマンドが生成する SQL 関数を使用してクエリを実装します。
- チュートリアル: リモート推論モデルの構築 - 次のチュートリアルでは、Amazon Redshift を使用することなく、Amazon SageMaker で以前にトレーニングおよびデプロイしたランダムカットフォレストモデルを作成する方法の手順を示します。
- チュートリアル: K-means クラスタリングモデルの構築 - このチュートリアルでは、Amazon Redshift ML を使用して、K-means アルゴリズムに基づく機械学習モデルを作成、トレーニングおよびデプロイします。
- チュートリアル: 複数クラス分類モデルの構築 - このチュートリアルでは、Amazon Redshift ML を使用して、複数クラス分類の問題を解決する機械学習モデルを作成します。多クラス分類アルゴ

リズムは、データポイントを 3 つ以上のクラスのいずれかに分類します。次に、CREATE MODEL コマンドが生成する SQL 関数を使用してクエリを実装します。

- チュートリアル: XGBoost モデルの構築 - このチュートリアルでは、Amazon S3 のデータを使用してモデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。XGBoost アルゴリズムは、勾配ブーストツリーアルゴリズムの最適化された実装です。
- チュートリアル: リグレッションモデルの構築 - このチュートリアルでは、Amazon Redshift ML を使用して機械学習リグレッションモデルを作成し、そのモデルに対して予測クエリを実行します。リグレッションモデルを使用すると、住宅の価格や、都市の自転車レンタルサービスを利用する人数など、数値的な結果を予測できます。
- チュートリアル: 線形学習によるリグレッションモデルの構築 - このチュートリアルでは、Amazon S3 のデータを使用して線形学習モデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。SageMaker 線形学習アルゴリズムは、リグレッション問題または複数クラス分類問題のいずれかを解決します。
- チュートリアル: 線形学習による複数クラス分類モデルの構築 - このチュートリアルでは、Amazon S3 のデータを使用して線形学習モデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。SageMaker 線形学習アルゴリズムは、リグレッション問題または分類問題のいずれかを解決します。

チュートリアル: カスタマーチャーンモデルの構築

このチュートリアルでは、Amazon Redshift ML を使用して CREATE MODEL コマンドでカスタマーチャーンモデルを作成し、ユーザーシナリオの予測クエリを実行します。次に、CREATE MODEL コマンドが生成する SQL 関数を使用してクエリを実装します。

簡単な CREATE MODEL コマンドを使用して、トレーニングデータのエクスポート、モデルのトレーニング、モデルのインポート、Amazon Redshift 予測関数の準備を行うことができます。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL ステートメントを使用します。

この例では、履歴情報を使用して、携帯電話事業者のカスタマーチャーンに対して機械学習モデルを構築します。まず、SageMaker は機械学習モデルをトレーニングし、任意のカスタマーのプロファイル情報を使用してモデルをテストします。モデルが検証されると、Amazon SageMaker はモデルと予測関数を Amazon Redshift にデプロイします。予測関数を使用して、カスタマーが解約するかしないかを予測できます。

ユースケースの例

Amazon Redshift ML を使用して、セールスリードが成立するかどうかを予測するなど、他の二項分類の問題を解決できます。また、金融取引が不正であるかどうかを予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを使用して予測を実行する

前提条件

このチュートリアルを完了するためには、以下のものが必要です。

- Amazon Redshift ML 用に Amazon Redshift クラスターをセットアップする必要があります。これを行うには、「Amazon Redshift ML 管理者によるクラスターと設定のセットアップ」のドキュメントを使用します。
- モデルの作成に使用する Amazon Redshift クラスターと、トレーニングデータを使用し、モデルアーティファクトのステージングを保管する Amazon S3 バケットは、同じ AWS リージョンに置かれている必要があります。
- このドキュメントで使用されている SQL コマンドおよびサンプルデータセットをダウンロードするには、次のいずれかの操作を行います。
  - [SQL commands (SQL コマンド)]、[Customer activity file (顧客のアクティビティファイル)]、および [Abalone file (アワビファイル)] をダウンロードします。
  - Amazon S3 用の AWS CLI を使用して、次のコマンドを実行します。独自のターゲットパスを使用できます。

```
aws s3 cp s3://redshift-downloads/redshift-ml/tutorial-scripts/redshift-ml-tutorial.sql </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/customer_activity/customer_activity.csv </target/path>
aws s3 cp s3://redshift-downloads/redshift-ml/abalone_xgb/abalone_xgb.csv </target/path>
```

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタ v2 を使用してクエリを編集および実行し、結果を視覚化します。

次のクエリを実行すると、`customer_activity` という名前のテーブルを作成し、サンプルデータセットを Amazon S3 から取り込みます。

```
DROP TABLE IF EXISTS customer_activity;

CREATE TABLE customer_activity (
state varchar(2),
account_length int,
area_code int,
phone varchar(8),
intl_plan varchar(3),
vMail_plan varchar(3),
vMail_message int,
day_mins float,
day_calls int,
day_charge float,
total_charge float,
eve_mins float,
eve_calls int,
eve_charge float,
night_mins float,
night_calls int,
night_charge float,
intl_mins float,
intl_calls int,
intl_charge float,
cust_serv_calls int,
churn varchar(6),
record_date date
);

COPY customer_activity
FROM 's3://redshift-downloads/redshift-ml/customer_activity/'
REGION 'us-east-1' IAM_ROLE default
FORMAT AS CSV IGNOREHEADER 1;
```

ステップ 2: 機械学習モデルを作成する

チャーンはこのモデルのターゲット入力です。モデルの他のすべての入力は、チャーンを予測する関数の作成に役立つ属性です。

次の例では、CREATE MODEL オペレーションを使用して、カスタマーがアクティブになるかどうかを予測するモデルを提供します。これには、カスタマーの年齢、郵便番号、使用量、およびケースなどの入力が使用されます。次の例で、amzn-s3-demo-bucket は、ユーザーの Amazon S3 バケットに置き換えます。

```
CREATE MODEL customer_churn_auto_model
FROM
    (
      SELECT state,
             account_length,
             area_code,
             total_charge/account_length AS average_daily_spend,
             cust_serv_calls/account_length AS average_daily_cases,
             churn
      FROM customer_activity
      WHERE  record_date < '2020-01-01'
     )
TARGET churn FUNCTION ml_fn_customer_churn_auto
IAM_ROLE default SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket'
);
```

先ほどの例の SELECT クエリは、トレーニングデータを作成します。TARGET 句は、CREATE MODEL オペレーションが予測する方法を学習するために使用する機械学習ラベルである列を指定します。ターゲット列の「churn」は、顧客がまだ有効なメンバーシップを持っているか、メンバーシップを一時停止しているかを示します。S3_BUCKET フィールドは、以前に作成した Amazon S3 バケットの名前です。Amazon S3 バケットは、Amazon Redshift と Amazon SageMaker の間でトレーニングデータとアーティファクトを共有するために使用されます。残りの列は、予測に使用される機能です。

CREATE MODEL コマンド基本的なユースケースの構文と機能の概要については、[「単純な CREATE MODEL」]を参照してください。

サーバー側の暗号化のアクセス許可を追加する (オプション)

Amazon Redshift はデフォルトでトレーニングに Amazon SageMaker Autopilot を使用します。特に、Amazon Redshift は、カスタマー指定の Amazon S3 バケットにトレーニングデータを安全にエクスポートします。`KMS_KEY_ID` を指定しない場合、データはデフォルトでサーバー側の暗号化 SSE-S3 を使用して暗号化されます。

入力に対し、AWS KMS マネージドキーを使用したサーバー側の暗号化 (SSE-KMS) が行われている場合は、次のアクセス許可を追加します。

```
{
    "Effect": "Allow",
    "Action": [
    "kms:Encrypt"
    "kms:Decrypt"
    ]
}
```

Amazon SageMaker のロールの詳細については、Amazon SageMaker デベロッパーガイドの「Amazon SageMaker ロール」を参照してください。

モデルトレーニングのステータスを確認する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

テーブルのステータスを確認するには、次のオペレーションを使用します。

```
SHOW MODEL customer_churn_auto_model;
```

次は先ほどのオペレーションの出力例です。

```
+--------------------------
+------------------------------------------------------------------------------------------------------
+
|            Key            |
             Value
             |
+--------------------------
+------------------------------------------------------------------------------------------------------
+
```

```
|        Model Name        |
        customer_churn_auto_model
             |
|       Schema Name        |
                 public
             |
|          Owner           |
                 awsuser
             |
|      Creation Time       |
        Tue, 14.06.2022 17:15:52
             |
|       Model State        |
                TRAINING
             |
|                          |

             |
|      TRAINING DATA:      |

             |
|          Query           | SELECT STATE, ACCOUNT_LENGTH, AREA_CODE, TOTAL_CHARGE /
 ACCOUNT_LENGTH AS AVERAGE_DAILY_SPEND, CUST_SERV_CALLS / ACCOUNT_LENGTH AS
 AVERAGE_DAILY_CASES, CHURN |
|                          |
         FROM CUSTOMER_ACTIVITY
             |
|                          |
    WHERE RECORD_DATE < '2020-01-01'
             |
|      Target Column       |
                  CHURN
             |
|                          |

             |
|       PARAMETERS:        |

             |
|        Model Type        |
                  auto
             |
```

```
|       Problem Type       |

              |
|        Objective         |

              |
|     AutoML Job Name      |
     redshiftml-20220614171552640901
              |
|      Function Name       |
        ml_fn_customer_churn_auto
              |
|   Function Parameters    |                                         state
 account_length area_code average_daily_spend average_daily_cases
                         |
| Function Parameter Types |
      varchar int4 int4 float8 int4
              |
|         IAM Role         |
         default-aws-iam-role
              |
|        S3 Bucket         |
            amzn-s3-demo-bucket
              |
|       Max Runtime        |
                 5400
              |
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
```

モデルトレーニングが完了すると、`model_state` 変数は `Model is Ready` になり、予測関数が使用可能になります。

ステップ 3: モデルを使用して予測を実行する

SQL ステートメントを使用すると、予測モデルによって行われた予測を表示できます。この例では、CREATE MODEL オペレーションによって作成された予測関数の名前は `ml_fn_customer_churn_auto` です。予測関数の入力引数は、`state` には varchar、`account_length` には integer など、機能のタイプに対応しています。予測関数の出力は、CREATE MODEL ステートメントの TARGET 列と同じ型です。

1. 2020 年 1 月 1 日より前のデータでモデルをトレーニングしたので、テストセットで予測関数を使用します。例えば、次のクエリは、2020 年 1 月 1 日以降にサインアップしたカスタマーが、解約するかどうかを予測します。

```
SELECT
    phone,
    ml_fn_customer_churn_auto(
        state,
        account_length,
        area_code,
        total_charge / account_length,
        cust_serv_calls / account_length
    ) AS active
FROM
    customer_activity
WHERE
    record_date > '2020-01-01';
```

2. 次の例では、異なるユースケースで同じ予測関数を使用しています。このケースでは、Amazon Redshift は、記録の日付が 2020 年 1 月 1 日以降のさまざまな州からのカスタマーにおいて、解約者と非解約者の割合を予測します。

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
SELECT
    state,
    SUM(
        CASE
            WHEN active = 'True.' THEN 1
```

```
            ELSE 0
        END
    ) AS churners,
    SUM(
        CASE
            WHEN active = 'False.' THEN 1
            ELSE 0
        END
    ) AS nonchurners,
    COUNT(*) AS total_per_state
FROM
    predicted
GROUP BY
    state
ORDER BY
    state;
```

3. 次の例では、ある状態で解約するカスタマーの割合を予測するユースケースに予測関数を使用します。このケースでは、Amazon Redshift は、記録の日付が 2020 年 1 月 1 日より後の場合の解約率を予測します。

```
WITH predicted AS (
    SELECT
        state,
        ml_fn_customer_churn_auto(
            state,
            account_length,
            area_code,
            total_charge / account_length,
            cust_serv_calls / account_length
        ) :: varchar(6) AS active
    FROM
        customer_activity
    WHERE
        record_date > '2020-01-01'
)
SELECT
    state,
    CAST((CAST((SUM(
        CASE
            WHEN active = 'True.' THEN 1
            ELSE 0
        END
```

```
    )) AS FLOAT) / CAST(COUNT(*) AS FLOAT)) AS DECIMAL (3, 2)) AS pct_churn,
    COUNT(*) AS total_customers_per_state
FROM
    predicted
GROUP BY
    state
ORDER BY
    3 DESC;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift ML 機械学習を使用するためのコスト
- CREATE MODEL コマンド
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: リモート推論モデルの構築

次のチュートリアルでは、Amazon Redshift を使用することなく、Amazon SageMaker で以前にトレーニングおよびデプロイしたランダムカットフォレストモデルを作成する方法の手順を示します。ランダムカットフォレストアルゴリズムは、データセット内の異常なデータポイントを検出します。リモート推論を使用してモデルを作成すると、ランダムカットフォレスト SageMaker モデルを Amazon Redshift に取り込むことができます。次に、Amazon Redshift で、SQL を使用してリモート SageMaker エンドポイントで予測を実行します。

CREATE MODEL コマンドを使用して、Amazon SageMaker エンドポイントから機械学習モデルをインポートし、Amazon Redshift 予測関数を準備できます。CREATE MODEL オペレーションを使用するときは、SageMaker 機械学習モデルのエンドポイント名を指定します。

このチュートリアルでは、SageMaker モデルエンドポイントを使用して Amazon Redshift の機械学習モデルを作成します。機械学習モデルの準備が整ったら、それを使用して Amazon Redshift で予

測を実行できます。まず、Amazon SageMaker でエンドポイントをトレーニングして作成し、次にエンドポイント名を取得します。次に、CREATE MODEL コマンドを使用して、Amazon Redshift ML でモデルを作成します。最後に、CREATE MODEL コマンドによって生成される予測関数を使用して、モデルに予測を実行します。

ユースケースの例

ランダムカットフォレストモデルとリモート推論を使用して、電子商取引の急激な増減を予測するなど、他のデータセットの異常を検出できます。また、天候や地震活動の著しい変化を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon SageMaker モデルにデプロイする
- ステップ 2: SageMaker モデルエンドポイントを取得する
- ステップ 3: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 4: Amazon Redshift ML を使用してモデルを作成する
- ステップ 5: モデルを使用して予測を実行する

前提条件

このチュートリアルを完了するためには、以下のものが必要です。

- Amazon Redshift 機械学習の管理の設定を完了しました。
- NYC タクシーデータセットをダウンロードし、Amazon S3 バケットを作成し、Amazon S3 バケットにデータをアップロードしました。
- SageMaker モデルとエンドポイントをトレーニングしてデプロイし、SageMaker エンドポイントの名前を取得する必要があります。この AWS CloudFormation テンプレートを使用して、すべての SageMaker リソースを AWS アカウントに自動的にプロビジョニングします。

ステップ 1: Amazon SageMaker モデルにデプロイする

1. モデルをデプロイするには、Amazon SageMaker コンソールに移動し、ナビゲーションペインで [Notebook] (ノートブック)から [Notebook instances] (ノートブックインスタンス) を選択します。
2. CloudFormation テンプレートによって作成された Jupyter Notebook に対して [Open Jupyter] (Jupyter を開く) を選択します。

3. [bring-your-own-model-remote-inference.ipynb] を選択します。
4. 次の行を Amazon S3 バケットとプレフィックスに置き換えて、トレーニングの入力と出力を Amazon S3 に保存するパラメータを設定します。

```
data_location=f"s3://{bucket}/{prefix}/",
output_path=f"s3://{bucket}/{prefix}/output",
```

5. [fast-forward] (早送り) ボタンを押すと、すべてのセルで実行されます。

ステップ 2: SageMaker モデルエンドポイントを取得する

Amazon SageMaker コンソールで、ナビゲーションペインの [Inference] (推論) で、[Endpoints] (エンドポイント) を選択し、モデル名を見つけます。Amazon Redshift でリモート推論モデルを作成するときは、モデルのエンドポイント名をコピーする必要があります。

ステップ 3: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタ v2 を使用して Amazon Redshift で以下の SQL コマンドを実行します。これらのコマンドは `rcf_taxi_data` テーブルが存在する場合はそれをドロップし、同じ名前のテーブルを作成して、サンプルデータセットをテーブルにロードします。

```
DROP TABLE IF EXISTS public.rcf_taxi_data CASCADE;

CREATE TABLE public.rcf_taxi_data (ride_timestamp timestamp, nbr_passengers int);

COPY public.rcf_taxi_data
FROM
    's3://sagemaker-sample-files/datasets/tabular/anomaly_benchmark_taxi/
NAB_nyc_taxi.csv'
    IAM_ROLE default
    IGNOREHEADER 1
    FORMAT AS CSV;
```

ステップ 4: Amazon Redshift ML を使用してモデルを作成する

次のクエリを実行し、前のステップで取得した SageMaker モデルエンドポイントを使用して Amazon Redshift ML でモデルを作成します。*randomcutforest-xxxxxxxxx* を独自の SageMaker エンドポイントの名前と置き換えます。

```
CREATE MODEL public.remote_random_cut_forest
```

```
FUNCTION remote_fn_rcf(int)
RETURNS decimal(10, 6) SAGEMAKER '<randomcutforest-xxxxxxxxx>' IAM_ROLE default;
```

モデルのステータスを確認する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

モデルのステータスを確認するには、次の SHOW MODEL オペレーションを使用します。

```
SHOW MODEL public.remote_random_cut_forest
```

出力には、SageMaker エンドポイントと関数名が表示されます。

```
+--------------------------+-----------------------------------------+
|        Model Name        |        remote_random_cut_forest         |
+--------------------------+-----------------------------------------+
|       Schema Name        |                 public                  |
|          Owner           |                 awsuser                 |
|      Creation Time       |        Wed, 15.06.2022 17:58:21         |
|       Model State        |                  READY                  |
|                          |                                         |
|       PARAMETERS:        |                                         |
|         Endpoint         |       <randomcutforest-xxxxxxxxx>       |
|      Function Name       |              remote_fn_rcf              |
|      Inference Type      |                 Remote                  |
| Function Parameter Types |                  int4                   |
|         IAM Role         |          default-aws-iam-role           |
+--------------------------+-----------------------------------------+
```

ステップ 5: モデルを使用して予測を実行する

Amazon SageMaker ランダムカットフォレストアルゴリズムは、データセット内の異常なデータポイントを検出するように設計されています。この例では、モデルは、重要なイベントによるタクシー乗車の急増を検出するように設計されています。このモデルを使用して、各データポイントの異常スコアを生成することにより、異常なイベントを予測できます。

タクシーデータセット全体の異常スコアを計算するには、次のクエリを使用します。ここでは、前のステップの CREATE MODEL ステートメントで使用した関数を参照しています。

```
SELECT
    ride_timestamp,
    nbr_passengers,
```

```
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data;
```

異常値の上限と下限を確認する (オプション)

次のクエリを実行して、平均スコアから 3 標準偏差を超えるスコアのデータポイントを見つけます。

```
WITH score_cutoff AS (
    SELECT
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean + 3 * std) AS score_cutoff_value
    FROM
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score > (
        SELECT
            score_cutoff_value
        FROM
            score_cutoff
    )
ORDER BY
    2 DESC;
```

次のクエリを実行して、平均スコアから 3 標準偏差を超えるスコアのデータポイントを見つけます。

```
WITH score_cutoff AS (
    SELECT
        STDDEV(public.remote_fn_rcf(nbr_passengers)) AS std,
        AVG(public.remote_fn_rcf(nbr_passengers)) AS mean,
        (mean - 3 * std) AS score_cutoff_value
    FROM
```

```
        public.rcf_taxi_data
)
SELECT
    ride_timestamp,
    nbr_passengers,
    public.remote_fn_rcf(nbr_passengers) AS score
FROM
    public.rcf_taxi_data
WHERE
    score < (
        SELECT
            score_cutoff_value
        FROM
            score_cutoff
    )
ORDER BY
    2 DESC;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: K-means クラスタリングモデルの構築

このチュートリアルでは、Amazon Redshift ML を使用して、k-means アルゴリズムに基づく機械学習モデルを作成、トレーニングおよびデプロイします。このアルゴリズムは、データ内でグループを検出する際にクラスタリングで発生する問題を解決します。K-means は、まだラベル付けされていないデータのグループ化に役立ちます。K-means クラスタリングの詳細については、「Amazon SageMaker デベロッパーガイドの「k-means クラスタリングの仕組み」を参照してください。

CREATE MODEL オペレーションを使用して、Amazon Redshift クラスターから K-means モデルを作成します。CREATE MODEL コマンドを使用して、トレーニングデータのエクスポート、モデルのトレーニング、モデルのインポート、Amazon Redshift 予測関数の準備を行うことができます。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL オペレーションを使用します。

このチュートリアルでは、K-means を[イベント、言語、トーンのグローバルデータベース (GDELT)](#)データセットで使用し、世界中のニュースを監視して、データを日々秒単位で保存します K-means は、似たようなトーン、俳優、または場所のイベントをグループ化します。データは、Amazon Simple Storage Service の 2 つの異なるフォルダに複数のファイルで保存されます。フォルダは、1979 年から 2013 年、毎日のアップデート、2013 年以降が履歴的になっています。この例では、履歴フォーマットを使用して、1979 年のデータを取り込みます。

ユースケースの例

Amazon Redshift ML では、ストリーミングサービスで同じような視聴習慣を持つカスタマーをグループ化するなど、クラスタリングに関するその他の問題を解決できます。また、Redshift ML を使用して、配送サービスに最適な配送センターの数を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを使用して予測を実行する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

1. Amazon Redshift クエリエディタv2 を使用して次のクエリを実行します。クエリはパブリックスキーマに `gdelt_data` テーブルが存在する場合はそれをドロップし、パブリックスキーマに同じ名前のテーブルを作成します。

```
DROP TABLE IF EXISTS gdelt_data CASCADE;
```

```
CREATE TABLE gdelt_data (
    GlobalEventId bigint,
    SqlDate bigint,
    MonthYear bigint,
    Year bigint,
    FractionDate double precision,
    Actor1Code varchar(256),
    Actor1Name varchar(256),
    Actor1CountryCode varchar(256),
    Actor1KnownGroupCode varchar(256),
    Actor1EthnicCode varchar(256),
    Actor1Religion1Code varchar(256),
    Actor1Religion2Code varchar(256),
    Actor1Type1Code varchar(256),
    Actor1Type2Code varchar(256),
    Actor1Type3Code varchar(256),
    Actor2Code varchar(256),
    Actor2Name varchar(256),
    Actor2CountryCode varchar(256),
    Actor2KnownGroupCode varchar(256),
    Actor2EthnicCode varchar(256),
    Actor2Religion1Code varchar(256),
    Actor2Religion2Code varchar(256),
    Actor2Type1Code varchar(256),
    Actor2Type2Code varchar(256),
    Actor2Type3Code varchar(256),
    IsRootEvent bigint,
    EventCode bigint,
    EventBaseCode bigint,
    EventRootCode bigint,
    QuadClass bigint,
    GoldsteinScale double precision,
    NumMentions bigint,
    NumSources bigint,
    NumArticles bigint,
    AvgTone double precision,
    Actor1Geo_Type bigint,
    Actor1Geo_FullName varchar(256),
    Actor1Geo_CountryCode varchar(256),
    Actor1Geo_ADM1Code varchar(256),
    Actor1Geo_Lat double precision,
    Actor1Geo_Long double precision,
    Actor1Geo_FeatureID bigint,
    Actor2Geo_Type bigint,
```

```
    Actor2Geo_FullName varchar(256),
    Actor2Geo_CountryCode varchar(256),
    Actor2Geo_ADM1Code varchar(256),
    Actor2Geo_Lat double precision,
    Actor2Geo_Long double precision,
    Actor2Geo_FeatureID bigint,
    ActionGeo_Type bigint,
    ActionGeo_FullName varchar(256),
    ActionGeo_CountryCode varchar(256),
    ActionGeo_ADM1Code varchar(256),
    ActionGeo_Lat double precision,
    ActionGeo_Long double precision,
    ActionGeo_FeatureID bigint,
    DATEADDED bigint
);
```

2. 次のクエリは、サンプルデータを gdelt_data テーブルにロードします。

```
COPY gdelt_data
FROM 's3://gdelt-open-data/events/1979.csv'
REGION 'us-east-1'
IAM_ROLE default
CSV
DELIMITER '\t';
```

トレーニングデータを検証する (オプション)

モデルがどのデータでトレーニングされるかを確認するには、次のクエリを使用します。

```
SELECT
    AvgTone,
    EventCode,
    NumArticles,
    Actor1Geo_Lat,
    Actor1Geo_Long,
    Actor2Geo_Lat,
    Actor2Geo_Long
FROM
    gdelt_data LIMIT 100;
```

ステップ 2: 機械学習モデルを作成する

次の例では、CREATE MODEL コマンドを使用して、データを 7 つのクラスターにグループ化するモデルを作成します。K 値は、データポイントが分割されるクラスターの数です。このモデルは、データポイントをそれらが互いに類似しているクラスターに分類します。データポイントをグループにクラスタリングすることにより、K-Means アルゴリズムは最適なクラスター中心を反復的に決定します。次に、アルゴリズムは各データポイントを最も近いクラスター中心に割り当てます。最も近い中心が同じである各メンバーは、同じグループに属します。1 つのグループには、可能な限り互いに類似した (かつ、他のグループのメンバーとは可能な限り相違する) メンバーが集められます。K 値は主観的であり、データポイント間の類似性を測定する方法に依存します。クラスターが不均等に分布している場合は、K 値を変更してクラスターサイズを平滑化できます。

次の例で、amzn-s3-demo-bucket は、ユーザーの Amazon S3 バケットに置き換えます。

```
CREATE MODEL news_data_clusters
FROM
    (
        SELECT
            AvgTone,
            EventCode,
            NumArticles,
            Actor1Geo_Lat,
            Actor1Geo_Long,
            Actor2Geo_Lat,
            Actor2Geo_Long
        FROM
            gdelt_data
    ) FUNCTION news_monitoring_cluster
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE KMEANS
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
    EXCEPT
    (K '7')
    SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

モデルトレーニングのステータスを確認する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

モデルのステータスを確認するには、次の SHOW MODEL オペレーションを使用して、`Model State` が`Ready` であるかを調べます。

```
SHOW MODEL NEWS_DATA_CLUSTERS;
```

モデルの準備が整うと、前のオペレーションの出力は `Model State` が `Ready` であることが表示されます。次は SHOW MODEL オペレーションの出力例です。

```
+--------------------------
+------------------------------------------------------------------------------------------------------
+
|        Model Name        |
 news_data_clusters                                            |
+--------------------------
+------------------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                      public
                                |
|          Owner           |                                                     awsuser
                                |
|      Creation Time       |                                             Fri, 17.06.2022
 16:32:19                                    |
|       Model State        |                                                      READY
                                |
|        train:msd         |                                                  2973.822754
                                |
|      train:progress      |                                                   100.000000
                                |
|     train:throughput     |                                                 237114.875000
                                |
|      Estimated Cost      |                                                    0.004983
                                |
|                          |
                                |
|      TRAINING DATA:      |
                                |
|          Query           | SELECT AVGTONE, EVENTCODE, NUMARTICLES, ACTOR1GEO_LAT,
 ACTOR1GEO_LONG, ACTOR2GEO_LAT, ACTOR2GEO_LONG |
|                          |                                                FROM GDELT_DATA
                                |
|                          |
                                |
```

```
|       PARAMETERS:        |
                                            |
|        Model Type        |                                                kmeans
                                            |
|    Training Job Name     |
 redshiftml-20220617163219978978-kmeans                                |
|      Function Name       |
 news_monitoring_cluster                                        |
|   Function Parameters    |       avgtone eventcode numarticles actor1geo_lat
 actor1geo_long actor2geo_lat actor2geo_long        |
| Function Parameter Types |                           float8 int8 int8 float8 float8
 float8 float8                              |
|         IAM Role         |                                      default-aws-iam-
role                                          |
|        S3 Bucket         |                                         amzn-s3-demo-
bucket                                        |
|       Max Runtime        |                                                 5400
                                            |
|                          |
                                            |
|     HYPERPARAMETERS:     |
                                            |
|       feature_dim        |                                                  7
                                            |
|            k             |                                                  7
                                            |
+--------------------------
+------------------------------------------------------------------------------------
+
```

ステップ 3: モデルを使用して予測を実行する

クラスターを特定する

モデルによってデータ内で識別される離散グループ (別名クラスター) を見つけることができます。クラスターとは、他のどのクラスター中心よりもそのクラスター中心に近いデータポイントの集合です。K 値はモデル内のクラスター数を表すため、クラスター中心の数も表します。次のクエリは、それぞれの `globaleventid` に関連付けられたクラスターを表示することによってクラスターを識別します。

```
SELECT
    globaleventid,
```

```
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS cluster
FROM
    gdelt_data;
```

データの分布を確認する

クラスター間のデータの分布をチェックして、選択した K 値によってデータがある程度均等に分布されたかどうかを確認できます。次のクエリを使用して、データがクラスター全体に均等に分布されているかどうかを判断します。

```
SELECT
    events_cluster,
    COUNT(*) AS nbr_events
FROM
    (
        SELECT
            globaleventid,
            news_monitoring_cluster(
                AvgTone,
                EventCode,
                NumArticles,
                Actor1Geo_Lat,
                Actor1Geo_Long,
                Actor2Geo_Lat,
                Actor2Geo_Long
            ) AS events_cluster
        FROM
            gdelt_data
    )
GROUP BY
    1;
```

クラスターが不均等に分布している場合は、K 値を変更してクラスターサイズを平滑化できます。

クラスター中心を決定する

データポイントは、他のどのクラスターセンターよりもクラスター中心の近くにあります。したがって、クラスター中心を調べることは、クラスターを定義するのに役立ちます。

次のクエリを実行して、イベントコード別の記事数に基づいてクラスター中心を決定します。

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    SUM(numArticles) AS numArticles
FROM
    gdelt_data
GROUP BY
    1,
    2;
```

クラスターのデータポイントに関する情報を表示する

次のクエリを使用して、5 番目のクラスターに割り当てられたポイントのデータを返します。選択した記事には 2 人の俳優が含まれている必要があります。

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    eventcode,
    actor1name,
```

```
    actor2name,
    SUM(numarticles) AS totalarticles
FROM
    gdelt_data
WHERE
    events_cluster = 5
    AND actor1name <> ' '
    AND actor2name <> ' '
GROUP BY
    1,
    2,
    3,
    4
ORDER BY
    5 desc;
```

同じ人種コードの俳優のイベントに関するデータを表示する

次のクエリは、肯定的なトーンのイベントについて書かれた記事数を係数します。また、このクエリでは、2 人の俳優の民族コードが同じである必要があり、各イベントが割り当てられているクラスターが返されます。

```
SELECT
    news_monitoring_cluster (
        AvgTone,
        EventCode,
        NumArticles,
        Actor1Geo_Lat,
        Actor1Geo_Long,
        Actor2Geo_Lat,
        Actor2Geo_Long
    ) AS events_cluster,
    SUM(numarticles) AS total_articles,
    eventcode AS event_code,
    Actor1EthnicCode AS ethnic_code
FROM
    gdelt_data
WHERE
    Actor1EthnicCode = Actor2EthnicCode
    AND Actor1EthnicCode <> ' '
    AND Actor2EthnicCode <> ' '
    AND AvgTone > 0
GROUP BY
```

```
    1,
    3,
    4
HAVING
    (total_articles) > 4
ORDER BY
    1,
    2 ASC;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: 複数クラス分類モデルの構築

このチュートリアルでは、Amazon Redshift ML を使用して、複数クラス分類の問題を解決する機械学習モデルを作成します。多クラス分類アルゴリズムは、データポイントを 3 つ以上のクラスのいずれかに分類します。次に、CREATE MODEL コマンドが生成する SQL 関数を使用してクエリを実装します。

CREATE MODEL コマンドを使用して、トレーニングデータのエクスポート、モデルのトレーニング、モデルのインポート、Amazon Redshift 予測関数の準備を行うことができます。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL オペレーションを使用します。

このチュートリアルでは、パブリックデータセットの E-Commerce Sales Forecast を使用します。これには、英国のオンライン小売業者の販売データが含まれます。生成するモデルは、特別なカスタマーロイヤルティプログラムで最もアクティブなカスタマーをターゲットにします。多クラス分類で

は、このモデルを使用して、カスタマーが 13 か月間でアクティブになる月数を予測できます。予測機能は、プログラムへの加入期間が 7 か月以上続くと予測されるカスタマーを指定します。

ユースケースの例

Amazon Redshift ML では、製品ラインからベストセラー製品を予測するなど、その他の多クラス分類の問題を解決できます。また、リンゴ、ナシ、オレンジの選択など、画像に含まれる果物を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを使用して予測を実行する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタv2 を使用する次のクエリを実行します。これらのクエリは、サンプルデータを Amazon Redshift にロードします。

1. 次のクエリでは、`ecommerce_sales` という名前のテーブルを作成します。

```
CREATE TABLE IF NOT EXISTS ecommerce_sales (
    invoiceno VARCHAR(30),
    stockcode VARCHAR(30),
    description VARCHAR(60),
    quantity DOUBLE PRECISION,
    invoicedate VARCHAR(30),
    unitprice DOUBLE PRECISION,
    customerid BIGINT,
    country VARCHAR(25)
);
```

2. 次のクエリは、サンプルデータを E-Commerce Sales Forecast データセットから ecommerce_sales テーブルにコピーします。

```
COPY ecommerce_sales
FROM
    's3://redshift-ml-multiclass/ecommerce_data.txt'
IAM_ROLE default
DELIMITER '\t'
IGNOREHEADER 1
REGION 'us-east-1'
MAXERROR 100;
```

データを分割する

Amazon Redshift ML でモデルを作成すると、SageMaker はデータをトレーニングセットとテストセットに自動的に分割し、SageMaker がモデルの精度を判断できるようにします。このステップでデータを手動で分割することにより、追加の予測セットを割り当てることでモデルの精度を検証できます。

次の SQL ステートメントを使用して、データをトレーニング、検証、および予測の 3 つのセットに分割します。

```
--creates table with all data
CREATE TABLE ecommerce_sales_data AS (
    SELECT
        t1.stockcode,
        t1.description,
        t1.invoicedate,
        t1.customerid,
        t1.country,
        t1.sales_amt,
        CAST(RANDOM() * 100 AS INT) AS data_group_id
    FROM
        (
            SELECT
                stockcode,
                description,
                invoicedate,
                customerid,
                country,
                SUM(quantity * unitprice) AS sales_amt
```

```
            FROM
                ecommerce_sales
            GROUP BY
                1,
                2,
                3,
                4,
                5
        ) t1
);

--creates training set
CREATE TABLE ecommerce_sales_training AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id < 80
);
```

```
--creates validation set
CREATE TABLE ecommerce_sales_validation AS (
    SELECT
        a.customerid,
        a.country,
        a.stockcode,
        a.description,
        a.invoicedate,
        a.sales_amt,
        (b.nbr_months_active) AS nbr_months_active
    FROM
        ecommerce_sales_data a
        INNER JOIN (
            SELECT
                customerid,
                COUNT(
                    DISTINCT(
                        DATE_PART(y, CAST(invoicedate AS DATE)) || '-' || LPAD(
                            DATE_PART(mon, CAST(invoicedate AS DATE)),
                            2,
                            '00'
                        )
                    )
                ) AS nbr_months_active
            FROM
                ecommerce_sales_data
            GROUP BY
                1
        ) b ON a.customerid = b.customerid
    WHERE
        a.data_group_id BETWEEN 80
        AND 90
);

--creates prediction set
CREATE TABLE ecommerce_sales_prediction AS (
    SELECT
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    FROM
```

```
        ecommerce_sales_data
    WHERE
        data_group_id > 90);
```

ステップ 2: 機械学習モデルを作成する

このステップでは、CREATE MODEL ステートメントを使用して、多クラス分類を使用して機械学習モデルを作成します。

次のクエリは、CREATE MODEL オペレーションを使用して、トレーニングセットを含む多クラス分類モデルを作成します。amzn-s3-demo-bucket は、ユーザーの Amazon S3 バケットに置き換えます。

```
CREATE MODEL ecommerce_customer_activity
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
            nbr_months_active
        FROM
            ecommerce_sales_training
    ) TARGET nbr_months_active FUNCTION predict_customer_activity IAM_ROLE default
 PROBLEM_TYPE MULTICLASS_CLASSIFICATION SETTINGS (
        S3_BUCKET 'amzn-s3-demo-bucket',
        S3_GARBAGE_COLLECT OFF
    );
```

このクエリでは、問題タイプを `Multiclass_Classification` に指定します。モデルについて予測するターゲットは `nbr_months_active` です。SageMaker がモデルのトレーニングを終了すると、`predict_customer_activity` 関数が作成されます。これは、Amazon Redshift で予測を行うために使用します。

モデルトレーニングのステータスを表示する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

次のクエリを使用して、モデルの状態や精度など、モデルのさまざまなメトリクスを返します。

```
SHOW MODEL ecommerce_customer_activity;
```

モデルの準備が整うと、前のオペレーションの出力は `Model State` が `Ready` であることが表示されます。次は SHOW MODEL オペレーションの出力例です。

```
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
|        Model Name        |
 ecommerce_customer_activity                                  |
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
|       Schema Name        |                                                  public
                                    |
|          Owner           |                                                  awsuser
                                    |
|      Creation Time       |                                  Fri, 17.06.2022 19:02:15
                                    |
|       Model State        |                                                   READY
                                    |
|   Training Job Status    |
 MaxAutoMLJobRuntimeReached                                   |
|   validation:accuracy    |                                                 0.991280
                                    |
|      Estimated Cost      |                                                 7.897689
                                    |
|                          |
                                    |
|      TRAINING DATA:      |
                                    |
|          Query           | SELECT CUSTOMERID, COUNTRY, STOCKCODE, DESCRIPTION,
 INVOICEDATE, SALES_AMT, NBR_MONTHS_ACTIVE |
|                          |                                 FROM
 ECOMMERCE_SALES_TRAINING                                 |
|      Target Column       |                                            NBR_MONTHS_ACTIVE
                                    |
|                          |
                                    |
|       PARAMETERS:        |
                                    |
|        Model Type        |                                                  xgboost
                                    |
```

```
|       Problem Type       |                                  MulticlassClassification
                                  |
|        Objective         |                                          Accuracy
                                  |
|     AutoML Job Name      |
 redshiftml-20220617190215268770                            |
|      Function Name       |
 predict_customer_activity                               |
|   Function Parameters    |                customerid country stockcode description
 invoicedate sales_amt                |
| Function Parameter Types |                          int8 varchar varchar varchar
 varchar float8                         |
|         IAM Role         |                                   default-aws-iam-role
                                  |
|        S3 Bucket         |                                       amzn-s3-demo-
bucket                                  |
|       Max Runtime        |                                            5400
                                  |
+--------------------------
+---------------------------------------------------------------------------------------
+
```

ステップ 3: モデルを使用して予測を実行する

次のクエリは、どのカスタマーがカスタマーロイヤルティプログラムの対象となるかを示します。モデルは、カスタマーが少なくとも 7 か月間アクティブになると予測する場合、ロイヤルティプログラムにそのカスタマーを選択します。

```
SELECT
    customerid,
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active
FROM
    ecommerce_sales_prediction
WHERE
    predicted_months_active >= 7
GROUP BY
```

```
    1,
    2
LIMIT
    10;
```

検証データに対して予測クエリを実行する (オプション)

検証データに対して次の予測クエリを実行して、モデルの精度レベルを確認します。

```
SELECT
    CAST(SUM(t1.match) AS decimal(7, 2)) AS predicted_matches,
    CAST(SUM(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(SUM(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
        SELECT
            customerid,
            country,
            stockcode,
            description,
            invoicedate,
            sales_amt,
            nbr_months_active,
            predict_customer_activity(
                customerid,
                country,
                stockcode,
                description,
                invoicedate,
                sales_amt
            ) AS predicted_months_active,
            CASE
                WHEN nbr_months_active = predicted_months_active THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN nbr_months_active <> predicted_months_active THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            ecommerce_sales_validation
    )t1;
```

エントリを逃した顧客の数を予測する (オプション)

次のクエリは、5 か月または 6 か月間のみアクティブであると予測されるカスタマーの数を比較します。このモデルは、これらのカスタマーがロイヤルティプログラムに選ばれないと予測しています。次に、クエリは、プログラムをわずかな差で逃す数と、ロイヤルティプログラムの対象となると予測される数を比較します。このクエリは、ロイヤルティプログラムのしきい値を下げるかどうかの決定を通知するために使用できます。また、プログラムをわずかな差で逃すカスタマーの数が多数存在するかどうかについても判断できます。これにより、これらのカスタマーに対して、ロイヤルティプログラムのメンバーシップを取得するための活動を増やすように促すことができます。

```
SELECT
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
        sales_amt
    ) AS predicted_months_active,
    COUNT(customerid)
FROM
    ecommerce_sales_prediction
WHERE
    predicted_months_active BETWEEN 5 AND 6
GROUP BY
    1
ORDER BY
    1 ASC
LIMIT
    10)
UNION
(SELECT
      NULL AS predicted_months_active,
    COUNT (customerid)
FROM
    ecommerce_sales_prediction
WHERE
    predict_customer_activity(
        customerid,
        country,
        stockcode,
        description,
        invoicedate,
```

```
        sales_amt
    ) >=7);
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: XGBoost モデルの構築

このチュートリアルでは、Amazon S3 のデータを使用してモデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。XGBoost アルゴリズムは、勾配ブーストツリーアルゴリズムの最適化された実装です。XGBoost は、他の勾配ブーストツリーアルゴリズムよりも多くのデータ型、リレーションシップ、および分布を処理します。XGBoost は、リグレッション、分類 (二項分類および多クラス分類)、ランキングの問題に使用できます。XGBoost アルゴリズムの詳細については「Amazon SageMaker デベロッパーガイド」の「XGBoost アルゴリズム」を参照してください。

`AUTO OFF` オプションの Amazon Redshift ML `CREATE MODEL` オペレーションは、現在 XGBoost を `MODEL_TYPE` としてサポートしています。目的やハイパーパラメータなどの関連情報をユースケースに基づいて、`CREATE MODEL` コマンドの一部として提供できます。

このチュートリアルでは、紙幣認証データセット を使用します。これは、特定の紙幣が本物か偽造かを予測する二項分類問題です。

ユースケースの例

Amazon Redshift ML を使用して、患者が健康か、または病気があるかを予測するなど、他の二項分類問題を解決できます。また、メールがスパムかスパムでないかを予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを使用して予測を実行する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタv2 を使用する次のクエリを実行します。

次のクエリは、2 つのテーブルを作成し、Amazon S3 からデータをロードし、データをトレーニングセットとテストセットに分割します。トレーニングセットを使用してモデルをトレーニングし、予測関数を作成します。次に、テストセットで予測関数をテストします。

```
--create training set table
CREATE TABLE banknoteauthentication_train(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
    entropy FLOAT,
    class INT
);

--Load into training table
COPY banknoteauthentication_train
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/train_data/' IAM_ROLE
 default REGION 'us-west-2' IGNOREHEADER 1 CSV;

--create testing set table
CREATE TABLE banknoteauthentication_test(
    variance FLOAT,
    skewness FLOAT,
    curtosis FLOAT,
```

```
    entropy FLOAT,
    class INT
);

--Load data into testing table
COPY banknoteauthentication_test
FROM
    's3://redshiftbucket-ml-sagemaker/banknote_authentication/test_data/'
    IAM_ROLE default
    REGION 'us-west-2'
    IGNOREHEADER 1
    CSV;
```

ステップ 2: 機械学習モデルを作成する

次のクエリは、前のステップで作成したトレーニングセットから Amazon Redshift ML で XGBoost モデルを作成します。amzn-s3-demo-bucket ユーザーの S3_BUCKET と置換します。これにより、入力データセットと他の Redshift ML アーティファクトが格納されます。

```
CREATE MODEL model_banknoteauthentication_xgboost_binary
FROM
    banknoteauthentication_train
    TARGET class
    FUNCTION func_model_banknoteauthentication_xgboost_binary
    IAM_ROLE default
    AUTO OFF
    MODEL_TYPE xgboost
    OBJECTIVE 'binary:logistic'
    PREPROCESSORS 'none'
    HYPERPARAMETERS DEFAULT
EXCEPT(NUM_ROUND '100')
SETTINGS(S3_BUCKET 'amzn-s3-demo-bucket');
```

モデルトレーニングのステータスを表示する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

次のクエリを使用して、モデルトレーニングの進行状況を監視します。

```
SHOW MODEL model_banknoteauthentication_xgboost_binary;
```

モデルが READY の場合、SHOW MODEL オペレーションは、次の出力例に示すように train:error メトリクスを提供します。train:error メトリクスは、小数点以下 6 桁までのモデルの精度の測定単位です。値 0 が最も正確で、値 1 が最も低い精度です。

```
+--------------------------+--------------------------------------------------+
|        Model Name        |   model_banknoteauthentication_xgboost_binary    |
+--------------------------+--------------------------------------------------+
| Schema Name              | public                                           |
| Owner                    | awsuser                                          |
| Creation Time            | Tue, 21.06.2022 19:07:35                         |
| Model State              | READY                                            |
| train:error              |                                         0.000000 |
| Estimated Cost           |                                         0.006197 |
|                          |                                                  |
| TRAINING DATA:           |                                                  |
| Query                    | SELECT *                                         |
|                          | FROM "BANKNOTEAUTHENTICATION_TRAIN"              |
| Target Column            | CLASS                                            |
|                          |                                                  |
| PARAMETERS:              |                                                  |
| Model Type               | xgboost                                          |
| Training Job Name        | redshiftml-20220621190735686935-xgboost          |
| Function Name            | func_model_banknoteauthentication_xgboost_binary |
| Function Parameters      | variance skewness curtosis entropy               |
| Function Parameter Types | float8 float8 float8 float8                      |
| IAM Role                 | default-aws-iam-role                             |
| S3 Bucket                | amzn-s3-demo-bucket                              |
| Max Runtime              |                                             5400 |
|                          |                                                  |
| HYPERPARAMETERS:         |                                                  |
| num_round                |                                              100 |
| objective                | binary:logistic                                  |
+--------------------------+--------------------------------------------------+
```

ステップ 3: モデルを使用して予測を実行する

モデルの精度を確認する

次の予測クエリでは、前のステップで作成した予測関数を使用してモデルの精度を確認します。このクエリをテストセットで実行して、モデルがトレーニングセットに対して過度な近さで対応しないこ

とを確認します。この近い対応はオーバーフィットとも呼ばれ、オーバーフィットによりモデルが信頼できない予測を行う可能性があります。

```
WITH predict_data AS (
    SELECT
        class AS label,
        func_model_banknoteauthentication_xgboost_binary (variance, skewness, curtosis,
 entropy) AS predicted,
        CASE
            WHEN label IS NULL THEN 0
            ELSE label
        END AS actual,
        CASE
            WHEN actual = predicted THEN 1 :: INT
            ELSE 0 :: INT
        END AS correct
    FROM
        banknoteauthentication_test
),
aggr_data AS (
    SELECT
        SUM(correct) AS num_correct,
        COUNT(*) AS total
    FROM
        predict_data
)
SELECT
    (num_correct :: FLOAT / total :: FLOAT) AS accuracy
FROM
    aggr_data;
```

本物の紙幣と偽造紙幣の量を予測する

次の予測クエリは、テストセットに含まれる本物の紙幣と偽造紙幣の予測量を返します。

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted
    FROM
        banknoteauthentication_test
)
SELECT
```

```
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    COUNT(1) AS count
FROM
    predict_data
GROUP BY
    1;
```

本物の紙幣と偽造紙幣の平均観測値を求める

次の予測クエリは、テストセットで本物および偽造であると予測される紙幣の各特長の平均値を返します。

```
WITH predict_data AS (
    SELECT
        func_model_banknoteauthentication_xgboost_binary(variance, skewness, curtosis,
 entropy) AS predicted,
          variance,
          skewness,
          curtosis,
          entropy
    FROM
        banknoteauthentication_test
)
SELECT
    CASE
        WHEN predicted = '0' THEN 'Original banknote'
        WHEN predicted = '1' THEN 'Counterfeit banknote'
        ELSE 'NA'
    END AS banknote_authentication,
    TRUNC(AVG(variance), 2) AS avg_variance,
    TRUNC(AVG(skewness), 2) AS avg_skewness,
    TRUNC(AVG(curtosis), 2) AS avg_curtosis,
    TRUNC(AVG(entropy), 2) AS avg_entropy
FROM
    predict_data
GROUP BY
    1
ORDER BY
    2;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: リグレッションモデルの構築

このチュートリアルでは、Amazon Redshift ML を使用して機械学習リグレッションモデルを作成し、そのモデルに対して予測クエリを実行します。リグレッションモデルを使用すると、住宅の価格や、都市の自転車レンタルサービスを利用する人数など、数値的な結果を予測できます。Amazon Redshift で CREATE MODEL コマンドをトレーニングデータと共に使用します。次に、Amazon Redshift ML はモデルをコンパイルし、トレーニング済みのモデルを Redshift にインポートして、SQL 予測関数を準備します。予測関数は、Amazon Redshift の SQL クエリで使用できます。

このチュートリアルでは、Amazon Redshift ML を使用して、1 日の任意の時間にトロント市の自転車シェアサービスを利用する人の数を予測するリグレッションモデルを構築します。モデルの入力には、祝日と気象条件が含まれます。この問題には数値的な結果が必要なため、リグレッションモデルを使用します。

CREATE MODEL コマンドを使用して、トレーニングデータをエクスポートし、モデルをトレーニングし、Amazon Redshift で SQL 関数として使用できるようにします。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL オペレーションを使用します。

ユースケースの例

Amazon Redshift ML では、カスタマーの生涯価値を予測するなど、その他のリグレッションの問題を解決できます。また、Redshift ML を使用して、最も収益性の高い価格と製品の売上を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを検証する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタv2 を使用する次のクエリを実行します。

1. 3 つのパブリックデータセットを Amazon Redshift に読み込むには、3 つのテーブルを作成する必要があります。データセットはトロントバイクライダーシップデータ、履歴気象データ、および履歴的な休日データを使用します。Amazon Redshift クエリエディタで次のクエリを実行し、`ridership`、`weather`、および `holiday` という名前のテーブルを作成します。

```
CREATE TABLE IF NOT EXISTS ridership (
    trip_id INT,
    trip_duration_seconds INT,
    trip_start_time timestamp,
    trip_stop_time timestamp,
    from_station_name VARCHAR(50),
    to_station_name VARCHAR(50),
    from_station_id SMALLINT,
    to_station_id SMALLINT,
    user_type VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS weather (
    longitude_x DECIMAL(5, 2),
    latitude_y DECIMAL(5, 2),
    station_name VARCHAR(20),
    climate_id BIGINT,
    datetime_utc TIMESTAMP,
    weather_year SMALLINT,
    weather_month SMALLINT,
```

```
    weather_day SMALLINT,
    time_utc VARCHAR(5),
    temp_c DECIMAL(5, 2),
    temp_flag VARCHAR(1),
    dew_point_temp_c DECIMAL(5, 2),
    dew_point_temp_flag VARCHAR(1),
    rel_hum SMALLINT,
    rel_hum_flag VARCHAR(1),
    precip_amount_mm DECIMAL(5, 2),
    precip_amount_flag VARCHAR(1),
    wind_dir_10s_deg VARCHAR(10),
    wind_dir_flag VARCHAR(1),
    wind_spd_kmh VARCHAR(10),
    wind_spd_flag VARCHAR(1),
    visibility_km VARCHAR(10),
    visibility_flag VARCHAR(1),
    stn_press_kpa DECIMAL(5, 2),
    stn_press_flag VARCHAR(1),
    hmdx SMALLINT,
    hmdx_flag VARCHAR(1),
    wind_chill VARCHAR(10),
    wind_chill_flag VARCHAR(1),
    weather VARCHAR(10)
);

CREATE TABLE IF NOT EXISTS holiday (holiday_date DATE, description VARCHAR(100));
```

2. 次のクエリは、前のステップで作成したテーブルにサンプルデータをロードします。

```
COPY ridership
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/ridership/'
    IAM_ROLE default
    FORMAT CSV
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY weather
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/weather/'
    IAM_ROLE default
```

```
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;

COPY holiday
FROM
    's3://redshift-ml-bikesharing-data/bike-sharing-data/holiday/'
    IAM_ROLE default
    FORMAT csv
    IGNOREHEADER 1
    DATEFORMAT 'auto'
    TIMEFORMAT 'auto'
    REGION 'us-west-2'
    gzip;
```

3. 次のクエリは、ridership および weather データセットで変換を実行して、バイアスや異常を除去します。バイアスおよび異常を除去すると、モデルの精度が向上します。このクエリは、ridership_view および weather_view という名前の 2 つの新しいビューを作成することによってテーブルを単純化します。

```
CREATE
OR REPLACE VIEW ridership_view AS
SELECT
    trip_time,
    trip_count,
    TO_CHAR(trip_time, 'hh24') :: INT trip_hour,
    TO_CHAR(trip_time, 'dd') :: INT trip_day,
    TO_CHAR(trip_time, 'mm') :: INT trip_month,
    TO_CHAR(trip_time, 'yy') :: INT trip_year,
    TO_CHAR(trip_time, 'q') :: INT trip_quarter,
    TO_CHAR(trip_time, 'w') :: INT trip_month_week,
    TO_CHAR(trip_time, 'd') :: INT trip_week_day
FROM
    (
        SELECT
            CASE
                WHEN TRUNC(r.trip_start_time) < '2017-07-01' :: DATE THEN
 CONVERT_TIMEZONE(
                    'US/Eastern',
                    DATE_TRUNC('hour', r.trip_start_time)
```

```
                  )
                  ELSE DATE_TRUNC('hour', r.trip_start_time)
              END trip_time,
              COUNT(1) trip_count
          FROM
              ridership r
          WHERE
              r.trip_duration_seconds BETWEEN 60
              AND 60 * 60 * 24
          GROUP BY
              1
      );

CREATE
OR REPLACE VIEW weather_view AS
SELECT
    CONVERT_TIMEZONE(
        'US/Eastern',
        DATE_TRUNC('hour', datetime_utc)
    ) daytime,
    ROUND(AVG(temp_c)) temp_c,
    ROUND(AVG(precip_amount_mm)) precip_amount_mm
FROM
    weather
GROUP BY
    1;
```

4. 次のクエリは、ridership_view および weather_view からの関連するすべての入力属性を trip_data テーブル組み合わせたテーブルを作成します。

```
CREATE TABLE trip_data AS
SELECT
    r.trip_time,
    r.trip_count,
    r.trip_hour,
    r.trip_day,
    r.trip_month,
    r.trip_year,
    r.trip_quarter,
    r.trip_month_week,
    r.trip_week_day,
    w.temp_c,
    w.precip_amount_mm,CASE
```

```
        WHEN h.holiday_date IS NOT NULL THEN 1
        WHEN TO_CHAR(r.trip_time, 'D') :: INT IN (1, 7) THEN 1
        ELSE 0
    END is_holiday,
    ROW_NUMBER() OVER (
        ORDER BY
            RANDOM()
    ) serial_number
FROM
    ridership_view r
    JOIN weather_view w ON (r.trip_time = w.daytime)
    LEFT OUTER JOIN holiday h ON (TRUNC(r.trip_time) = h.holiday_date);
```

サンプルデータを表示する (オプション)

次のクエリはテーブルのエントリを示しています。このオペレーションを実行すると、テーブルが正しく作成されたことを確認できます。

```
SELECT *
FROM trip_data
LIMIT 5;
```

次は先ほどのオペレーションの出力例です。

```
+---------------------+------------+-----------+----------+------------+-----------
+--------------+-----------------+---------------+--------+------------------
+------------+---------------+
|      trip_time      | trip_count | trip_hour | trip_day | trip_month | trip_year
 | trip_quarter | trip_month_week | trip_week_day | temp_c | precip_amount_mm |
 is_holiday | serial_number |
+---------------------+------------+-----------+----------+------------+-----------
+--------------+-----------------+---------------+--------+------------------
+------------+---------------+
| 2017-03-21 22:00:00 |         47 |        22 |       21 |          3 |        17 |
         1 |               3 |             3 |      1 |                0 |          0 |
            1 |
| 2018-05-04 01:00:00 |         19 |         1 |        4 |          5 |        18 |
         2 |               1 |             6 |     12 |                0 |          0 |
            3 |
```

```
| 2018-01-11 10:00:00 |         93 |        10 |       11 |          1 |         18 |
        1 |               2 |             5 |      9 |                0 |          0 |
            5 |
| 2017-10-28 04:00:00 |         20 |         4 |       28 |         10 |         17 |
        4 |               4 |             7 |     11 |                0 |          1 |
            7 |
| 2017-12-31 21:00:00 |         11 |        21 |       31 |         12 |         17 |
        4 |               5 |             1 |    -15 |                0 |          1 |
            9 |
+---------------------+-----------+----------+---------+-----------+-----------
+-------------+-----------------+--------------+--------+-----------------
+------------+--------------+
```

属性間の相関関係を表示する (オプション)

相関関係の決定は、属性間の関連性の強度を測定するのに役立ちます。関連性のレベルは、ターゲット出力に何が影響するかを判断するのに役立ちます。このチュートリアルでは、ターゲット出力は trip_count です。

次のクエリは、sp_correlation プロシージャを作成するか、または置き換えます。sp_correlation というストアドプロシージャを使用すると、Amazon Redshift のテーブルのある属性と他の属性の相関関係を示すことができます。

```
CREATE OR REPLACE PROCEDURE sp_correlation(source_schema_name in varchar(255),
 source_table_name in varchar(255), target_column_name in varchar(255),
 output_temp_table_name inout varchar(255)) AS $$
DECLARE
  v_sql varchar(max);
  v_generated_sql varchar(max);
  v_source_schema_name varchar(255)=lower(source_schema_name);
  v_source_table_name varchar(255)=lower(source_table_name);
  v_target_column_name varchar(255)=lower(target_column_name);
BEGIN
  EXECUTE 'DROP TABLE IF EXISTS ' || output_temp_table_name;
  v_sql = '
SELECT
  ''CREATE temp table '|| output_temp_table_name||' AS SELECT ''|| outer_calculation||
  '' FROM (SELECT COUNT(1) number_of_items, SUM('||v_target_column_name||')
 sum_target, SUM(POW('||v_target_column_name||',2)) sum_square_target, POW(SUM('||
v_target_column_name||'),2) square_sum_target,''||
  inner_calculation||
  '' FROM (SELECT ''||
  column_name||
```

```
  '' FROM '||v_source_table_name||'))''
FROM
  (
  SELECT
    DISTINCT
    LISTAGG(outer_calculation,'','') OVER () outer_calculation
    ,LISTAGG(inner_calculation,'','') OVER () inner_calculation
    ,LISTAGG(column_name,'','') OVER () column_name
  FROM
    (
    SELECT
      CASE WHEN atttypid=16 THEN ''DECODE(''||column_name||'',true,1,0)'' ELSE
 column_name END column_name
      ,atttypid
      ,''CAST(DECODE(number_of_items * sum_square_''||rn||'' - square_sum_''||
rn||'',0,null,(number_of_items*sum_target_''||rn||'' - sum_target * sum_''||rn||
        '')/SQRT((number_of_items * sum_square_target - square_sum_target) *
 (number_of_items * sum_square_''||rn||
        '' - square_sum_''||rn||''))) AS numeric(5,2)) ''||column_name
 outer_calculation
      ,''sum(''||column_name||'') sum_''||rn||'',''||
            ''SUM(trip_count*''||column_name||'') sum_target_''||rn||'',''||
            ''SUM(POW(''||column_name||'',2)) sum_square_''||rn||'',''||
            ''POW(SUM(''||column_name||''),2) square_sum_''||rn inner_calculation
    FROM
      (
      SELECT
        row_number() OVER (order by a.attnum) rn
        ,a.attname::VARCHAR column_name
        ,a.atttypid
      FROM pg_namespace AS n
        INNER JOIN pg_class AS c ON n.oid = c.relnamespace
        INNER JOIN pg_attribute AS a ON c.oid = a.attrelid
      WHERE a.attnum > 0
        AND n.nspname = '''||v_source_schema_name||'''
        AND c.relname = '''||v_source_table_name||'''
        AND a.atttypid IN (16,20,21,23,700,701,1700)
      )
    )
)';
  EXECUTE v_sql INTO v_generated_sql;
  EXECUTE  v_generated_sql;
END;
```

```
$$ LANGUAGE plpgsql;
```

次のクエリはターゲットカラム、trip_count、およびデータセット内のその他の数値属性の相関関係を示します。

```
call sp_correlation(
    'public',
    'trip_data',
    'trip_count',
    'tmp_corr_table'
);

SELECT
    *
FROM
    tmp_corr_table;
```

次は先の sp_correlation オペレーションの出力例です。

```
+------------+-----------+----------+------------+-----------+-------------
+-----------------+---------------+--------+-----------------+-----------
+---------------+
| trip_count | trip_hour | trip_day | trip_month | trip_year | trip_quarter
 | trip_month_week | trip_week_day | temp_c | precip_amount_mm | is_holiday |
 serial_number |
+------------+-----------+----------+------------+-----------+-------------
+-----------------+---------------+--------+-----------------+-----------
+---------------+
|          1 |      0.32 |     0.01 |       0.18 |      0.12 |          0.18 |
     0 |           0.02 |   0.53 |            -0.07 |      -0.13 |             0 |
+------------+-----------+----------+------------+-----------+-------------
+-----------------+---------------+--------+-----------------+-----------
+---------------+
```

ステップ 2: 機械学習モデルを作成する

1. 次のクエリは、データセットの 80% をトレーニング用に、20% を検証用に指定して、データをトレーニングセットと検証セットに分割します。トレーニングセットとは、モデルに最適なアルゴリズムを特定するための ML モデルの入力です。モデルを作成したら、検証セットを使用してモデルの精度を検証します。

```
CREATE TABLE training_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
    temp_c,
    precip_amount_mm,
    is_holiday
FROM
    trip_data
WHERE
    serial_number > (
        SELECT
            COUNT(1) * 0.2
        FROM
            trip_data
    );

CREATE TABLE validation_data AS
SELECT
    trip_count,
    trip_hour,
    trip_day,
    trip_month,
    trip_year,
    trip_quarter,
    trip_month_week,
    trip_week_day,
    temp_c,
    precip_amount_mm,
    is_holiday,
    trip_time
FROM
    trip_data
WHERE
    serial_number <= (
        SELECT
            COUNT(1) * 0.2
```

```
        FROM
            trip_data
    );
```

2. 次のクエリは、リグレッションモデルを作成し、任意の入力日時の `trip_count` 値を予測します。次の例で、amzn-s3-demo-bucket は、ユーザーの S3 バケットに置き換えます。

```
CREATE MODEL predict_rental_count
FROM
    training_data TARGET trip_count FUNCTION predict_rental_count
    IAM_ROLE default
    PROBLEM_TYPE regression
    OBJECTIVE 'mse'
    SETTINGS (
        s3_bucket 'amzn-s3-demo-bucket',
        s3_garbage_collect off,
        max_runtime 5000
    );
```

ステップ 3: モデルを検証する

1. 次のクエリを使用してモデルの側面を出力し、出力で平均二乗誤差メトリクスを見つけます。平均二乗誤差は、リグレッション問題の典型的な精度指標です。

```
show model predict_rental_count;
```

2. 検証データに対して次の予測クエリを実行し、予測されるトリップ数と実際のトリップ数を比較します。

```
SELECT
    trip_time,
    actual_count,
    predicted_count,
    (actual_count - predicted_count) difference
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
```

```
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    )
LIMIT
    5;
```

3. 次のクエリは、検証データに基づいて平均二乗誤差と二乗平均平方根誤差を計算します。平均二乗誤差と二乗平均平方根誤差を使用して、予測された数値ターゲットと実際の数値解の間の距離を測定します。優れたモデルは、両方の指標で低いスコアになります。両方のメトリクスは、次のクエリの値を返します。

```
SELECT
    ROUND(
        AVG(POWER((actual_count - predicted_count), 2)),
        2
    ) mse,
    ROUND(
        SQRT(AVG(POWER((actual_count - predicted_count), 2))),
        2
    ) rmse
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
```

```
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    );
```

4. 次のクエリは、2017 年 1 月 1 日の各トリップ時間に対するトリップ数の誤差率を計算します。このクエリは、誤差率が最も低い時間から誤差率が最も高い時間にトリップ時間を並べ替えます。

```
SELECT
    trip_time,
    CAST(ABS(((actual_count - predicted_count) / actual_count)) * 100 AS DECIMAL
 (7,2)) AS pct_error
FROM
    (
        SELECT
            trip_time,
            trip_count AS actual_count,
            PREDICT_RENTAL_COUNT (
                trip_hour,
                trip_day,
                trip_month,
                trip_year,
                trip_quarter,
                trip_month_week,
                trip_week_day,
                temp_c,
                precip_amount_mm,
                is_holiday
            ) predicted_count
        FROM
            validation_data
    )
WHERE
   trip_time LIKE '2017-01-01 %%:%%:%%'
ORDER BY
   2 ASC;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: 線形学習によるリグレッションモデルの構築

このチュートリアルでは、Amazon S3 のデータを使用して線形学習モデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。SageMaker 線形学習アルゴリズムは、リグレッション問題または複数クラス分類問題のいずれかを解決します。リグレッション問題と多クラス分類問題の詳細については、「Amazon SageMaker デベロッパーガイド」の「機械学習パラダイムの問題タイプ」を参照してください。このチュートリアルでは、リグレッション問題を解決します。線形学習アルゴリズムは、多数のモデルを並列でトレーニングし、最も最適化されたモデルを自動的に決定します。Amazon Redshift で CREATE MODEL オペレーションを使用し、SageMaker で線形学習モデルを作成して、予測関数を Amazon Redshift に送信します。線形学習アルゴリズムの詳細については、「Amazon SageMaker デベロッパーガイド」の「線形学習アルゴリズム」を参照してください。

CREATE MODEL コマンドを使用して、トレーニングデータのエクスポート、モデルのトレーニング、モデルのインポート、Amazon Redshift 予測関数の準備を行うことができます。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL オペレーションを使用します。

線形学習モデルは、連続的な目標または離散的な目標のいずれかを最適化します。連続的な目標はリグレッションに使用され、離散変数は分類に使用されます。リグレッション法など、一部の方法は、連続的な目標のみに対する解を提供します。線形学習アルゴリズムは、Naive Bayes 手法などの単純なハイパーパラメータ最適化手法より高速です。単純な最適化手法では、各入力変数は独立していると仮定します。線形学習アルゴリズムを使用するには、入力の次元を表す列を指定し、観測値を表す

行を指定する必要があります。線形学習アルゴリズムの詳細については「Amazon SageMaker デベロッパーガイド」の「線形学習アルゴリズム」を参照してください。

このチュートリアルでは、アワビの年齢を予測する線形学習モデルを作成します。アワビのデータセットに対してCREATE MODEL コマンドを使用し、アワビの物理的測定値の関係を決定します。次に、モデルを使用してアワビの年齢を判断します。

ユースケースの例

家の価格を予測するなど、線形学習と Amazon Redshift ML を使用して他のリグレッション問題を解決することもできます。また、Redshift ML を使用して、都市の自転車レンタルサービスを利用する人数を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを検証する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタv2 を使用する次のクエリを実行します。これらのクエリは、Redshift にサンプルデータをロードし、データをトレーニングセットと検証セットに分割します。

1. 次のクエリは、`abalone_dataset` テーブルを作成します。

```
CREATE TABLE abalone_dataset (
    id INT IDENTITY(1, 1),
    Sex CHAR(1),
    Length float,
    Diameter float,
    Height float,
    Whole float,
```

```
    Shucked float,
    Viscera float,
    Shell float,
    Rings integer
);
```

2. 次のクエリは、サンプルデータを Amazon S3 の [アワビのデータセット](#)から Amazon Redshift で以前に作成した abalone_dataset テーブルにコピーします。

```
COPY abalone_dataset
FROM
    's3://redshift-ml-multiclass/abalone.csv' REGION 'us-east-1' IAM_ROLE default CSV
 IGNOREHEADER 1 NULL AS 'NULL';
```

3. データを手動で分割することにより、追加の予測セットを割り当てることでモデルの精度を検証できます。次のクエリは、データを 2 つのセットに分割します。abalone_training テーブルはトレーニング用で、abalone_validation テーブルは検証用です。

```
CREATE TABLE abalone_training as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) < 8;

CREATE TABLE abalone_validation as
SELECT
    *
FROM
    abalone_dataset
WHERE
    mod(id, 10) >= 8;
```

ステップ 2: 機械学習モデルを作成する

このステップでは、CREATE MODEL ステートメントを使用して、線形学習アルゴリズムでの機械学習モデルを作成します。

次のクエリは、S3 バケットを使用して CREATE MODEL オペレーションで線形学習モデルを作成します。amzn-s3-demo-bucket をユーザーの S3 バケットに置き換えます。

```
CREATE MODEL model_abalone_ring_prediction
FROM
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS target_label
        FROM
            abalone_training
    ) TARGET target_label FUNCTION f_abalone_ring_prediction IAM_ROLE default
 MODEL_TYPE LINEAR_LEARNER PROBLEM_TYPE REGRESSION OBJECTIVE 'MSE' SETTINGS (
        S3_BUCKET 'amzn-s3-demo-bucket',
        MAX_RUNTIME 15000
    );
```

モデルトレーニングのステータスを表示する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

次のクエリを使用して、モデルトレーニングの進行状況を監視します。

```
SHOW MODEL model_abalone_ring_prediction;
```

モデルの準備が整うと、先ほどのオペレーションの出力は、次の例のようになります。出力には、平均二乗誤差である validation:mse メトリクスを使用します。平均二乗誤差を使用して、次のステップでモデルの精度を検証します。

```
+--------------------------
+------------------------------------------------------------------------------------------
+
|         Model Name         |
 model_abalone_ring_prediction                                    |
+--------------------------
+------------------------------------------------------------------------------------------
+
```

```
| Schema Name              | public
                                          |
| Owner                    | awsuser
                                          |
| Creation Time            | Thu, 30.06.2022 18:00:10
                                          |
| Model State              | READY
                                          |
| validation:mse           |
                                4.168633 |
| Estimated Cost           |
                                4.291608 |
|                          |
                                          |
| TRAINING DATA:           |
                                          |
| Query                    | SELECT SEX , LENGTH , DIAMETER , HEIGHT , WHOLE ,
 SHUCKED , VISCERA , SHELL, RINGS AS TARGET_LABEL |
|                          | FROM ABALONE_TRAINING
                                          |
| Target Column            | TARGET_LABEL
                                          |
|                          |
                                          |
| PARAMETERS:              |
                                          |
| Model Type               | linear_learner
                                          |
| Problem Type             | Regression
                                          |
| Objective                | MSE
                                          |
| AutoML Job Name          | redshiftml-20220630180010947843
                                          |
| Function Name            | f_abalone_ring_prediction
                                          |
| Function Parameters      | sex length diameter height whole shucked viscera shell
                                          |
| Function Parameter Types | bpchar float8 float8 float8 float8 float8 float8 float8
                                          |
| IAM Role                 | default-aws-iam-role
                                          |
| S3 Bucket                | amzn-s3-demo-bucket
                                          |
```

```
| Max Runtime                |
                                   15000 |
+--------------------------
+-----------------------------------------------------------------------------------------------------
+
```

ステップ 3: モデルを検証する

1. 次の予測クエリは、平均二乗誤差と二乗平均平方根誤差を計算して abalone_validation データセットでモデルの精度を検証します。

```
SELECT
    ROUND(AVG(POWER((tgt_label - predicted), 2)), 2) mse,
    ROUND(SQRT(AVG(POWER((tgt_label - predicted), 2))), 2) rmse
FROM
    (
        SELECT
            Sex,
            Length,
            Diameter,
            Height,
            Whole,
            Shucked,
            Viscera,
            Shell,
            Rings AS tgt_label,
            f_abalone_ring_prediction(
                Sex,
                Length,
                Diameter,
                Height,
                Whole,
                Shucked,
                Viscera,
                Shell
            ) AS predicted,
            CASE
                WHEN tgt_label = predicted then 1
                ELSE 0
            END AS match,
            CASE
                WHEN tgt_label <> predicted then 1
                ELSE 0
```

```
            END AS nonmatch
        FROM
            abalone_validation
    ) t1;
```

前のクエリの出力は次の例のようになります。平均二乗誤差メトリクスの値は、SHOW MODEL オペレーションの出力によって示される validation:mse メトリクスに似ています。

```
+-----+--------------------+
| mse |        rmse        |
+-----+--------------------+
| 5.1 | 2.2600000000000002 |
+-----+--------------------+
```

2. 次のクエリを使用して、予測関数で EXPLAIN_MODEL オペレーションを実行します。このオペレーションにより、モデルの説明可能性レポートが返されます。EXPLAIN_MODEL オペレーションの詳細については「Amazon Redshift データベースデベロッパーガイド」の「EXPLAIN_MODEL関数」を参照してください。

```
SELECT
    EXPLAIN_MODEL ('model_abalone_ring_prediction');
```

次の情報は、前の EXPLAIN_MODEL オペレーションによって生成されたモデルの説明可能性レポートの例です。各入力の値は Shapley 値です。Shapley 値は、各入力がモデルの予測に与える影響を表し、値の高い入力ほど予測に大きな影響を与えます。この例では、値の高い入力は、アワビの年齢の予測により大きな影響を与えます。

```
{
    "explanations": {
        "kernel_shap": {
            "label0": {
                "expected_value" :10.290688514709473,
                "global_shap_values": {
                    "diameter" :0.6856910187882492,
                    "height" :0.4415323937124035,
                    "length" :0.21507476107609084,
                    "sex" :0.448611774505744,
                    "shell" :1.70426496893776,
                    "shucked" :2.1181392924386994,
                    "viscera" :0.342220754059912,
                    "whole" :0.6711906974084011
```

```
                }
            }
        }
    },
    "version" :"1.0"
};
```

3. 次のクエリを使用して、まだ成熟していないアワビについて、モデルが行う正しい予測のパーセンテージを計算します。未熟なアワビは輪が 10 個以下で、正確な予測では、実際の輪の数が 1 輪以内で高精度となります。

```
SELECT
    TRUNC(
        SUM(
            CASE
                WHEN ROUND(
                    f_abalone_ring_prediction(
                        Sex,
                        Length,
                        Diameter,
                        Height,
                        Whole,
                        Shucked,
                        Viscera,
                        Shell
                    ),
                    0
                ) BETWEEN Rings - 1
                AND Rings + 1 THEN 1
                ELSE 0
            END
        ) / CAST(COUNT(SHELL) AS FLOAT),
        4
    ) AS prediction_pct
FROM
    abalone_validation
WHERE
    Rings <= 10;
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

チュートリアル: 線形学習による複数クラス分類モデルの構築

このチュートリアルでは、Amazon S3 のデータを使用して線形学習モデルを作成し、Amazon Redshift ML を使用してそのモデルで予測クエリを実行します。SageMaker 線形学習アルゴリズムは、リグレッション問題または分類問題のいずれかを解決します。リグレッション問題と多クラス分類問題の詳細については、「Amazon SageMaker デベロッパーガイド」の「機械学習パラダイムの問題タイプ」を参照してください。このチュートリアルでは、複数クラス分類問題を解決します。線形学習アルゴリズムは、多数のモデルを並列でトレーニングし、最も最適化されたモデルを自動的に決定します。Amazon Redshift で CREATE MODEL オペレーションを使用し、SageMaker で線形学習モデルを作成して、予測関数を Amazon Redshift に送信します。線形学習アルゴリズムの詳細については「Amazon SageMaker デベロッパーガイド」の「線形学習アルゴリズム」を参照してください。

CREATE MODEL コマンドを使用して、トレーニングデータのエクスポート、モデルのトレーニング、モデルのインポート、Amazon Redshift 予測関数の準備を行うことができます。トレーニングデータをテーブルまたは SELECT ステートメントとして指定するには、CREATE MODEL オペレーションを使用します。

線形学習モデルは、連続的な目標または離散的な目標のいずれかを最適化します。連続的な目標はリグレッションに使用され、離散変数は分類に使用されます。リグレッション法など、一部の方法は、連続的な目標のみに対する解を提供します。線形学習アルゴリズムは、Naive Bayes 手法などの単純なハイパーパラメータ最適化手法より高速です。単純な最適化手法では、各入力変数は独立していると仮定します。線形学習アルゴリズムは、多数のモデルを並列でトレーニングし、最も最適化されたモデルを選択します。同様のアルゴリズムが XGBoost です。XGBoost は、より単純で弱いモデルのセットから推定値を組み合わせて予測を行います。XGBoost の詳細については、Amazon SageMaker デベロッパーガイドの「XGBoost アルゴリズム」を参照してください。

線形学習アルゴリズムを使用するには、入力の次元を表す列を指定し、観測値を表す行を指定する必要があります。線形学習アルゴリズムの詳細については「Amazon SageMaker デベロッパーガイド」の「線形学習アルゴリズム」を参照してください。

このチュートリアルでは、特定地域の植生の種類を予測する線形学習モデルを作成します。UCI Machine Learning リポジトリの Covertype データセット で CREATE MODEL コマンドを使用します。次に、コマンドで作成された予測関数を使用して、荒野地域の植生の種類を特定します。森林の種類の種類は、通常、樹木の一種です。Redshift ML がモデルの作成に使用する入力には、土壌タイプ、道路までの距離、および荒野地域の指定が含まれます。データセットの詳細については、UCI Machine Learning リポジトリの Covertype データセットを参照してください。

ユースケースの例

Amazon Redshift ML では、画像から植物の種を予測するなど、線形学習を使用して他の複数クラス分類問題を解決することができます。また、カスタマーが購入する製品の数量を予測することもできます。

タスク

- 前提条件
- ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする
- ステップ 2: 機械学習モデルを作成する
- ステップ 3: モデルを検証する

前提条件

このチュートリアルを完了するには、Amazon Redshift ML の「管理の設定」を完了している必要があります。

ステップ 1: Amazon S3 から Amazon Redshift にデータをロードする

Amazon Redshift クエリエディタv2 を使用する次のクエリを実行します。これらのクエリは、Redshift にサンプルデータをロードし、データをトレーニングセットと検証セットに分割します。

1. 次のクエリは、`covertype_data` テーブルを作成します。

```
CREATE TABLE public.covertype_data (
    elevation bigint ENCODE az64,
```

```
    aspect bigint ENCODE az64,
    slope bigint ENCODE az64,
    horizontal_distance_to_hydrology bigint ENCODE az64,
    vertical_distance_to_hydrology bigint ENCODE az64,
    horizontal_distance_to_roadways bigint ENCODE az64,
    hillshade_9am bigint ENCODE az64,
    hillshade_noon bigint ENCODE az64,
    hillshade_3pm bigint ENCODE az64,
    horizontal_distance_to_fire_points bigint ENCODE az64,
    wilderness_area1 bigint ENCODE az64,
    wilderness_area2 bigint ENCODE az64,
    wilderness_area3 bigint ENCODE az64,
    wilderness_area4 bigint ENCODE az64,
    soil_type1 bigint ENCODE az64,
    soil_type2 bigint ENCODE az64,
    soil_type3 bigint ENCODE az64,
    soil_type4 bigint ENCODE az64,
    soil_type5 bigint ENCODE az64,
    soil_type6 bigint ENCODE az64,
    soil_type7 bigint ENCODE az64,
    soil_type8 bigint ENCODE az64,
    soil_type9 bigint ENCODE az64,
    soil_type10 bigint ENCODE az64,
    soil_type11 bigint ENCODE az64,
    soil_type12 bigint ENCODE az64,
    soil_type13 bigint ENCODE az64,
    soil_type14 bigint ENCODE az64,
    soil_type15 bigint ENCODE az64,
    soil_type16 bigint ENCODE az64,
    soil_type17 bigint ENCODE az64,
    soil_type18 bigint ENCODE az64,
    soil_type19 bigint ENCODE az64,
    soil_type20 bigint ENCODE az64,
    soil_type21 bigint ENCODE az64,
    soil_type22 bigint ENCODE az64,
    soil_type23 bigint ENCODE az64,
    soil_type24 bigint ENCODE az64,
    soil_type25 bigint ENCODE az64,
    soil_type26 bigint ENCODE az64,
    soil_type27 bigint ENCODE az64,
    soil_type28 bigint ENCODE az64,
    soil_type29 bigint ENCODE az64,
    soil_type30 bigint ENCODE az64,
    soil_type31 bigint ENCODE az64,
```

```
    soil_type32 bigint ENCODE az64,
    soil_type33 bigint ENCODE az64,
    soil_type34 bigint ENCODE az64,
    soil_type35 bigint ENCODE az64,
    soil_type36 bigint ENCODE az64,
    soil_type37 bigint ENCODE az64,
    soil_type38 bigint ENCODE az64,
    soil_type39 bigint ENCODE az64,
    soil_type40 bigint ENCODE az64,
    cover_type bigint ENCODE az64
) DISTSTYLE AUTO;
```

2. 次のクエリは、サンプルデータを Amazon S3 の Covertype データセットから Amazon Redshift で以前に作成した covertype_data テーブルにコピーします。

```
COPY public.covertype_data
FROM
    's3://redshift-ml-multiclass/covtype.data.gz' IAM_ROLE DEFAULT gzip DELIMITER ','
 REGION 'us-east-1';
```

3. データを手動で分割することにより、追加のテストセットを割り当てることでモデルの精度を検証できます。次のクエリは、データを 2 つのセットに分割します。covertype_training テーブルはトレーニング用、covertype_validation テーブルは検証用、covertype_test テーブルはモデルのテスト用です。トレーニングセットを使用してモデルをトレーニングし、検証セットを使用してモデルの開発を検証します。次に、テストセットを使用してモデルのパフォーマンスをテストし、モデルがデータセットにオーバーフィットか、またはアンダーフィットかを確認します。

```
CREATE TABLE public.covertype_data_prep AS
SELECT
    a.*,
    CAST (random() * 100 AS int) AS data_group_id
FROM
    public.covertype_data a;

--training dataset
 CREATE TABLE public.covertype_training as
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
```

```
    data_group_id < 80;

--validation dataset
 CREATE TABLE public.covertype_validation AS
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id BETWEEN 80
    AND 89;

--test dataset
 CREATE TABLE public.covertype_test AS
SELECT
    *
FROM
    public.covertype_data_prep
WHERE
    data_group_id > 89;
```

ステップ 2: 機械学習モデルを作成する

このステップでは、CREATE MODEL ステートメントを使用して、線形学習アルゴリズムでの機械学習モデルを作成します。

次のクエリは、S3 バケットを使用して CREATE MODEL オペレーションで線形学習モデルを作成します。amzn-s3-demo-bucket をユーザーの S3 バケットに置き換えます。

```
CREATE MODEL forest_cover_type_model
FROM
    (
        SELECT
            Elevation,
            Aspect,
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
```

```
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
```

```
            Cover_type
        from
            public.covertype_training
    ) TARGET cover_type FUNCTION predict_cover_type IAM_ROLE default MODEL_TYPE
 LINEAR_LEARNER PROBLEM_TYPE MULTICLASS_CLASSIFICATION OBJECTIVE 'Accuracy' SETTINGS (
        S3_BUCKET 'amzn-s3-demo-bucket',
        S3_GARBAGE_COLLECT OFF,
        MAX_RUNTIME 15000
    );
```

モデルトレーニングのステータスを表示する (オプション)

SHOW MODEL コマンドを使用して、モデルの準備が完了したことを知ることができます。

次のクエリを使用して、モデルトレーニングの進行状況を監視します。

```
SHOW MODEL forest_cover_type_model;
```

モデルの準備が整うと、先ほどのオペレーションの出力は、次の例のようになります。出力は、`validation:multiclass_accuracy` メトリクスを提供し、これは次の例の右側に表示できます。複数クラス精度は、モデルによって正しく分類されたデータポイントの割合を測定します。次のステップでは、複数クラス精度を使用してモデルの精度を検証します。

```
+--------------------------------
+----------------------------------------------------------------------------------------------------
+
|              Key              |



 Value





                                                         |
+--------------------------------
+----------------------------------------------------------------------------------------------------
+
| Model Name                    | forest_cover_type_model
```

```
                                                |
| Schema Name                   | public

                                                |
| Owner                         | awsuser

                                                |
| Creation Time                 | Tue, 12.07.2022 20:24:32

                                                |
| Model State                   | READY
```

```
                                                  |
| validation:multiclass_accuracy |

                                          0.724952 |
| Estimated Cost                 |

                                          5.341750 |
|                                |

                                                  |
| TRAINING DATA:                 |

                                                  |
| Query                          | SELECT ELEVATION, ASPECT, SLOPE,
 HORIZONTAL_DISTANCE_TO_HYDROLOGY, VERTICAL_DISTANCE_TO_HYDROLOGY,
```

```
 HORIZONTAL_DISTANCE_TO_ROADWAYS, HILLSHADE_9AM, HILLSHADE_NOON, HILLSHADE_3PM ,
 HORIZONTAL_DISTANCE_TO_FIRE_POINTS, WILDERNESS_AREA1, WILDERNESS_AREA2,
 WILDERNESS_AREA3, WILDERNESS_AREA4, SOIL_TYPE1, SOIL_TYPE2, SOIL_TYPE3, SOIL_TYPE4,
 SOIL_TYPE5, SOIL_TYPE6, SOIL_TYPE7, SOIL_TYPE8, SOIL_TYPE9, SOIL_TYPE10 , SOIL_TYPE11,
 SOIL_TYPE12 , SOIL_TYPE13 , SOIL_TYPE14, SOIL_TYPE15, SOIL_TYPE16, SOIL_TYPE17,
 SOIL_TYPE18, SOIL_TYPE19, SOIL_TYPE20, SOIL_TYPE21, SOIL_TYPE22, SOIL_TYPE23,
 SOIL_TYPE24, SOIL_TYPE25, SOIL_TYPE26, SOIL_TYPE27, SOIL_TYPE28, SOIL_TYPE29,
 SOIL_TYPE30, SOIL_TYPE31, SOIL_TYPE32, SOIL_TYPE33, SOIL_TYPE34, SOIL_TYPE36,
 SOIL_TYPE37, SOIL_TYPE38, SOIL_TYPE39, SOIL_TYPE40, COVER_TYPE |
|                                | FROM PUBLIC.COVERTYPE_TRAINING

                                                   |
| Target Column                  | COVER_TYPE

                                                   |
|                                |

                                                   |
| PARAMETERS:                    |
```

```
                                                      |
| Model Type                      | linear_learner

                                                      |
| Problem Type                    | MulticlassClassification

                                                      |
| Objective                       | Accuracy

                                                      |
| AutoML Job Name                 | redshiftml-20220712202432187659
```

```
                                                        |
| Function Name                 | predict_cover_type

                                                        |
| Function Parameters           | elevation aspect slope
 horizontal_distance_to_hydrology vertical_distance_to_hydrology
 horizontal_distance_to_roadways hillshade_9am hillshade_noon hillshade_3pm
 horizontal_distance_to_fire_points wilderness_area1 wilderness_area2 wilderness_area3
 wilderness_area4 soil_type1 soil_type2 soil_type3 soil_type4 soil_type5 soil_type6
 soil_type7 soil_type8 soil_type9 soil_type10 soil_type11 soil_type12 soil_type13
 soil_type14 soil_type15 soil_type16 soil_type17 soil_type18 soil_type19 soil_type20
 soil_type21 soil_type22 soil_type23 soil_type24 soil_type25 soil_type26 soil_type27
 soil_type28 soil_type29 soil_type30 soil_type31 soil_type32 soil_type33 soil_type34
 soil_type36 soil_type37 soil_type38 soil_type39 soil_type40
                                                      |
| Function Parameter Types      | int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8 int8
 int8 int8 int8 int8 int8 int8 int8 int8 int8

                                                               |
| IAM Role                      | default-aws-iam-role

                                                        |
```

```
| S3 Bucket                        | amzn-s3-demo-bucket

                                                    |
| Max Runtime                      |

                                              15000 |
+--------------------------------
+----------------------------------------------------------------------------------------------------
+
```

ステップ 3: モデルを検証する

1. 次の予測クエリは、複数クラス精度を計算し、 `covertype_validation` データセットでモデルの精度を検証します。複数クラス精度とは、モデルの予測が正しいパーセンテージのことです。

```
SELECT
    CAST(sum(t1.match) AS decimal(7, 2)) AS predicted_matches,
    CAST(sum(t1.nonmatch) AS decimal(7, 2)) AS predicted_non_matches,
    CAST(sum(t1.match + t1.nonmatch) AS decimal(7, 2)) AS total_predictions,
    predicted_matches / total_predictions AS pct_accuracy
FROM
    (
        SELECT
            Elevation,
            Aspect,
            Slope,
            Horizontal_distance_to_hydrology,
            Vertical_distance_to_hydrology,
            Horizontal_distance_to_roadways,
```

```
            HIllshade_9am,
            Hillshade_noon,
            Hillshade_3pm,
            Horizontal_Distance_To_Fire_Points,
            Wilderness_Area1,
            Wilderness_Area2,
            Wilderness_Area3,
            Wilderness_Area4,
            soil_type1,
            Soil_Type2,
            Soil_Type3,
            Soil_Type4,
            Soil_Type5,
            Soil_Type6,
            Soil_Type7,
            Soil_Type8,
            Soil_Type9,
            Soil_Type10,
            Soil_Type11,
            Soil_Type12,
            Soil_Type13,
            Soil_Type14,
            Soil_Type15,
            Soil_Type16,
            Soil_Type17,
            Soil_Type18,
            Soil_Type19,
            Soil_Type20,
            Soil_Type21,
            Soil_Type22,
            Soil_Type23,
            Soil_Type24,
            Soil_Type25,
            Soil_Type26,
            Soil_Type27,
            Soil_Type28,
            Soil_Type29,
            Soil_Type30,
            Soil_Type31,
            Soil_Type32,
            Soil_Type33,
            Soil_Type34,
            Soil_Type36,
            Soil_Type37,
```

```
            Soil_Type38,
            Soil_Type39,
            Soil_Type40,
            Cover_type AS actual_cover_type,
            predict_cover_type(
                Elevation,
                Aspect,
                Slope,
                Horizontal_distance_to_hydrology,
                Vertical_distance_to_hydrology,
                Horizontal_distance_to_roadways,
                HIllshade_9am,
                Hillshade_noon,
                Hillshade_3pm,
                Horizontal_Distance_To_Fire_Points,
                Wilderness_Area1,
                Wilderness_Area2,
                Wilderness_Area3,
                Wilderness_Area4,
                soil_type1,
                Soil_Type2,
                Soil_Type3,
                Soil_Type4,
                Soil_Type5,
                Soil_Type6,
                Soil_Type7,
                Soil_Type8,
                Soil_Type9,
                Soil_Type10,
                Soil_Type11,
                Soil_Type12,
                Soil_Type13,
                Soil_Type14,
                Soil_Type15,
                Soil_Type16,
                Soil_Type17,
                Soil_Type18,
                Soil_Type19,
                Soil_Type20,
                Soil_Type21,
                Soil_Type22,
                Soil_Type23,
                Soil_Type24,
                Soil_Type25,
```

```
                Soil_Type26,
                Soil_Type27,
                Soil_Type28,
                Soil_Type29,
                Soil_Type30,
                Soil_Type31,
                Soil_Type32,
                Soil_Type33,
                Soil_Type34,
                Soil_Type36,
                Soil_Type37,
                Soil_Type38,
                Soil_Type39,
                Soil_Type40
            ) AS predicted_cover_type,
            CASE
                WHEN actual_cover_type = predicted_cover_type THEN 1
                ELSE 0
            END AS match,
            CASE
                WHEN actual_cover_type <> predicted_cover_type THEN 1
                ELSE 0
            END AS nonmatch
        FROM
            public.covertype_validation
    ) t1;
```

前のクエリの出力は次の例のようになります。複数クラス精度の値は、SHOW MODEL オペレーションの出力によって示される `validation:multiclass_accuracy` メトリクスに似ています。

```
+-------------------+-----------------------+-------------------+--------------+
| predicted_matches | predicted_non_matches | total_predictions | pct_accuracy |
+-------------------+-----------------------+-------------------+--------------+
|             41211 |                 16324 |             57535 |   0.71627704 |
+-------------------+-----------------------+-------------------+--------------+
```

2. 次のクエリは、`wilderness_area2` の最も一般的な植生の種類を予測します。このデータセットには、4 つの荒野地域と 7 つの植生の種類が含まれています。荒野地域は複数の植生の種類が存在する場合があります。

```
SELECT t1. predicted_cover_type, COUNT(*)
```

```
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
   Horizontal_distance_to_hydrology,
   Vertical_distance_to_hydrology,
   Horizontal_distance_to_roadways,
   HIllshade_9am,
   Hillshade_noon,
   Hillshade_3pm ,
   Horizontal_Distance_To_Fire_Points,
   Wilderness_Area1,
   Wilderness_Area2,
   Wilderness_Area3,
   Wilderness_Area4,
   soil_type1,
   Soil_Type2,
   Soil_Type3,
   Soil_Type4,
   Soil_Type5,
   Soil_Type6,
   Soil_Type7,
   Soil_Type8,
   Soil_Type9,
   Soil_Type10 ,
   Soil_Type11,
   Soil_Type12 ,
   Soil_Type13 ,
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
   Soil_Type23,
   Soil_Type24,
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
```

```
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
```

```
    Soil_Type19,
    Soil_Type20,
    Soil_Type21,
    Soil_Type22,
    Soil_Type23,
    Soil_Type24,
    Soil_Type25,
    Soil_Type26,
    Soil_Type27,
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40) AS predicted_cover_type

FROM public.covertype_test
WHERE wilderness_area2 = 1)
t1
GROUP BY 1;
```

先ほどのオペレーションの出力は、次の例のようになります。この出力は、モデルが、植生の大部分が植生 1 の種類であり、一部が植生2と 7 の種類である予測したことを意味します。

```
+----------------------+-------+
| predicted_cover_type | count |
+----------------------+-------+
|                    2 |   564 |
|                    7 |    97 |
|                    1 |  2309 |
+----------------------+-------+
```

3. 次のクエリは、1 つの荒野地域で最も一般的な植生の種類を示しています。このクエリは、その植生の種類の量とその植生の種類の荒野地域を表示します。

```
SELECT t1. predicted_cover_type, COUNT(*), wilderness_area
```

```
FROM
(
SELECT
   Elevation,
   Aspect,
   Slope,
   Horizontal_distance_to_hydrology,
   Vertical_distance_to_hydrology,
   Horizontal_distance_to_roadways,
   HIllshade_9am,
   Hillshade_noon,
   Hillshade_3pm ,
   Horizontal_Distance_To_Fire_Points,
   Wilderness_Area1,
   Wilderness_Area2,
   Wilderness_Area3,
   Wilderness_Area4,
   soil_type1,
   Soil_Type2,
   Soil_Type3,
   Soil_Type4,
   Soil_Type5,
   Soil_Type6,
   Soil_Type7,
   Soil_Type8,
   Soil_Type9,
   Soil_Type10 ,
   Soil_Type11,
   Soil_Type12 ,
   Soil_Type13 ,
   Soil_Type14,
   Soil_Type15,
   Soil_Type16,
   Soil_Type17,
   Soil_Type18,
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
   Soil_Type23,
   Soil_Type24,
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
```

```
    Soil_Type28,
    Soil_Type29,
    Soil_Type30,
    Soil_Type31,
    Soil_Type32,
    Soil_Type33,
    Soil_Type34,
    Soil_Type36,
    Soil_Type37,
    Soil_Type38,
    Soil_Type39,
    Soil_Type40,
    predict_cover_type( Elevation,
    Aspect,
    Slope,
    Horizontal_distance_to_hydrology,
    Vertical_distance_to_hydrology,
    Horizontal_distance_to_roadways,
    HIllshade_9am,
    Hillshade_noon,
    Hillshade_3pm ,
    Horizontal_Distance_To_Fire_Points,
    Wilderness_Area1,
    Wilderness_Area2,
    Wilderness_Area3,
    Wilderness_Area4,
    soil_type1,
    Soil_Type2,
    Soil_Type3,
    Soil_Type4,
    Soil_Type5,
    Soil_Type6,
    Soil_Type7,
    Soil_Type8,
    Soil_Type9,
    Soil_Type10,
    Soil_Type11,
    Soil_Type12,
    Soil_Type13,
    Soil_Type14,
    Soil_Type15,
    Soil_Type16,
    Soil_Type17,
    Soil_Type18,
```

```
   Soil_Type19,
   Soil_Type20,
   Soil_Type21,
   Soil_Type22,
   Soil_Type23,
   Soil_Type24,
   Soil_Type25,
   Soil_Type26,
   Soil_Type27,
   Soil_Type28,
   Soil_Type29,
   Soil_Type30,
   Soil_Type31,
   Soil_Type32,
   Soil_Type33,
   Soil_Type34,
   Soil_Type36,
   Soil_Type37,
   Soil_Type38,
   Soil_Type39,
   Soil_Type40) AS predicted_cover_type,
   CASE WHEN Wilderness_Area1 = 1 THEN 1
        WHEN Wilderness_Area2 = 1 THEN 2
        WHEN Wilderness_Area3 = 1 THEN 3
        WHEN Wilderness_Area4 = 1 THEN 4
        ELSE 0
   END AS wilderness_area

FROM public.covertype_test)
t1
GROUP BY 1, 3
ORDER BY 2 DESC
LIMIT 1;
```

先ほどのオペレーションの出力は、次の例のようになります。

```
+----------------------+-------+-----------------+
| predicted_cover_type | count | wilderness_area |
+----------------------+-------+-----------------+
|                    2 | 15738 |               1 |
+----------------------+-------+-----------------+
```

関連トピック

Amazon Redshift ML の詳細については、次のドキュメントを参照してください。

- Amazon Redshift 機械学習を使用するためのコスト
- CREATE MODEL オペレーション
- EXPLAIN_MODEL 関数

機械学習の詳細については、以下のドキュメントを参照してください。

- 機械学習の概要
- 初心者やエキスパート向けの機械学習
- 機械学習予測の公平性とモデルの説明可能性とは

Amazon Redshift ML と Amazon Bedrock の統合

このセクションでは、Amazon Redshift ML と Amazon Bedrock の統合を使用する方法について説明します。この機能では、SQL を使用して Amazon Bedrock モデルを呼び出すことができ、Amazon Redshift データウェアハウスのデータを使用して、テキスト生成、感情分析、翻訳などの生成 AI アプリケーションを構築できます。

トピック

- Amazon Redshift ML と Amazon Bedrock の統合のための IAM ロールの作成または更新
- Amazon Redshift ML と Amazon Bedrock の統合用の外部モデルの作成
- Amazon Redshift ML と Amazon Bedrock の統合用の外部モデルの使用
- Amazon Redshift ML と Amazon Bedrock の統合のためのプロンプトエンジニアリング

Amazon Redshift ML と Amazon Bedrock の統合のための IAM ロールの作成または更新

このセクションでは、Amazon Redshift ML と Amazon Bedrock の統合で使用する IAM ロールの作成方法について説明します。

Amazon Redshift ML と Amazon Bedrock の統合で使用する IAM ロールに次のポリシーを追加します。

- `AmazonBedrockFullAccess`

Amazon Redshift が他のサービスとやり取りするロールを引き受けることを許可するには、IAM ロールに以下の信頼ポリシーを追加します。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": [
               "redshift.amazonaws.com"
            ]
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

クラスターまたは名前空間が VPC にある場合は、「Amazon Redshift ML 管理者によるクラスターと設定のセットアップ」の手順に従います。

より制限の厳しいポリシーが必要な場合は、以下のページで指定された Amazon Bedrock アクセス許可のみを含むポリシーを作成できます。

- Amazon Redshift ML 管理者によるクラスターと設定のセットアップ
- Amazon Redshift 機械学習 (ML) を使用するために必要なアクセス許可

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

Amazon Redshift ML と Amazon Bedrock の統合用の外部モデルの作成

このセクションでは、Amazon Redshift データウエアハウス内で Amazon Bedrock のインターフェイスとして使用する外部モデルを作成する方法について説明します。

Amazon Redshift から Amazon Bedrock モデルを呼び出すには、まず `CREATE EXTERNAL MODEL` コマンドを実行する必要があります。このコマンドは、データベースに外部モデルオブジェクト

と、Amazon Bedrock でテキストコンテンツを生成するために使用される関連するユーザー関数を作成します。

次のコード例は、基本的な CREATE EXTERNAL MODEL コマンドを示しています。

```
CREATE EXTERNAL MODEL llm_claude
FUNCTION llm_claude_func
IAM_ROLE '<IAM role arn>'
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID 'anthropic.claude-v2:1',
   PROMPT 'Summarize the following text:');
```

CREATE EXTERNAL MODEL コマンドには、メッセージをサポートするすべての基盤モデル (FM) 向けの Amazon Bedrock との統一され一貫したインターフェイスがあります。これは、 CREATE EXTERNAL MODEL コマンドを使用する場合、またはリクエストタイプを UNIFIED に明示的に指定する場合のデフォルトのオプションです。詳細については、「Amazon Bedrock API ドキュメント」の「Converse API documentation」を参照してください。

FM がメッセージをサポートしていない場合は、request_type 設定を RAW に設定する必要があります。request_type を RAW に設定する場合、選択した FM に基づいて推論関数を使用する際、Amazon Bedrock に送信するリクエストを作成する必要があります。

CREATE EXTERNAL MODEL コマンドの PROMPT パラメータは静的プロンプトです。アプリケーションで動的プロンプトが必要な場合は、推論関数を使用する際に指定する必要があります。詳細については、以下の「Amazon Redshift ML と Amazon Bedrock の統合のためのプロンプトエンジニアリング」を参照してください。

CREATE EXTERNAL MODEL ステートメントとそのパラメータ、および設定の詳細については、「CREATE EXTERNAL MODEL」を参照してください。

Amazon Redshift ML と Amazon Bedrock の統合用の外部モデルの使用

このセクションでは、提供されたプロンプトに応答して外部モデルを呼び出してテキストを生成する方法について説明します。外部モデルを呼び出すには、CREATE EXTERNAL MODEL で作成した推論関数を使用します。

トピック

- UNIFIED リクエストタイプモデルによる推論
- RAW リクエストタイプモデルによる推論

- [リーダーのみの関数としての推論関数]
- [推論関数の使用に関する注意事項]

UNIFIED リクエストタイプモデルによる推論

UNIFIED リクエストタイプのモデルを使用した推論関数には、関数に順番に渡される次の 3 つのパラメータがあります。

- 入力テキスト (必須): このパラメータは、Amazon Redshift が Amazon Bedrock に渡す入力テキストを指定します。
- 推論設定と追加のモデルリクエストフィールド (オプション): Amazon Redshift はこれらのパラメータを Converse モデル API の対応するパラメータに渡します。

次のコード例は、UNIFIED タイプの推論関数の使用方法を示しています。

```
SELECT llm_claude_func(input_text, object('temperature', 0.7, 'maxtokens', 500))
   FROM some_data;
```

RAW リクエストタイプモデルによる推論

RAW リクエストタイプのモデルを使用した推論関数には、SUPER データタイプのパラメータが 1 つだけあります。このパラメータの構文は、使用する Amazon Bedrock モデルによって異なります。

次のコード例は、RAW タイプの推論関数の使用方法を示しています。

```
SELECT llm_titan_func(
    object(
        "inputText", "Summarize the following text: " | input_text,
        "textGenerationConfig", object("temperature", 0.5, "maxTokenCount", 500)
    )
)
FROM some_data;
```

リーダーのみの関数としての推論関数

Amazon Bedrock モデルの推論関数は、それらを使用するクエリがテーブルを参照しない場合、リーダーノードのみの関数として実行できます。これは、LLM にすぐに質問をしたい場合に役立ちます。

次のコード例は、リーダーのみの推論関数の使用方法を示しています。

```
SELECT general_titan_llm_func('Summarize the benefits of LLM on data analytics in 100
 words');
```

推論関数の使用に関する注意事項

Amazon Redshift ML と Amazon Bedrock の統合で推論関数を使用する場合は、以下の点に注意してください。

- すべての Amazon Bedrock モデルのパラメータ名では、大文字と小文字が区別されます。パラメータがモデルに必要なパラメータと一致しない場合、Amazon Bedrock はアラートなしにパラメータを無視することがあります。
- 推論クエリのスループットは、Amazon Bedrock がさまざまなリージョンで提供するさまざまなモデルのランタイムクォータによって制限されます。詳細については、「Amazon Bedrock ユーザーガイド」の「Quotas for Amazon Bedrock」を参照してください。
- 保証された一貫したスループットが必要な場合は、Amazon Bedrock から必要なモデルのプロビジョニングされたスループットを取得することを検討してください。詳細については、「Amazon Bedrock ユーザーガイド」の「Increase model invocation capacity with Provisioned Throughput in Amazon Bedrock」を参照してください。
- 大量のデータを含む推論クエリでは、スロットリング例外が発生する可能性があります。これは、Amazon Bedrock のランタイムクォータが限られているためです。Amazon Redshift は複数回リクエストを再試行しますが、プロビジョニングされていないモデルのスループットは可変である可能性があるため、クエリはスロットリングされる可能性があります。

Amazon Redshift ML と Amazon Bedrock の統合のためのプロンプトエンジニアリング

このセクションでは、外部モデルで静的プロンプトを使用する方法について説明します。

外部モデルで静的プレフィックスおよび静的サフィックスプロンプトを使用するには、`CREATE EXTERNAL MODEL` ステートメントの `PROMPT` および `SUFFIX` パラメータを使用してそれらを指定します。これらのプロンプトは、外部モデルを使用してすべてのクエリに追加されます。

次の例は、外部モデルにプレフィックスおよびサフィックスプロンプトを追加する方法を示しています。

```
CREATE EXTERNAL MODEL llm_claude
FUNCTION llm_claude_func
IAM_ROLE '<IAM role arn>'
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID 'anthropic.claude-v2:1',
   PROMPT 'Summarize the following text:',
   SUFFIX 'Respond in an analytic tone');
```

動的プロンプトについては、関数入力で連結して推論関数を使用する際に指定できます。次の例は、推論関数で動的プロンプトを使用する方法を示しています。

```
SELECT llm_claude_func('Summarize the following review:' | input_text | 'The review
 should have formal tone.')
FROM some_data
```

クエリパフォーマンスのチューニング

Amazon Redshift は、構造化クエリ言語 (SQL) に基づくクエリを使用して、システム内のデータおよびオブジェクトとやり取りします。データ操作言語 (DML) は、データの表示、追加、変更、削除に使用する SQL のサブセットです。データ定義言語 (DDL) は、テーブルやビューなどのデータベースオブジェクトの追加、変更、削除に使用する SQL のサブセットです。

システムがセットアップされると、通常は DML を最も多く使用します (特にデータを取得および表示する SELECT コマンド)。Amazon Redshift で有効なデータ取得クエリを記述するには、SELECT についてよく理解し、Amazon Redshift テーブル設計のベストプラクティスにまとめられているヒントを当てはめてクエリの効果を最大限に高めてください。

Amazon Redshift がクエリを処理する方法について理解するには、クエリ処理セクションと クエリの分析と改善 セクションを使用します。この情報を診断ツールと組み合わせて適用することで、クエリのパフォーマンスに関する問題を特定して取り除くことができます。

Amazon Redshift クエリで発生する可能性のある一般的な問題と重大な問題を特定して対処するには、クエリのトラブルシューティングセクションを使用します。

トピック

- クエリ処理
- クエリの分析と改善
- クエリのトラブルシューティング

クエリ処理

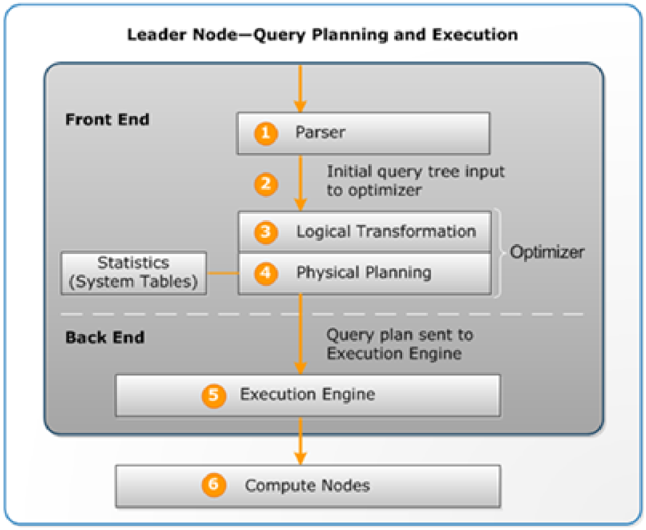
Amazon Redshift は、パーサーおよびオプティマイザを通じて送信された SQL クエリをルーティングし、クエリプランを作成します。その後、実行エンジンは、クエリプランをコードに変換し、そのコードを実行するためにコンピューティングノードに送信します。

トピック

- クエリプランと実行ワークフロー
- クエリプランの作成と解釈
- クエリプランステップの確認
- クエリパフォーマンスに影響を与える要因

クエリプランと実行ワークフロー

次の図は、クエリの計画と実行のワークフローの概要です。



クエリプランと実行ワークフローは以下のステップに従います。

1. リーダーノードはクエリを受け取り、SQL を解析します。
2. クエリツリーパーサーは、元のクエリの論理的な表現である初期クエリツリーを生成します。次に、Amazon Redshift は、このクエリツリーをクエリオプティマイザに入力します。
3. オプティマイザは、クエリを評価し、必要に応じて書き換えて効率を最大限に高めます。このプロセスにより、関連するクエリが複数作成されて、単一のクエリが置き換えられることがあります。
4. オプティマイザは、最高のパフォーマンスで実行されるように 1 つのクエリプラン (または、前のステップで複数のクエリが生成された場合は複数のクエリプラン) を生成します。クエリプランは、結合の種類、結合の順序、集計オプション、データ分散要件などの実行オプションを指定します。

   クエリプランを表示するには、EXPLAINコマンドを使用できます。クエリプランは、複雑なクエリを分析およびチューニングするための基本ツールです。詳細については、「クエリプランの作成と解釈」を参照してください。
5. 実行エンジンは、クエリプランをステップ、セグメント、ストリームに変換します。

   [ステップ]

      各ステップは、クエリ実行時に必要な別個の操作です。コンピューティングノードはステップを組み合わせることによってクエリ、結合、または他のデータベース操作を実行できます。

Segment

1 つのプロセスで実行できる複数のステップの組み合わせ。コンピューティングノードスライスによって実行可能な最小コンパイル単位でもあります。スライスは、Amazon Redshift の並列処理単位です。並行して実行されるストリーム内のセグメント。

ストリーム

使用できるコンピューティングノードスライスに並列化するセグメントのコレクション。

実行エンジンは、ステップ、セグメント、ストリームに基づいてコンパイルされたコードを生成します。コンパイルされたコードの実行は解釈されたコードよりも速く、使用するコンピューティングキャパシティも少なくなります。その後、このコンパイル済みコードは、コンピューティングノードにブロードキャストされます。

> **Note**
>
> クエリのベンチマークを行うときは、常に、クエリの 2 回目の実行の時間を比較する必要があります。1 回目の実行の時間には、コードをコンパイルするオーバーヘッドが含まれるためです。詳細については、「クエリパフォーマンスに影響を与える要因」を参照してください。

6. コンピューティングノードスライスは、クエリセグメントを並列的に実行します。このプロセスの一部として、Amazon Redshift は、最適化されたネットワーク通信、メモリ、ディスク管理を利用して、クエリプランのステップから次のステップに中間結果を渡します。これは、クエリ実行の高速化にも役立ちます。

ステップ 5 と 6 はストリームごとに 1 回ずつ行われます。エンジンは、1 つのストリームに対して実行可能なセグメントを作成し、コンピューティングノードに送信します。そのストリームのセグメントが完了したら、エンジンは次のストリームのセグメントを生成します。これにより、エンジンは前のストリームで何が発生したかを分析し (操作がディスクベースであったかどうかなど)、次のストリーム内におけるセグメントの生成に影響を与えることができます。

コンピューティングノードは、完了すると最終処理を行うためクエリの結果をリーダーノードに返します。リーダーノードは、データを 1 つの結果セットにマージし、必要なソートまたは集計すべてに対処します。次に、リーダーノードは結果をクライアントに返します。

> **Note**
> コンピューティングノードは、必要に応じてクエリの実行中に一部のデータをリーダーノードに返すことがあります。例えば、LIMIT 句を含むサブクエリがある場合、さらに処理を行うためデータがクラスターに再分散される前に制限がリーダーノードに適用されます。

クエリプランの作成と解釈

クエリプランを使用して、クエリを実行するために必要な個々のオペレーションに関する情報を取得することができます。クエリプランを操作する前に、まず Amazon Redshift がクエリの処理とクエリプランの作成を扱う方法を理解することをお勧めします。詳細については、「クエリプランと実行ワークフロー」を参照してください。

クエリプランを作成するには、EXPLAINコマンドに続いて実際のクエリテキストを指定します。クエリプランから、次の情報が提供されます。

- 実行エンジンが実行する操作 (下から順に結果を読み取る)。
- 各操作で実行されるステップの種類。
- 各操作で使用されるテーブルと列。
- 各操作で処理されるデータの量 (行数とバイト単位のデータの幅)
- 操作の相対的なコスト。コストは、プラン内のステップの相対的な実行回数を比較する評価基準です。コストは、実際の実行回数やメモリの消費に関する正確な情報を提供したり、実行プラン間の意味のある比較を提供したりするわけではありません。ほとんどのリソースを消費しているクエリ内の操作を示しています。

EXPLAIN コマンドは実際にクエリを実行しません。クエリが現在の操作条件の下で実行される場合に Amazon Redshift が実行するプランが表示されるだけです。テーブルのスキーマまたはデータを変更し、ANALYZEをもう一度実行して統計メタデータを更新した場合、クエリプランが異なる可能性があります。

EXPLAIN によるクエリプラン出力は簡略化され、クエリ実行の概要になっています。並行クエリ処理の詳細を示すわけではありません。詳細な情報を表示するには、クエリ自体を実行した後、SVL_QUERY_SUMMARY ビューまたは SVL_QUERY_REPORT ビューからクエリの概要情報を取得します。これらのビューの使用の詳細については、「クエリの概要の分析」を参照してください。

次の例に、EVENT テーブルでの簡単な GROUP BY クエリの EXPLAIN 出力を示します。

```
explain select eventname, count(*) from event group by eventname;

                            QUERY PLAN
-------------------------------------------------------------------
XN HashAggregate  (cost=131.97..133.41 rows=576 width=17)
  ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=17)
```

EXPLAIN は、操作ごとに次のメトリクスを返します。

Cost

プラン内で操作を比較する際に役立つ相対的な値。コストは、2 つのピリオドで区切られた 2 つの小数点値で構成されます (`cost=131.97..133.41` など)。最初の値 (この場合は 131.97) は、この操作の最初の行を返す相対コストを示します。2 番目の値 (この場合は 133.41) は、操作を完了する相対コストを示します。クエリプランのコストはプランを調べる際に累積されるので、この例の HashAggregate コスト (131.97..133.41) には、その下の Seq Scan コスト (0.00..87.98) が含まれます。

Rows

返される行数の推定値。この例では、スキャンにより 8798 行が返されることが予期されます。HashAggregate 演算子自体は、576 行 (重複したイベント名より後は結果セットから破棄されます) を返すことが予期されます。

> **Note**
>
> 行の予測は、ANALYZE コマンドによって生成された、利用可能な統計情報に基づいています。ANALYZE が最近実行されていない場合、予測の信頼性は下がります。

Width

バイト単位の、予測される平均的な行の幅。この例では、平均的な行の幅は 17 バイトであると予期されます。

EXPLAIN の演算子

このセクションでは、EXPLAIN 出力で最もよく使用される演算子を簡単に示します。演算子の詳細なリストについては、SQL コマンドセクションの「EXPLAIN」を参照してください。

Sequential Scan 演算子

Sequential Scan 演算子 (Seq Scan) は、テーブルスキャンを示します。Seq Scan はテーブル内の各列を最初から最後まで連続的にスキャンし、各行のクエリ制約を (WHERE 句で) 評価します。

Join 演算子

Amazon Redshift は、結合されるテーブルの物理的な設計、結合に必要なデータの場所、クエリ固有の要件に基づいて、結合演算子を選択します。

- Nested Loop

  最適性が最も低いネステッドループは、主にクロス結合 (デカルト積) および一部の不等結合に使用されます。
- Hash Join および Hash

  ハッシュ結合およびハッシュは、通常はネステッドループ結合よりも高速で、内部結合および左右の外部結合に使用されます。これらの演算子は、結合列が分散キーでもソートキーでもないテーブルを結合するときに使用されます。ハッシュ演算子は結合の内部テーブルのハッシュテーブルを作成します。ハッシュ結合演算子は外部テーブルを読み取り、結合列をハッシュし、内部ハッシュテーブルで一致を検索します。
- Merge Join

  通常、結合が高速であれば、内部結合と外部結合にマージ結合が使用されます。マージ結合は完全結合には使用されません。この演算子は、結合列が分散キーとソートキーの両方である結合テーブルで、結合テーブルの 20 % 未満がソートされていない場合に使用されます。また、ソートされた 2 つのテーブルを順に読み取り、一致する行を検索します。ソートされていない列の割合を表示するには、SVV_TABLE_INFOシステムテーブルをクエリします。
- 空間結合

  通常、空間データの近接性に基づく高速結合であり、`GEOMETRY` と `GEOGRAPHY` のデータ型に使用されます。

Aggregate 演算子

クエリプランは、集計関数および GROUP BY 操作を含むクエリで次の演算子を使用します。

- Aggregate

  AVG や SUM などのスカラー集計関数の演算子。

- HashAggregate

  未ソートのグループ化された集計関数の演算子。

- GroupAggregate

  ソート済みのグループ化された集計関数の演算子。

ソート演算子

クエリプランは、クエリで結果セットをソートまたは結合する必要があるときに次の演算子を使用します。

- 並べ替え

  ORDER BY 句およびその他のソート操作 (UNION クエリや結合、SELECT DISTINCT クエリ、ウィンドウ関数で必要となるソートなど) を評価します。

- Merge

  並行操作から導出される、ソートされた中間結果に従って最終的なソート結果を作成します。

UNION、INTERSECT、および EXCEPT 演算子

クエリプランは、UNION, INTERSECT、および EXCEPT を使用したセット操作を含むクエリに次の演算子を使用します。

- サブクエリ

  UNION クエリを実行するのに使用されます。

- Hash Intersect [Distinct]

  INTERSECT クエリを実行するために使用されます。

- SetOp Except

  EXCEPT (または MINUS) クエリの実行に使用されます。

その他の演算子

次の演算子は、ルーチンクエリの EXPLAIN 出力にも頻繁に出現します。

- Unique

  SELECT DISTINCT クエリと UNION クエリの重複を削除します。
- 制限

  LIMIT 句を処理します。
- Window

  ウィンドウ関数を実行します。
- 結果

  テーブルアクセスを伴わないスカラー関数を実行します。
- Subplan

  特定のサブクエリに使用されます。
- ネットワーク

  さらに処理するために中間結果をリーダーノードに送ります。
- Materialize

  ネストループ結合および一部のマージ結合への入力のために行を保存します。

EXPLAIN での結合

クエリオプティマイザは、クエリと基礎となるテーブルの構造に応じて、異なる結合の種類を使用してテーブルデータを取得します。EXPLAIN 出力は、結合の種類、使用するテーブル、およびクラスター全体にテーブルデータが分散される方法を参照して、クエリの処理方法を説明します。

結合の種類の例

次の例は、クエリオプティマイザが使用できるさまざまな結合の種類を示しています。クエリプランに使用される結合の種類は、関係するテーブルの物理的な設計によって異なります。

例: 2 つのテーブルのハッシュ結合

次のクエリは、CATID 列で EVENT と CATEGORY を結合します。CATID は CATEGORY の分散キーおよびソートキーですが、EVENT の分散キーおよびソートキーではありません。ハッシュ結合は、EVENT を外部テーブルとして、CATEGORY を内部テーブルとして実行されま

す。CATEGORY は小さいテーブルであるため、プランナーは DS_BCAST_INNER を使用してクエリ処理中にそのコピーをコンピューティングノードにブロードキャストします。この例の結合コストは、プランの累積コストのほとんどを占めます。

```
explain select * from category, event where category.catid=event.catid;

                               QUERY PLAN
-------------------------------------------------------------------------
 XN Hash Join DS_BCAST_INNER  (cost=0.14..6600286.07 rows=8798 width=84)
   Hash Cond: ("outer".catid = "inner".catid)
   ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
   ->  XN Hash  (cost=0.11..0.11 rows=11 width=49)
         ->  XN Seq Scan on category  (cost=0.00..0.11 rows=11 width=49)
```

> **Note**
>
> EXPLAIN 出力の演算子が揃ってインデントされていることは、それらの操作が相互に依存せず、並行して開始できることを示している場合があります。前の例では、EVENT テーブルおよびハッシュ操作のスキャンは揃っていませんが、EVENT スキャンはハッシュ操作が完全に完了するまで待機する必要があります。

例: 2 つのテーブルのマージ結合

次のクエリでも SELECT * が使用されますが、LISTID が両方のテーブルの分散キーおよびソートキーとして設定されている LISTID 列の SALES と LISTING が結合されます。マージ結合が選択され、結合にデータの再分散は必要ありません (DS_DIST_NONE)。

```
explain select * from sales, listing where sales.listid = listing.listid;
QUERY PLAN
-----------------------------------------------------------------------------
XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
  Merge Cond: ("outer".listid = "inner".listid)
  ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
  ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
```

次の例では、同じクエリ内の異なる種類の結合を示します。前の例と同じように、SALES と LISTING はマージ結合されますが、3 番目のテーブルである EVENT は、マージ結合の結果とハッシュ結合される必要があります。ハッシュ結合は、ここでもブロードキャストのコストを発生させます。

```
explain select * from sales, listing, event
where sales.listid = listing.listid and sales.eventid = event.eventid;
                                  QUERY PLAN
----------------------------------------------------------------------------
XN Hash Join DS_BCAST_INNER  (cost=109.98..3871130276.17 rows=172456 width=132)
  Hash Cond: ("outer".eventid = "inner".eventid)
  ->  XN Merge Join DS_DIST_NONE  (cost=0.00..6285.93 rows=172456 width=97)
        Merge Cond: ("outer".listid = "inner".listid)
        ->  XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=44)
        ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=53)
  ->  XN Hash  (cost=87.98..87.98 rows=8798 width=35)
        ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=35)
```

例: 結合、集計、およびソート

以下のクエリは、SALES および EVENT テーブルのハッシュ結合を実行し、次に集計およびソートオペレーションを実行して、グループ化された SUM 関数および ORDER BY 句を計上します。最初の Sort 演算子はコンピューティングノードで並行に実行されます。次に、Network 演算子は結果をリーダーノードに送ります。リーダーノードでは、Merge 演算子がソートされた最終的な結果を作成します。

```
explain select eventname, sum(pricepaid) from sales, event
where sales.eventid=event.eventid group by eventname
order by 2 desc;
                                           QUERY PLAN
---------------------------------------------------------------------------------
 XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
        Send to leader
        ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
              Sort Key: sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                          Hash Cond: ("outer".eventid = "inner".eventid)
                          ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                          ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                                ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

データの再分散

結合の EXPLAIN 出力は、結合を容易にするためにクラスター周囲にデータを移動する方法も指定します。このデータの移動は、ブロードキャストまたは再分散のいずれかとすることができます。ブロードキャストでは、結合の一方の側からのデータ値は、各コンピューティングノードから他方の各コンピューティングノードにコピーされ、各コンピューティングノードはデータの完全なコピーで終了します。再分散では、特定のデータ値は現在のスライスから新しいスライス (別のノードにある場合があります) に送信されます。通常、データは再分散され、結合で該当する他方のテーブルの分散キーと一致します (その分散キーが結合する列の 1 つである場合)。いずれのテーブルにも、結合する列の 1 つに分散キーがない場合、両方のテーブルが分散されるか、内部テーブルが各ノードにブロードキャストされます。

EXPLAIN 出力は、内部テーブルと外部テーブルも参照します。内部テーブルが最初にスキャンされ、クエリプランの下部付近に表示されます。内部テーブルは、一致について調査されるテーブルです。通常、このテーブルはメモリに保持されますが、通常はハッシュのソーステーブルであり、可能な場合は、結合される 2 つのテーブルのうち小さい方になります。外部テーブルは、内部テーブルに対して一致させる行のソースです。通常はディスクから読み込まれます。クエリオプティマイザは、前回実行した ANALYZE コマンドから生成されたデータベース統計情報に基づいて内部テーブルと外部テーブルを選択します。クエリの FROM 句のテーブルの順序により、どのテーブルが内部でどのテーブルが外部かは決まりません。

結合を容易にするためにデータを移動する方法を特定するには、クエリプランで次の属性を使用します。

- DS_BCAST_INNER

  内部テーブル全体のコピーがすべてのコンピューティングノードにブロードキャストされます。
- DS_DIST_ALL_NONE

  内部テーブルは DISTSTYLE ALL を使用してすべてのノードにすでに分散されているため、再分散は不要です。
- DS_DIST_NONE

  テーブルは再分散されない。対応するスライスがノード間でのデータ移動なしで結合されるため、共存結合が可能となる。
- DS_DIST_INNER

  内部テーブルが再分散されます。
- DS_DIST_OUTER

外部テーブルが再分散されます。

- DS_DIST_ALL_INNER

  外部テーブルで DISTSTYLE ALL が使用されるため、内部テーブル全体が単一スライスに再分散されます。

- DS_DIST_BOTH

  両方のテーブルが再分散されます。

クエリプランステップの確認

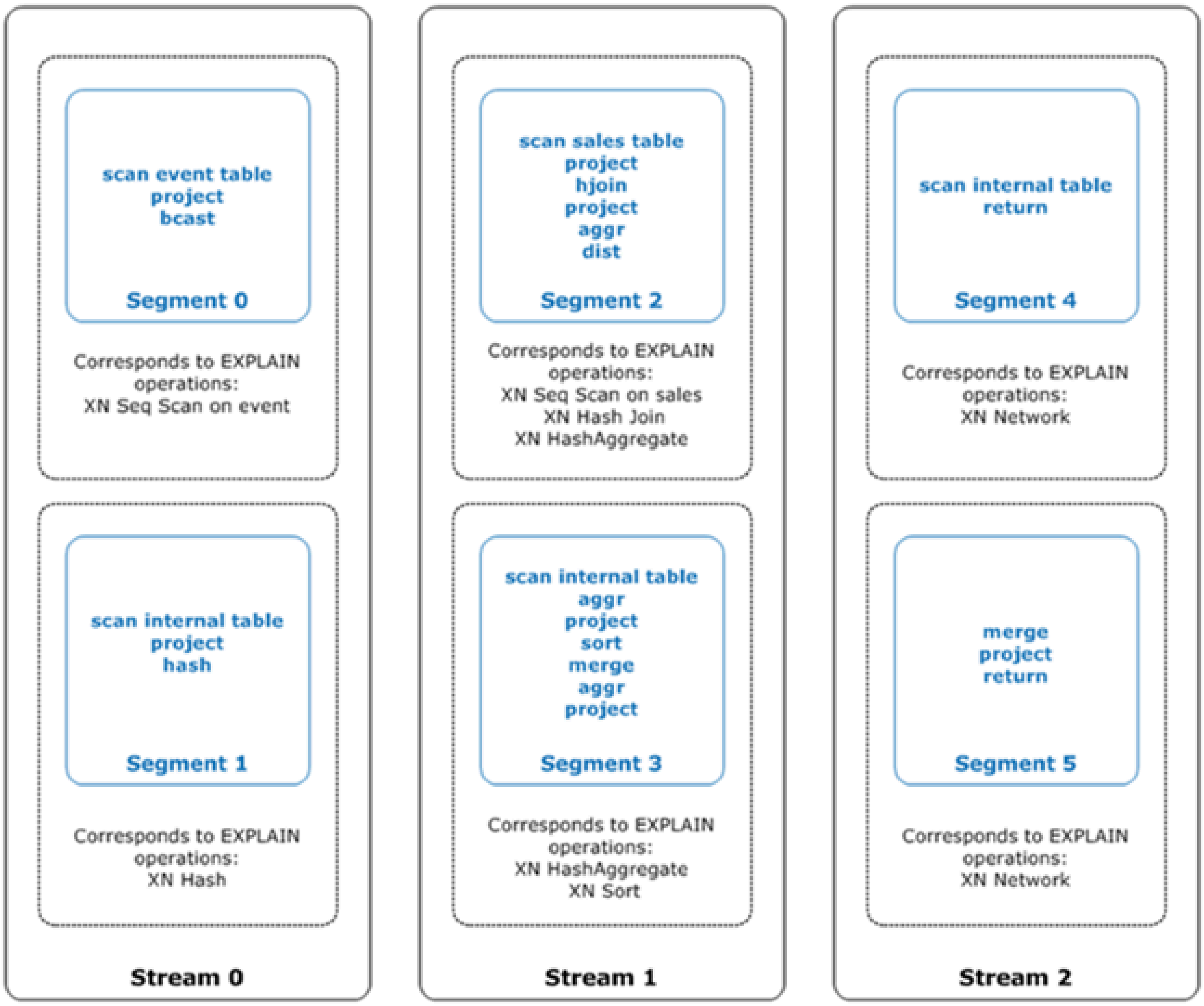
EXPLAIN コマンドを実行して、クエリプランのステップを確認できます。次の例では、SQL クエリとその出力について説明します。クエリプランを最初から読むと、クエリを実行するのに必要な各論理操作について理解できます。詳細については、「クエリプランの作成と解釈」を参照してください。

```
explain
select eventname, sum(pricepaid) from sales, event
where sales.eventid = event.eventid
group by eventname
order by 2 desc;
```

```
XN Merge  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
  Merge Key: sum(sales.pricepaid)
  ->  XN Network  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
        Send to leader
        ->  XN Sort  (cost=1002815366604.92..1002815366606.36 rows=576 width=27)
              Sort Key: sum(sales.pricepaid)
              ->  XN HashAggregate  (cost=2815366577.07..2815366578.51 rows=576
 width=27)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=109.98..2815365714.80
 rows=172456 width=27)
                          Hash Cond: ("outer".eventid = "inner".eventid)
                          ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456
 width=14)
                          ->  XN Hash  (cost=87.98..87.98 rows=8798 width=21)
                                ->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798
 width=21)
```

クエリプランの生成の一環として、クエリオプティマイザーはプランをストリーム、セグメント、ステップに分割します。クエリオプティマイザーは、プランを分割して、データとクエリワークロードを計算ノードに分散する準備をします。ストリーム、セグメントおよびステップの詳細については、「クエリプランと実行ワークフロー」を参照してください。

次の図は、前述のクエリと関連するクエリプランを表しています。これは、クエリオペレーションが、Amazon Redshift が計算ノードスライスのコンパイル済みコードを生成するために使用するステップにどのようにマップされるかを表示します。各クエリプラン操作は、セグメント内の複数のステップにマッピングされます。場合によっては、ストリーム内の複数のセグメントにマッピングされます。



この図では、クエリオプティマイザーは次のようにクエリプランを実行します。

1. `Stream 0` では、クエリは、`Segment 0`テーブルをスキャンする順次スキャン操作で `events` を実行します。クエリは、結合内の内部テーブルのハッシュテーブルを作成するハッシュ操作を使用して `Segment 1` に進みます。
2. `Stream 1` では、クエリは、`Segment 2`テーブルをスキャンする順次スキャン操作で `sales` を実行します。結合列が分散キーでもソートキーでもないテーブルを結合するために、ハッシュ結合を使用して `Segment 2` を続行します。結果を集計するハッシュ集計を使用して、`Segment 2`を再度続行します。次に、クエリではハッシュ集計操作を使用して `Segment 3` を実行し、未ソートのグループ化された集計関数および、ORDER BY 句およびその他ソート操作を評価するソート操作を実行します。
3. `Stream 2` では、クエリが`Segment 4` と `Segment 5` でネットワーク操作を実行し、今後の処理のために中間結果をリーダーノードに送信します。

クエリの最後のセグメントはデータを返します。戻り値一式が集約またはソートされている場合、計算ノードはそれぞれ中間結果をリーダーノードに送信します。その後、リーダーノードがデータをマージし、最終結果を要求元のクライアントに送り返すことができます。

EXPLAIN 演算子の詳細については、「EXPLAIN」を参照してください。

クエリパフォーマンスに影響を与える要因

いくつかの要因がクエリパフォーマンスに影響を与える可能性があります。データ、クラスター、データベース操作の次の側面はすべて、クエリ処理の速度に影響を与えます。

- ノード、プロセッサ、スライスの数 – コンピューティングノードはスライスに分割されています。ノードが多いということは、プロセッサとスライスが多いことを意味するため、クエリの各部分をスライス間で同時実行することによりプロエスをすばやく処理することが可能になります。ただし、ノードが多いとコストも上昇するため、システムに適したコストとパフォーマンスのバランスを見つける必要があります。Amazon Redshift クラスターアーキテクチャの詳細については、データウェアハウスシステムのアーキテクチャを参照してください。
- ノードタイプ — Amazon Redshift クラスターでは、いくつかのノードタイプのうち 1 つを使用できます。各ノードタイプは様々なサイズを提供し、クラスターを適切に拡張することができるように制限します。ノードサイズによって、クラスター内の各ノードのストレージ容量、メモリ、CPU、および料金が決まります。ノードタイプの詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift でのクラスター管理の概要」を参照してください。
- データディストリビューション – Amazon Redshift は、テーブルのディストリビューションスタイルに応じて、テーブルデータをコンピューティングノードに保存します。クエリを実行すると、

クエリオプティマイザが結合と集計を実行するための必要に応じて、データをコンピューティングノードに再分散します。テーブルに適した分散スタイルを選択すると、結合を実行する前にデータを必要な場所に配置しておくことによって、再分散ステップの影響を最小限に抑えることができます。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

- データのソート順 – Amazon Redshift は、テーブルのソートキーに応じたソート順で、テーブルデータをディスクに保存します。クエリオプティマイザとクエリプロセッサは、データの保存場所に関する情報を使用して、スキャンする必要があるブロックの数を減らすため、クエリの速度が向上します。詳細については、「ソートキー」を参照してください。
- データセットサイズ – クラスター内のデータのボリュームが大きくなると、スキャンおよび再分散する必要がある行が増えるため、クエリのパフォーマンスが低下する可能性があります。データのバキューム処理とアーカイブを定期的に実行し、クエリデータセットを制限する述語を使用することによって、この影響を緩和できます。
- 同時オペレーション – 複数のオペレーションを一度に実行すると、クエリパフォーマンスに影響が及ぶ可能性があります。操作が実行されるたびに使用可能なクエリーキューの 1 つ以上のスロットが取得され、それらのスロットに関連付けられたメモリが使用されます。他の操作が実行されている場合、十分なクエリキュースロットを使用できない可能性があります。この場合、クエリは処理を開始する前にスロットが空くのを待つ必要があります。クエリキューの作成と設定の詳細については、「ワークロード管理」を参照してください。
- クエリ構造 – クエリの書き込み方法は、そのパフォーマンスに影響を与えます。できる限り少ないデータを処理して返すクエリを記述すると、ニーズを満たすことができます。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。
- コードコンパイル – Amazon Redshift は、各クエリ実行プランのコードを生成してコンパイルします。

  コンパイル済みコードは、インタプリタを使用してオーバーヘッドを削除するため、高速で実行されます。通常、コードが初めて生成およびコンパイルされるときは、ある程度のオーバーヘッドコストが生じます。その結果、最初に実行したときのクエリのパフォーマンスは、誤解を招く場合があります。1 回限りのクエリを実行するときは、オーバーヘッドコストは特に顕著になります。クエリのパフォーマンスを判断するには、必ず 2 回目にクエリを実行します。Amazon Redshift は、サーバーレスコンパイルサービスを使用して、Amazon Redshift クラスターのコンピューティングリソースを超えてクエリコンパイルをスケーリングします。コンパイルされたコードセグメントは、クラスターでローカルにキャッシュされるだけでなく、事実上無制限のキャッシュに保存されます。このキャッシュは、クラスターの再起動後も保持されます。同じクエリをそれ以降に実行すると、コンパイルフェーズをスキップできるため、高速になります。

キャッシュは Amazon Redshift のバージョン間で互換性がないため、バージョンのアップグレード後にクエリが実行されると、コンパイルキャッシュがフラッシュされ、コードが再コンパイルされます。クエリに厳密な SLA がある場合は、クラスターテーブルのデータをスキャンするクエリセグメントを事前に実行することをお勧めします。これにより、Amazon Redshift でベーステーブルデータがキャッシュされるため、バージョンアップグレード後のクエリの計画時間を短縮できます。スケーラブルなコンパイルサービスを使用することで、Amazon Redshift はコードを並行してコンパイルし、常に高速のパフォーマンスを実現できます。ワークロードの高速化の大きさは、クエリの複雑さと同時実行性によって異なります。

クエリの分析と改善

Amazon Redshift データウェアハウスから情報を取得するには、非常に大量のデータに対して複雑なクエリを実行する必要があります。この処理には時間がかかる場合があります。クエリ処理を可能な限り高速に実行するため、潜在的なパフォーマンスの問題を特定するのに使用できるいくつかのツールがあります。

トピック

- [クエリ分析ワークフロー]
- [クエリアラートの確認]
- [クエリプランの分析]
- [クエリの概要の分析]
- [クエリパフォーマンスの向上]
- [クエリ調整用の診断クエリ]

クエリ分析ワークフロー

クエリに予想より時間がかかる場合、以下の手順を使用し、クエリのパフォーマンスに悪影響を与える可能性のある問題を特定して修正します。システムのどのクエリがパフォーマンスの調整の恩恵を受けるかわからない場合、まず「優先して調整が必要なクエリの特定」で診断クエリを実行します。

1. テーブルがベストプラクティスに従って設計されていることを確認します。詳細については、「Amazon Redshift テーブル設計のベストプラクティス」を参照してください。
2. テーブルの不必要なデータを削除またはアーカイブできるかどうかを確認します。例えば、クエリが常に過去 6 か月分のデータをターゲットとするが、テーブルには過去 18 か月分のデータがあ

るとします。この場合、古いデータを削除またはアーカイブして、スキャンおよび分散が必要なレコード数を削減することができます。

3. クエリ内のテーブルで VACUUM コマンドを実行し、スペースを回復して行を再ソートします。未ソート領域が大きく、クエリが結合または述語でソートキーを使用する場合、VACUUM を実行すると役立ちます。
4. クエリ内のテーブルで ANALYZE コマンドを実行し、統計が最新であることを確認します。クエリ内のいずれかのテーブルのサイズが最近大きく変更された場合、ANALYZE を実行すると役立ちます。ANALYZE コマンドを全体に実行すると時間がかかりすぎる場合、1 つの列に ANALYZE を実行して処理時間を短縮してください。この方法でもテーブルのサイズ統計が更新されます。テーブルサイズは、クエリプランにおいて重要な要素です。
5. クエリがコンパイルおよびキャッシュされるように、クエリがクライアントタイプごとに 1 回実行されたことを確認します (クライアントが使用する接続プロトコルのタイプに基づいて)。この方法により、クエリの後続の実行時間が短縮されます。詳細については、「クエリパフォーマンスに影響を与える要因」を参照してください。
6. STL_ALERT_EVENT_LOG テーブルを確認し、クエリの潜在的な問題を特定して修正します。詳細については、「クエリアラートの確認」を参照してください。
7. EXPLAIN コマンドを実行してクエリプランを取得し、そのクエリプランを使用してクエリを最適化します。詳細については、「クエリプランの分析」を参照してください。
8. SVL_QUERY_SUMMARY ビューと SVL_QUERY_REPORT ビューを使用して概要情報を取得し、その情報を使用してクエリを最適化します。詳細については、「クエリの概要の分析」を参照してください。

場合によっては、高速に実行されるべきクエリが、実行時間の長い別のクエリが終了するまで待機せざるを得ないこともあります。その場合、クエリ自体を改善する方法はないかもしれませんが、さまざまなクエリタイプのクエリキューを作成して使用することで、システム全体のパフォーマンスを向上できます。クエリのキュー待機時間を調べるには、「クエリのキュー待機時間の確認」を参照してください。クエリキューの設定の詳細については、「ワークロード管理」を参照してください。

クエリアラートの確認

STL_ALERT_EVENT_LOG システムテーブルを使用し、クエリの潜在的なパフォーマンスの問題を特定して解決するには、以下の手順を実行します。

1. 次の操作を実行してクエリ ID を調べます。

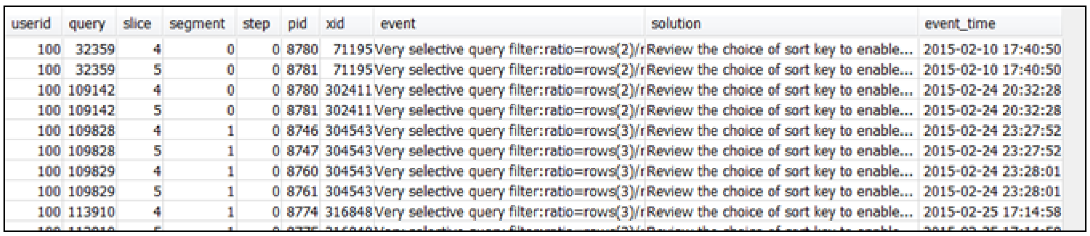
```
select query, elapsed, substring
```

```
from svl_qlog
order by query
desc limit 5;
```

substring フィールドで切り捨てられたクエリテキストを調べ、選択する query 値を特定します。クエリを複数回実行した場合、query値が小さい行の elapsed 値を使用します。これは、コンパイル済みバージョンの行です。多くのクエリを実行している場合、クエリが確実に含められるように、LIMIT 句により使用される値を大きくすることができます。

2. クエリの STL_ALERT_EVENT_LOG から行を選択します。

```
Select * from stl_alert_event_log where query = MyQueryID;
```



userid	query	slice	segment	step	pid	xid	event	solution	event_time
100	32359	4	0	0	8780	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	32359	5	0	0	8781	71195	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-10 17:40:50
100	109142	4	0	0	8780	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109142	5	0	0	8781	302411	Very selective query filter:ratio=rows(2)/r	Review the choice of sort key to enable...	2015-02-24 20:32:28
100	109828	4	1	0	8746	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109828	5	1	0	8747	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:27:52
100	109829	4	1	0	8760	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	109829	5	1	0	8761	304543	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-24 23:28:01
100	113910	4	1	0	8774	316848	Very selective query filter:ratio=rows(3)/r	Review the choice of sort key to enable...	2015-02-25 17:14:58

3. クエリの結果を評価します。次の表を使用して、特定した問題の考えられる解決策を見つけます。

> **Note**
>
> すべてのクエリに STL_ALERT_EVENT_LOG の行があるとは限りません。問題が特定されたクエリのみです。

問題	イベント値	ソリューション値	推奨される解決策
クエリ内のテーブルの統計がないか古い。	クエリプランナー統計がない	ANALYZE コマンドを実行する	「テーブル統計がないか古い」を参照してください。

問題	イベント値	ソリューション値	推奨される解決策
クエリプランにネステッドループ結合 (最適性が最も低い結合) がある。	クエリプラン内にネストッドループ結合	結合述語を確認してデカルト積を回避する	「Nested Loop」を参照してください。
削除済みだがバキューム未処理としてマークされた比較的多数の行、または挿入済みだがコミットされていない比較的多数の行がスキャンによってスキップされた。	多数の削除された行がスキャンされた	VACUUM コマンドを実行して削除されたスペースを回復する	「非実体行または未コミット行」を参照してください。
ハッシュ結合または集計で 100 万を超える行が再分散された。	多数の行がネットワーク全体に分散された : 集計を処理するために RowCount 行が分散された	分散キーの選択を確認して、結合または集計をコロケーションする	「十分最適でないデータ分散」を参照してください。
ハッシュ結合で 100 万を超える行がブロードキャストされた。	多数の行がネットワーク全体にブロードキャストされた	分散キーの選択を確認して結合をコロケーションし、分散されたテーブルの使用を検討する	「十分最適でないデータ分散」を参照してください。
内部テーブル全体が単一ノードに再分散されたために直列実行を強制する、DS_DIST_ALL_INNER 再分散スタイルがクエリプランで指定されました。	クエリプラン内のハッシュ結合の DS_DIST_ALL_INNER	分散戦略の選択を確認し、外部テーブルではなく内部テーブルを分散する	「十分最適でないデータ分散」を参照してください。

クエリプランの分析

EXPLAIN コマンドを実行して、クエリプランを取得します。

クエリプランを分析する前に、クエリプランを読む方法の知識が必要です。クエリプランを読む方法がわからない場合、次に進む前に「クエリプランの作成と解釈」を参照することをお勧めします。

クエリプランにより提供されるデータを分析するには、以下の手順を実行します。

1. 最もコストが高い手順を特定します。残りの手順を進める際、それらの手順の最適化に集中してください。
2. 結合の種類を参照します。
   - ネステッドループ: この結合は通常、結合条件が省略されたために発生します。推奨される解決策については、「Nested Loop」を参照してください。
   - ハッシュおよびハッシュ結合: ハッシュ結合は、結合列が分散キーでもソートキーでもないテーブルを結合するときに使用されます。推奨される解決策については、「ハッシュ結合」を参照してください。
   - マージ結合: 変更は必要ありません。
3. どのテーブルが内部結合に使用され、どのテーブルが外部結合に使用されるのかに注目してください。クエリエンジンは通常、内部結合に小さいテーブルを選択し、外部結合に大きいテーブルを選択します。統計が古いと思われる場合、このような選択は行われません。推奨される解決策については、「テーブル統計がないか古い」を参照してください。
4. 高コストのソート操作があるかどうかを調べます。そのような操作がある場合、「未ソート行または正しくソートされていない行」で推奨される解決策を参照してください。
5. 高コストの操作が存在する以下のブロードキャスト演算子を探します。
   - DS_BCAST_INNER: テーブルがすべてのコンピューティングノードにブロードキャストされていることを示します。これは、小さなテーブルなら問題ありませんが、大きなテーブルには適しません。
   - DS_DIST_ALL_INNER: すべてのワークロードが 1 つのスライスにあることを示します。
   - DS_DIST_BOTH: 重度の再分散を示します。

   これらの状況に推奨される解決策については、「十分最適でないデータ分散」を参照してください。

クエリの概要の分析

EXPLAIN により生成されるクエリプランより詳細な実行ステップと統計を取得するには、SVL_QUERY_SUMMARYシステムビューと SVL_QUERY_REPORT システムビューを使用します。

SVL_QUERY_SUMMARY は、クエリの統計をストリームで提供します。提供される情報を使用して、コストが高いステップ、実行時間が長いステップ、ディスクへの書き込みを行うステップに関する問題を特定することができます。

SVL_QUERY_REPORT システムビューでは、SVL_QUERY_SUMMARY と似た情報を参照できますが、システムごとではなくコンピューティングノードごとのみです。クラスター全体における不均等なデータ分散を検出するためのスライスレベル情報 (データ分散スキューとも呼ばれます) を使用できます。データ分散が不均等になっていると、一部のノードの処理量が他のノードより多くなり、クエリパフォーマンスに悪影響を与えます。

トピック

SVL_QUERY_SUMMARY ビューの使用

SVL_QUERY_SUMMARY を使用して、クエリの概要情報をストリームで分析するには、以下を実行します。

1. 次のクエリを実行してクエリ ID を調べます。

```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

   substring フィールドの切り捨てられたクエリテキストを調べ、どの query 値がクエリを表しているかを確認します。クエリを複数回実行した場合、query値が小さい行の elapsed 値を使用します。これは、コンパイル済みバージョンの行です。多くのクエリを実行している場合、クエリが確実に含められるように、LIMIT 句により使用される値を大きくすることができます。

2. クエリの SVL_QUERY_SUMMARY から行を選択します。結果をストリーム、セグメント、ステップ順に並べ替えます。

```
select * from svl_query_summary where query = MyQueryID order by stm, seg, step;
```

   結果の例は次のとおりです。

Result 1 | Messages

userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem	is_rrscan	is_delayed_scan	rows_pre_filter
1	249059	0	0	0	58	27	4	192			scan tbl=246 name=Internal Worktable	f	0	f	f	0
1	249059	0	0	1	58	27	4	0			project	f	0	f	f	0
1	249059	0	0	2	58	27	4	64			save tbl=249	f	481296384	f	f	0
1	249059	1	1	0	20	20	1	48			scan tbl=250 name=Internal Worktable	f	0	f	f	0
1	249059	1	1	1	20	20	1	0			dist	f	0	f	f	0
1	249059	1	2	0	2275	1350	1	48			scan tbl=19221 name=Internal Worktable	f	0	f	f	0
1	249059	1	2	1	2275	1350	1	0			project	f	0	f	f	0
1	249059	1	2	2	2275	1350	1	16			save tbl=249	f	475004928	f	f	0
1	249059	2	3	0	1640	792	5	80			scan tbl=249 name=Internal Worktable	f	0	f	f	0
1	249059	2	3	1	1640	792	5	80			sort tbl=248	f	468713472	f	f	0
1	249059	3	4	0	26	9	5	80			scan tbl=248 name=Internal Worktable	f	0	f	f	0
1	249059	3	4	1	26	9	5	0			return	f	0	f	f	0
1	249059	3	5	0	49	49	0	0			merge	f	0	f	f	0
1	249059	3	5	1	49	49	5	0			project	f	0	f	f	0
1	249059	3	5	2	49	49	0	0			return	f	0	f	f	0

3. 「クエリの概要へのクエリプランのマッピング」の情報を使用して、クエリプラン内の操作にステップをマッピングします。これらは、列およびバイト (クエリプランの行 x 幅) の値とほぼ同じになります。大きく異なる場合、「テーブル統計がないか古い」で推奨される解決策を参照してください。

4. `is_diskbased` フィールドに、ステップの `t` 値がある (true) かどうかを調べます。ハッシュ、集計、およびソートは、システムでクエリ処理に十分なメモリが割り当てられていない場合に、ディスクにデータを書き込む可能性が高い演算子です。

   `is_diskbased` が true の場合、「クエリに割り当てられてメモリが不十分」で推奨される解決策を参照してください。

5. `label` フィールドの値を確認し、ステップのいずれかの場所に AGG-DIST-AGG シーケンスがあるかどうかを調べます。ある場合、コストの高い 2 ステップの集約であることを示しています。これを修正するには、GROUP BY 句を変更して分散キー (複数ある場合は 1 番目のキー) を使用します。

6. 各セグメントの `maxtime` 値を確認します (セグメントのすべてのステップ間で同じです)。`maxtime` 値が最も大きいセグメントを特定し、このセグメント内のステップで次の演算子を確認します。

> **Note**
>
> `maxtime` 値が大きくても、セグメントに問題があるとは限りません。値が大きくても、セグメントの処理に時間がかからないことがあります。ストリーム内のすべてのセグメントは同時に開始します。ただし、ダウンストリームセグメントによっては、アップストリームセグメントからデータを取得するまで実行できません。そのセグメントの `maxtime` 値には待機時間と処理時間の両方が含まれるため、この効果により長時間かかるように見えることがあります。

- BCAST または DIST: これらの場合、maxtime値が大きいと多数の行が再分散される可能性があります。推奨される解決策については、「十分最適でないデータ分散」を参照してください。
- HJOIN (ハッシュ結合): 問題のステップの rows フィールド内の値が、クエリ内の最後の RETURN ステップの rows 値と比較して非常に大きい場合、「ハッシュ結合」で推奨される解決策を参照してください。
- SCAN/SORT: 結合ステップの直前にあるステップの SCAN、SORT、SCAN、MERGE シーケンスを探します。このパターンは、未ソートのデータがスキャン、ソートされた後、テーブルのソート済み領域とマージされます。

  SCAN ステップの行の値が、クエリ内の最後の RETURN ステップにある行の値と比較して非常に大きいかどうかを確認します。このパターンは、後で破棄される行を実行エンジンがスキャンしていることを示します (非効率的です)。推奨される解決策については、「述語の制限が不十分」を参照してください。

  SCAN ステップの maxtime 値が大きい場合、「十分最適でない WHERE 句」で推奨される解決策を参照します。

  SORT ステップの rows 値がゼロ以外の場合、「未ソート行または正しくソートされていない行」で推奨される解決策を参照してください。

7. 最後の RETURN ステップの前にある 5～10 個のステップの rows 値と bytes 値を確認し、クライアントに返されるデータの量を調べます。このプロセスには少し工夫が必要です。

   例えば、次のクエリの概要では、3 番目の PROJECT ステップにより bytes 値ではなく rows 値が提供されています。先行するステップで同じ rows 値を持つステップを探すと、行とバイトの両方の情報を提供する SCAN ステップが見つかります。

   結果の例を次に示します。

userid	query	stm	seg	step	maxtime	avgtime	rows	bytes	rate_row	rate_byte	label	is_diskbased	workmem
1	187435	2	5	2	14307	12797	0	0			hash tbl=256	f	46871347
1	187435	3	6	0	531	308	387	229104			scan tbl=242 name=Internal Worktable	f	
1	187435	3	6	1	531	308	387	0			project	f	
1	187435	3	6	2	531	308	387	222912			save tbl=245	f	38063308
1	187435	4	7	0	390	390	0	0			scan tbl=238 name=Internal Worktable	f	
1	187435	4	7	1	390	390	0	0			dist	f	
1	187435	4	8	0	1218	1066	0	0			scan tbl=134954 name=Internal Worktable	f	
1	187435	4	8	1	1218	1066	0	0			project	f	
1	187435	4	8	2	1218	1066	0	0			save tbl=245	f	37434163
1	187435	5	9	0	171	83	387	222912			scan tbl=245 name=Internal Worktable	f	
1	187435	5	9	1	171	83	387	60120			dist	f	
1	187435	5	10	0	3579	3383	387	222912			scan tbl=134955 name=Internal Worktable	f	
1	187435	5	10	1	3579	3383	387	0			project	f	
1	187435	5	10	2	3579	3383	0	0			hjoin tbl=256	f	
1	187435	5	10	3	3579	3383	0	0			project	f	
1	187435	5	10	4	3579	3383	0	0			sort tbl=259	f	36805017
1	187435	6	11	0	10	7	0	0			scan tbl=259 name=Internal Worktable	f	
1	187435	6	11	1	10	7	0	0			return	f	
1	187435	6	12	0	9	9	0	0			merge	f	
1	187435	6	12	1	9	9	0	0			project	f	
1	187435	6	12	2	9	9	0	0			return	f	

非常に大量のデータを返す場合、「非常に大きな結果セット」で推奨される解決策を参照してください。

8. 他のステップと比較して、いずれかのステップの `bytes` 値が `rows` 値より大きいかどうかを確認します。このパターンは、多くの列を選択していることを示す場合があります。推奨される解決策については、「大きい SELECT リスト」を参照してください。

SVL_QUERY_REPORT ビューの使用

SVL_QUERY_REPORT を使用して、クエリの概要情報をスライスで分析するには、以下を実行します。

1. 次の操作を実行してクエリ ID を調べます。

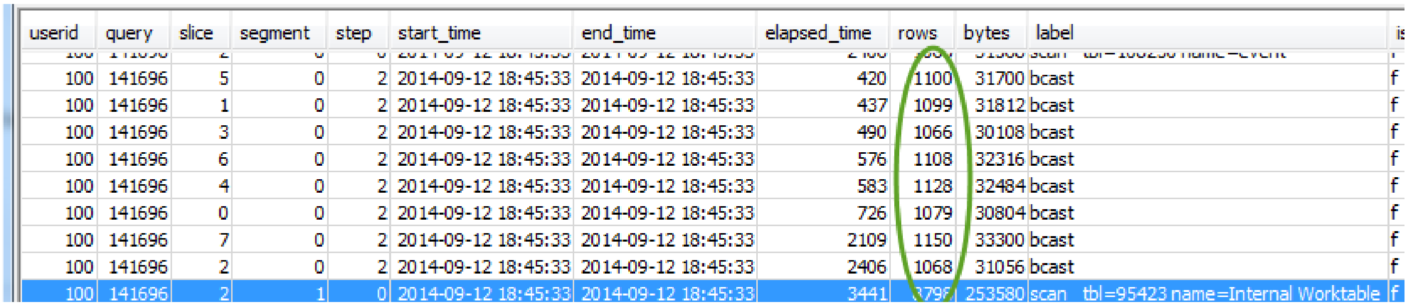
```
select query, elapsed, substring
from svl_qlog
order by query
desc limit 5;
```

`substring` フィールドの切り捨てられたクエリテキストを調べ、どの `query` 値がクエリを表しているかを確認します。クエリを複数回実行した場合、`query`値が小さい行の `elapsed` 値を使用します。これは、コンパイル済みバージョンの行です。多くのクエリを実行している場合、クエリが確実に含められるように、LIMIT 句により使用される値を大きくすることができます。

2. クエリの SVL_QUERY_REPORT から行を選択します。segment、step、elapsed_time、rows により結果を並べ替えます。

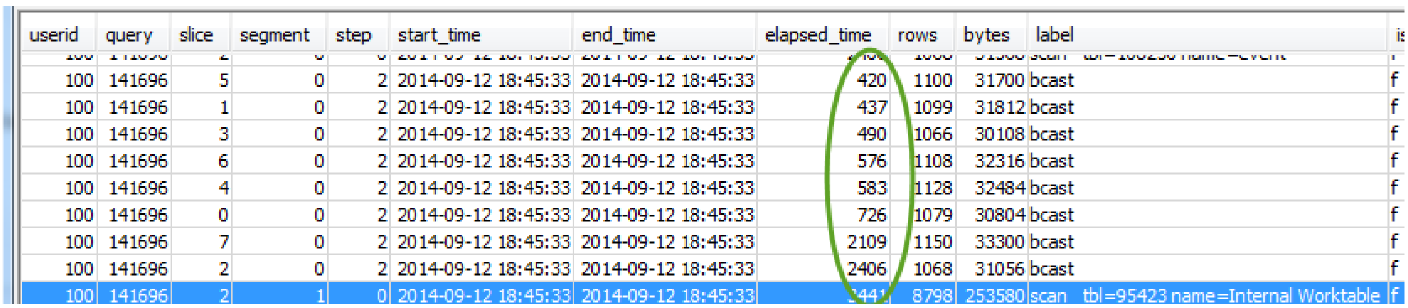
```
select * from svl_query_report where query = MyQueryID order by segment, step,
 elapsed_time, rows;
```

3. 各ステップで、すべてのスライスがほぼ同じ数の行を処理していることを確認します。



userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label	is
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast	f
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast	f
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast	f
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast	f
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast	f
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast	f
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast	f
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast	f
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	8798	253580	scan tbl=95423 name=Internal Worktable	f

さらに、すべてのスライスにほぼ同じ時間がかかっていることを確認します。



userid	query	slice	segment	step	start_time	end_time	elapsed_time	rows	bytes	label	is
100	141696	5	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	420	1100	31700	bcast	f
100	141696	1	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	437	1099	31812	bcast	f
100	141696	3	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	490	1066	30108	bcast	f
100	141696	6	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	576	1108	32316	bcast	f
100	141696	4	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	583	1128	32484	bcast	f
100	141696	0	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	726	1079	30804	bcast	f
100	141696	7	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2109	1150	33300	bcast	f
100	141696	2	0	2	2014-09-12 18:45:33	2014-09-12 18:45:33	2406	1068	31056	bcast	f
100	141696	2	1	0	2014-09-12 18:45:33	2014-09-12 18:45:33	3441	8798	253580	scan tbl=95423 name=Internal Worktable	f

これらの値が大きく異なる場合、この特定のクエリの分散スタイルが十分最適でないことによるデータ分散スキューを示している可能性があります。推奨される解決策については、「十分最適でないデータ分散」を参照してください。

クエリの概要へのクエリプランのマッピング

クエリの概要を分析する際は、クエリプランからクエリの概要のステップ (ラベルフィールドの値により識別) に操作をマッピングすることで、さらに詳細な情報を得ることができます。次の表では、クエリプランの操作をクエリの概要のステップにマッピングしています。

クエリプランの操作	ラベルフィールドの値	説明
Aggregate HashAggregate GroupAggregate	AGGR	集計関数および GROUP 条件を評価します。

クエリプランの操作	ラベルフィールドの値	説明
DS_BCAST_INNER	BCAST (ブロードキャスト)	テーブル全体または一部の行セット (テーブルのフィルタリングされた行セットなど) をすべてのノードにブロードキャストします。
クエリプランに表示されない	DELETE	テーブルから行を削除します。
DS_DIST_NONE DS_DIST_ALL_NONE DS_DIST_INNER DS_DIST_ALL_INNER DS_DIST_ALL_BOTH	DIST (分散)	並列結合または他の並列処理を行うため、行からノードに分散します。
HASH	HASH	ハッシュ結合で使用するためにハッシュテーブルを構築します。
Hash Join	HJOIN (ハッシュ結合)	2 つのテーブル、または中間結果セットのハッシュ結合を実行します。
クエリプランに表示されない	INSERT	行をテーブルに挿入します。
制限	制限	LIMIT 句を結果セットに適用します。
Merge	MERGE	並列ソートまたは結合操作から派生した行をマージします。

クエリプランの操作	ラベルフィールドの値	説明
Merge Join	MJOIN (マージ結合)	2 つのテーブル、または中間結果セットのマージ結合を実行します。
Nested Loop	NLOOP (ネステッドループ)	2 つのテーブル、または中間結果セットのネステッドループ結合を実行します。
クエリプランに表示されない	PARSE	ロードするために、文字列をバイナリ値に解析します。
プロジェクト	PROJECT	式を評価します。
ネットワーク	RETURN	行をリーダーまたはクライアントに返します。
クエリプランに表示されない	SAVE	次の処理ステップで使用するために行をマテリアライズします。
Seq Scan	SCAN	テーブルまたは中間結果セットをスキャンします。
Sort	SORT	後続の他の操作 (結合や集計など) の必要に応じて、ORDER BY 句を満たすために、行または中間結果セットをソートします。
Unique	UNIQUE	SELECT DISTINCT 句を適用するか、他の操作の必要に応じて重複を排除します。
Window	WINDOW	集計およびランキングウィンドウ関数を計算します。

クエリパフォーマンスの向上

Amazon Redshift のクエリパフォーマンスに影響を与える一般的な問題と、それらの問題を診断して解決する手順を以下に示します。

トピック

テーブル統計がないか古い

テーブル統計がないか古い場合、次の状況が発生することがあります。

- EXPLAIN コマンドの結果として警告メッセージが表示される。
- STL_ALERT_EVENT_LOG に統計がないことを示すアラートイベントが記録される。詳細については、「クエリアラートの確認」を参照してください。

この問題を修正するには、ANALYZEを実行します。

Nested Loop

ネステッドループがある場合、STL_ALERT_EVENT_LOG にネステッドループのアラートイベントが記録されることがあります。ネステッドループにあるクエリの特定 でクエリを実行することで、このイベントタイプを識別することもできます。詳細については、「クエリアラートの確認」を参照してください。

この問題を修正するには、クエリでクロス結合を確認し、可能であれば削除します。クロス結合は、2 つのテーブルのデカルト積を算出する結合条件のない結合です。これらは通常、可能な結合タイプの中で最も遅いネステッドループ結合として実行されます。

ハッシュ結合

ハッシュ結合が存在する場合、次の状況が発生する可能性があります。

- クエリプランにハッシュおよびハッシュ結合操作がある。詳細については、「クエリプランの分析」を参照してください。
- SVL_QUERY_SUMMARY の maxtime 値が最も大きいセグメントに HJOIN ステップがある。詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

この問題を修正するには、いくつかの方法を実行することができます。

- 可能であれば、マージ結合を使用するようにクエリを書き換えます。これは、分散キーおよびソートキーの両方である結合列を指定することで行うことができます。
- SVL_QUERY_SUMMARY 内の HJOIN ステップにある行の値が、クエリ内の最後の RETURN ステップにある行と比較して非常に大きい場合、クエリを書き換えて一意の列で結合することができるかどうかを確認します。クエリが一意の列 (プライマリキーなど) で結合されない場合、結合に関係する行数が増加します。

非実体行または未コミット行

非実体行または未コミット行が存在する場合、非実体行が多すぎることを示すアラートイベントが STL_ALERT_EVENT_LOG に記録されることがあります。詳細については、「クエリアラートの確認」を参照してください。

この問題を修正するには、いくつかの方法を実行することができます。

- クエリテーブルのアクティブなロードオペレーションについては、Amazon Redshift コンソールの [Loads (ロード)] タブを確認してください。アクティブなロード操作がある場合、アクションを実行する前にそれらの操作が完了するまで待ちます。
- アクティブなロード操作がない場合、クエリテーブルで VACUUM を実行して、削除済みの行を除去します。

未ソート行または正しくソートされていない行

未ソート行または正しくソートされていない行が存在する場合、STL_ALERT_EVENT_LOG にかなり限定的なアラートイベントが記録されることがあります。詳細については、「クエリアラートの確認」を参照してください。

データスキューまたは未ソート行のあるテーブルの特定 でクエリを実行することにより、クエリ内のいずれかのテーブルに未ソート領域が大量にあるかどうかを確認することもできます。

この問題を修正するには、いくつかの方法を実行することができます。

- クエリテーブルで VACUUM を実行し、行を再ソートします。
- クエリテーブルでソートキーを確認し、改善できる点がないかどうかを調べます。変更を加える前に、必ずこのクエリのパフォーマンスと他の重要なクエリやシステム全体のパフォーマンスを比較検討してください。詳細については、「ソートキー」を参照してください。

十分最適でないデータ分散

データ分散が十分に最適でない場合、次の状況が発生する可能性があります。

- 直列実行、大量のブロードキャスト、または大量の分散に関するアラートイベントが STL_ALERT_EVENT_LOG に記録される。詳細については、「クエリアラートの確認」を参照してください。
- 各スライスがあるステップについて処理している行の数が大きく異なる。詳細については、「SVL_QUERY_REPORT ビューの使用」を参照してください。
- 各スライスがあるステップにかけている時間が大きく異なる。詳細については、「SVL_QUERY_REPORT ビューの使用」を参照してください。

上記のどれもあてはまらない場合、データスキューまたは未ソート行のあるテーブルの特定でクエリを実行することにより、クエリ内のいずれかのテーブルにデータスキューがないかどうかを確認することもできます。

この問題を修正するには、クエリ内のテーブルの分散スタイルを確認して、改善できる点がないかを調べます。変更を加える前に、必ずこのクエリのパフォーマンスと他の重要なクエリやシステム全体のパフォーマンスを比較検討してください。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

クエリに割り当てられてメモリが不十分

クエリに割り当てられたメモリが不十分な場合、`is_diskbased`値が true のステップが SVL_QUERY_SUMMARY に存在する可能性があります。詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

この問題を修正するには、クエリが使用するクエリスロットの数を一時的に増やして、クエリに多くのメモリを割り当てます。ワークロード管理 (WLM) は、キューに設定された同時実行レベルと同等のスロットをクエリキューに確保します。例えば、同時実行レベルが 5 のキューには 5 つのスロットがあります。キューに割り当てられたメモリは、各スロットに均等に割り当てられます。1 つのクエリに複数のスロットを割り当てると、そのクエリはすべてのスロットのメモリにアクセスできます。クエリのスロットを一時的に増やす方法の詳細については、「wlm_query_slot_count」を参照してください。

十分最適でない WHERE 句

WHERE 句により実行されるテーブルスキャンが多すぎる場合、SCAN ステップのセグメントの SVL_QUERY_SUMMARY 内の `maxtime` 値が最も大きくなる可能性があります。詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

この問題を修正するには、最も大きいテーブルのプライマリソート列に基づいて WHERE 句をクエリに追加します。この方法により、スキャン時間を最小限に抑えることができます。詳細については、「Amazon Redshift テーブル設計のベストプラクティス」を参照してください。

述語の制限が不十分

制限が不十分な述語がクエリにある場合、SVL_QUERY_SUMMARY にある `maxtime` 値が最も大きいセグメントの SCAN ステップの `rows` 値が、クエリ内の最後の RETURN ステップにある `rows` 値と比較してかなり大きくなることがあります。詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

この問題を修正するには、クエリに述語を追加するか、既存の述語の制限を強めて出力を絞り込んでみてください。

非常に大きな結果セット

クエリにより非常に大きな結果セットが返される場合、UNLOADを使用して Amazon S3 に結果が書き込まれるようにクエリを書き換えることを検討してください。この方法を実行すると、並列処理を活用することで RETURN ステップのパフォーマンスが向上します。非常に大きな結果セットを確認する方法の詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

大きい SELECT リスト

クエリに非常に大きい SELECT リストがある場合、SVL_QUERY_SUMMARY 内のいずれかのステップの `bytes` 値が `rows` 値より大きくなる (他のステップと比較して) ことがあります。`bytes` 値が大きいことは、多くの列を選択していることを示す場合があります。詳細については、「SVL_QUERY_SUMMARY ビューの使用」を参照してください。

この問題を修正するには、選択している列を確認し、削除ができるかどうかを調べます。

クエリ調整用の診断クエリ

クエリや基礎となるテーブルの、クエリパフォーマンスに影響を与える可能性がある問題を特定するには、次のクエリを使用します。これらのクエリは、「クエリの分析と改善」で説明されているクエリ調整プロセスと同時に使用することをお勧めします。

> **Note**
>
> これらのクエリは Amazon Redshift でプロビジョニングされたクラスター用です。これらのクエリは Redshift Serverless ワークグループでは使用できません。

トピック

- 優先して調整が必要なクエリの特定
- データスキューまたは未ソート行のあるテーブルの特定
- ネステッドループにあるクエリの特定
- クエリのキュー待機時間の確認
- テーブルごとのクエリアラートの確認
- 統計がないテーブルの特定

優先して調整が必要なクエリの特定

以下のクエリは、過去 7 日間に実行されたステートメントのうち、最も時間がかかった上位 50 位のステートメントを特定します。この結果を利用することで、異常に時間がかかっているクエリを特定できます。また、頻繁に実行されるクエリ (結果セットに複数回出現するクエリ) を特定できます。これらのクエリは多くの場合、システムパフォーマンスを高めるために優先して調整する必要があります。

このクエリは、特定された各クエリに関連するアラートイベントの数も生成します。これらのアラートは、クエリのパフォーマンスを高めるために使用できる詳細を提供します。詳細については、「クエリアラートの確認」を参照してください。

```
select trim(database) as db, count(query) as n_qry,
max(substring (qrytext,1,80)) as qrytext,
min(run_minutes) as "min" ,
max(run_minutes) as "max",
avg(run_minutes) as "avg", sum(run_minutes) as total,
max(query) as max_query_id,
max(starttime)::date as last_run,
sum(alerts) as alerts, aborted
from (select userid, label, stl_query.query,
trim(database) as database,
trim(querytxt) as qrytext,
md5(trim(querytxt)) as qry_md5,
starttime, endtime,
(datediff(seconds, starttime,endtime)::numeric(12,2))/60 as run_minutes,
alrt.num_events as alerts, aborted
from stl_query
left outer join
(select query, 1 as num_events from stl_alert_event_log group by query ) as alrt
on alrt.query = stl_query.query
where userid <> 1 and starttime >=  dateadd(day, -7, current_date))
group by database, label, qry_md5, aborted
order by total desc limit 50;
```

データスキューまたは未ソート行のあるテーブルの特定

次のクエリは、データ分散が均等でない (データスキュー) テーブルや、未ソート行の割合が多いテーブルを特定します。

skew 値が小さい場合、テーブルデータが適切に分散されていることを示します。テーブルの skew 値が 4.00 以上の場合、データ分散スタイルを変更することを検討してください。詳細については、「十分最適でないデータ分散」を参照してください。

テーブルの pct_unsorted 値が 20 パーセントを超える場合、VACUUMコマンドの実行を検討します。詳細については、「未ソート行または正しくソートされていない行」を参照してください。

各テーブルの mbytes 値と pct_of_total 値も見直してください。これらの列は、テーブルのサイズと、raw ディスクスペースに対してテーブルが消費しているスペースの割合を示します。raw ディ

スク容量には、Amazon Redshift が内部用に予約するスペースの分も含まれます。このため、ユーザーが利用できるディスク容量として表示される名目上のディスク容量より大きな数字になります。この情報をもとに、最も大きいテーブルの 2.5 倍以上の空きディスク容量があることを確認します。このスペースがあれば、システムは複雑なクエリの処理時に中間結果をディスクに書き込むことができます。

```
select trim(pgn.nspname) as schema,
trim(a.name) as table, id as tableid,
decode(pgc.reldiststyle,0, 'even',1,det.distkey ,8,'all') as distkey,
 dist_ratio.ratio::decimal(10,4) as skew,
det.head_sort as "sortkey",
det.n_sortkeys as "#sks", b.mbytes,
decode(b.mbytes,0,0,((b.mbytes/part.total::decimal)*100)::decimal(5,2)) as
 pct_of_total,
decode(det.max_enc,0,'n','y') as enc, a.rows,
decode( det.n_sortkeys, 0, null, a.unsorted_rows ) as unsorted_rows ,
decode( det.n_sortkeys, 0, null, decode( a.rows,0,0, (a.unsorted_rows::decimal(32)/
a.rows)*100) )::decimal(5,2) as pct_unsorted
from (select db_id, id, name, sum(rows) as rows,
sum(rows)-sum(sorted_rows) as unsorted_rows
from stv_tbl_perm a
group by db_id, id, name) as a
join pg_class as pgc on pgc.oid = a.id
join pg_namespace as pgn on pgn.oid = pgc.relnamespace
left outer join (select tbl, count(*) as mbytes
from stv_blocklist group by tbl) b on a.id=b.tbl
inner join (select attrelid,
min(case attisdistkey when 't' then attname else null end) as "distkey",
min(case attsortkeyord when 1 then attname  else null end ) as head_sort ,
max(attsortkeyord) as n_sortkeys,
max(attencodingtype) as max_enc
from pg_attribute group by 1) as det
on det.attrelid = a.id
inner join ( select tbl, max(mbytes)::decimal(32)/min(mbytes) as ratio
from (select tbl, trim(name) as name, slice, count(*) as mbytes
from svv_diskusage group by tbl, name, slice )
group by tbl, name ) as dist_ratio on a.id = dist_ratio.tbl
join ( select sum(capacity) as  total
from stv_partitions where part_begin=0 ) as part on 1=1
where mbytes is not null
order by  mbytes desc;
```

ネステッドループにあるクエリの特定

次のクエリは、ネステッドループに関して記録されたアラートイベントを持つクエリを特定します。ネステッドループ状態を修正する方法については、「Nested Loop」を参照してください。

```
select query, trim(querytxt) as SQL, starttime
from stl_query
where query in (
select distinct query
from stl_alert_event_log
where event like 'Nested Loop Join in the query plan%')
order by starttime desc;
```

クエリのキュー待機時間の確認

以下のクエリでは、最近のクエリが、実行される前にクエリキューで空きスロットを待機していた時間の長さが示されます。待機時間が長い傾向が見られる場合、スループットを高めるためにクエリキュー設定を変更できます。詳細については、「手動 WLM を実装する」を参照してください。

```
select trim(database) as DB , w.query,
substring(q.querytxt, 1, 100) as querytxt,  w.queue_start_time,
w.service_class as class, w.slot_count as slots,
w.total_queue_time/1000000 as queue_seconds,
w.total_exec_time/1000000 exec_seconds, (w.total_queue_time+w.total_Exec_time)/1000000
 as total_seconds
from stl_wlm_query w
left join stl_query q on q.query = w.query and q.userid = w.userid
where w.queue_start_Time >= dateadd(day, -7, current_Date)
and w.total_queue_Time > 0  and w.userid >1
and q.starttime >= dateadd(day, -7, current_Date)
order by w.total_queue_time desc, w.queue_start_time desc limit 35;
```

テーブルごとのクエリアラートの確認

次のクエリは、アラートイベントが記録されたテーブルを特定します。また、最も頻繁に発生したアラートの種類も特定します。

特定されたテーブルの行の minutes 値が大きい場合、そのテーブルで、ANALYZEや VACUUM を実行するなどの定期的なメンテナンスが必要かどうかを確認します。

行の count 値が大きいが table が null の場合、関連する event 値の STL_ALERT_EVENT_LOG に対してクエリを実行し、そのアラートが頻繁に派生する理由を調査します。

```
select trim(s.perm_table_name) as table,
(sum(abs(datediff(seconds, s.starttime, s.endtime)))/60)::numeric(24,0) as minutes,
 trim(split_part(l.event,':',1)) as event,  trim(l.solution) as solution,
max(l.query) as sample_query, count(*)
from stl_alert_event_log as l
left join stl_scan as s on s.query = l.query and s.slice = l.slice
and s.segment = l.segment and s.step = l.step
where l.event_time >=  dateadd(day, -7, current_Date)
group by 1,3,4
order by 2 desc,6 desc;
```

統計がないテーブルの特定

次のクエリは、統計がないテーブルに対して実行中のクエリの数を示します。このクエリが任意の行を返す場合、plannode値を調べて影響を受けるテーブルを特定した後、そのテーブルで ANALYZE を実行します。

```
select substring(trim(plannode),1,100) as plannode, count(*)
from stl_explain
where plannode like '%missing statistics%'
group by plannode
order by 2 desc;
```

クエリのトラブルシューティング

このセクションは、Amazon Redshift クエリで発生する可能性のある一般的な問題と重大な問題を特定し、それらの問題に対処するためのクイックリファレンスとして追加しました。

トピック

- 接続できない
- クエリがハングする
- クエリに時間がかかりすぎる
- ロードが失敗する
- ロードに時間がかかりすぎる
- ロードデータが正しくない
- JDBC フェッチサイズパラメータの設定

これらの提案は、トラブルシューティングの最初のステップです。以下のリソースにはより詳細な情報が記載されていますので、合わせて参照してください。

- Amazon Redshift クラスターとデータベースへのアクセス
- 自動テーブル最適化
- Amazon Redshift でのデータのロード
- チュートリアル: Amazon S3 からデータをロードする

接続できない

次の理由により、クエリ接続が失敗する可能性があります。以下のトラブルシューティングアプローチをお勧めします。

クライアントがサーバーに接続できない

SSL またはサーバー証明書を使用している場合、接続の問題をトラブルシューティングしているときにまずこの複雑さを排除します。その後、解決策を見つけたら、もう一度 SSL またはサーバー証明書を追加します。詳細については、「Amazon Redshift 管理ガイド」の「接続のセキュリティオプションを設定する」を参照してください。

接続が拒否される

一般的に、接続の確立に失敗したことを示すエラーメッセージを受け取った場合、クラスターにアクセスするためのアクセス許可に問題があることを意味します。詳細については、「Amazon Redshift 管理ガイド」の「接続が拒否または失敗する」を参照してください。

クエリがハングする

次の理由で、クエリがハングしたり、応答を停止したりすることがあります。以下のトラブルシューティングアプローチをお勧めします。

データベースへの接続が中断された

最大送信単位 (MTU) のサイズを小さくします。MTU サイズにより、ネットワーク接続を介して 1 つのイーサネットフレームで転送できるパケットの最大サイズ (バイト単位) が決まります。詳細につ

いては、「Amazon Redshift 管理ガイド」の「The connection to the database is dropped」(データベースへの接続が中断する) を参照してください

データベースへの接続がタイムアウトした

COPY コマンドなどの長いクエリを実行すると、データベースへのクライアント接続がハングまたはタイムアウトしているように見えます。この場合、Amazon Redshift コンソールにはクエリが完了したと表示されますが、クライアントツール自体はまだクエリを実行しているように見えることがあります。接続がいつ停止したかに応じて、クエリの結果がないか、不完全になる可能性があります。この効果は、中間ネットワークコンポーネントによってアイドル接続が終了すると発生します。詳細については、「Amazon Redshift 管理ガイド」の「Firewall Timeout Issue」(ファイアウォールのタイムアウト問題) を参照してください。

ODBC 使用時にクライアント側のメモリ不足エラーが発生する

クライアントアプリケーションが ODBC 接続を使用し、クエリで作成される結果セットが大きすぎてメモリが足りなくなる場合、カーソルを使用して、結果セットをクライアントアプリケーションに渡すことができます。詳細については、「DECLARE」および「カーソルを使用するときのパフォーマンスに関する考慮事項」を参照してください。

JDBC 使用時にクライアント側のメモリ不足エラーが発生する

JDBC 接続で大規模な結果セットを取得しようとすると、クライアント側のメモリ不足エラーが発生する可能性があります。詳細については、「JDBC フェッチサイズパラメータの設定」を参照してください。

デッドロックがある可能性がある

デッドロックがあると考えられる場合は、以下を試してください。

- STV_LOCKS および STL_TR_CONFLICT システムテーブルで複数のテーブルの更新に起因する競合を見つけます。
- PG_CANCEL_BACKEND 関数を使用して、1 つ以上の競合しているクエリをキャンセルします。
- PG_TERMINATE_BACKEND 関数を使用して、セッションを終了します。これにより、終了したセッションで現在実行されているトランザクションがすべてのロックを解除し、トランザクションがロールバックされます。
- 同時書き込み操作のスケジュールを慎重に設定します。詳細については、「同時書き込み操作を管理する」を参照してください。

クエリに時間がかかりすぎる

次の理由で、クエリに時間がかかりすぎる場合があります。以下のトラブルシューティングアプローチをお勧めします。

テーブルが最適化されていない

並列処理を最大限に活用できるようにテーブルのソートキー、分散スタイル、および圧縮エンコードを設定します。詳細については、自動テーブル最適化を参照してください。

クエリがディスクに書き込みを行っている

クエリが少なくともクエリ実行の一部でディスクに書き込みを行っている可能性があります。詳細については、「クエリパフォーマンスの向上」を参照してください。

クエリが他のクエリの終了を待つ必要がある

クエリキューを作成し、別の種類のクエリを適切なキューに割り当てることで、システムの全体的なパフォーマンスを改善できる可能性があります。詳細については、「ワークロード管理」を参照してください。

クエリが最適化されていない

説明プランを分析して、クエリを書き換えることが可能かどうか、またはデータベースを最適化することが可能かどうかを調べます。詳細については、「クエリプランの作成と解釈」を参照してください。

クエリの実行により多くのメモリが必要である

特定のクエリにより多くのメモリが必要な場合は、wlm_query_slot_countを増やすことによって使用可能なメモリを増やすことができます。

データベースに対して VACUUM コマンドを実行する必要がある

大量の行数を追加、削除、変更した場合、データをソートキー順序でロードしていなければ、VACUUM コマンドを実行します。VACUUM コマンドを実行すると、データが再編成され、ソート順序が維持され、パフォーマンスが復旧されます。詳細については、「テーブルのバキューム処理」を参照してください。

実行時間の長いクエリのトラブルシューティングに役立つその他のリソース

以下は、クエリのチューニングに役立つシステムビューのトピックとその他のドキュメントセクションです。

- STV_INFLIGHT システムビューには、クラスターで実行されているクエリが表示されます。STV_RECENTS と一緒に使用すると、現在実行中のクエリや最近完了したクエリを判別できます。
- SYS_QUERY_HISTORY はトラブルシューティングに役立ちます。DDL クエリと DML クエリが、関連するプロパティと共に表示されます。例えば、`running` や `failed` などの現在のステータス、それぞれの実行にかかった時間、同時実行スケーリングクラスターに対してクエリが実行されたかどうかなどを確認できます。
- STL_QUERYTEXT は、SQL コマンドのクエリテキストを取得します。さらに、STL_QUERYTEXT を STV_INFLIGHT に結合する SVV_QUERY_INFLIGHT には、より多くのクエリメタデータが表示されます。
- トランザクションロックの競合は、クエリパフォーマンスの問題を引き起こす可能性があります。現在テーブルをロックしているトランザクションについては、「SVV_TRANSACTIONS」を参照してください。
- 「優先して調整が必要なクエリの特定」には、最近実行したどのクエリが最も時間がかかったかを判断するのに役立つトラブルシューティングクエリが紹介されています。これにより、改善が必要なクエリに集中して取り組むことができます。
- クエリ管理の詳細を参照し、クエリキューの管理方法を理解したい場合は、「ワークロード管理」にその方法が記載されています。ワークロード管理は高度な機能であるため、通常は自動ワークロード管理を利用することをお勧めします。

ロードが失敗する

次の理由で、データロードが失敗する可能性があります。以下のトラブルシューティングアプローチをお勧めします。

データソースが別の AWS リージョンにある

デフォルトでは、COPY コマンドで指定された Amazon S3 バケットまたは Amazon DynamoDB テーブルは、クラスターと同じ AWS リージョンに置かれている必要があります。データとクラスターが異なるリージョンにある場合、次のようなエラーが発生します。

```
The bucket you are attempting to access must be addressed using the specified endpoint.
```

可能な限り、クラスターとデータソースが同じリージョンに配置されるようにしてください。COPY コマンドで REGION オプションを使用することによって、別のリージョンを指定できます。

> **Note**
>
> クラスターとデータソースが異なる AWS リージョンにある場合、データ転送コストが発生します。レイテンシーも高くなります。

COPY コマンドが失敗する

STL_LOAD_ERRORS にクエリして、特定のロード中に発生したエラーを見つけます。詳細については、「STL_LOAD_ERRORS」を参照してください。

ロードに時間がかかりすぎる

次の理由で、ロード操作に時間がかかりすぎる場合があります。以下のトラブルシューティングアプローチをお勧めします。

COPY が 1 つのファイルからデータをロードする

ロードデータを複数のファイルに分割します。1 つの大容量ファイルからすべてのデータをロードする場合、Amazon Redshift は低速なシリアル化されたロードを実行します。ファイル数は、クラスター内のスライス数の倍数にする必要があり、ファイルはほぼ同じサイズ (圧縮後 1 MB ~ 1 GB) にする必要があります。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。

ロード操作で複数の COPY コマンドを使用する

複数の COPY コマンドを同時に使用して複数のファイルから 1 つのテーブルをロードする場合、Amazon Redshift は低速なシリアル化されたロードを実行します。この場合、1 つの COPY コマンドを使用します。

ロードデータが正しくない

COPY オペレーションが、次の方法で誤ったデータをロードする場合があります。以下のトラブルシューティングアプローチをお勧めします。

違うファイルがロードされる

オブジェクトプレフィックスを使用してデータファイルを指定すると、不要なファイルが読み取られることがあります。ロードするファイルを正確に指定するには、代わりにマニフェストファイルを使

用します。詳細については、COPY コマンドの copy_from_s3_manifest_file オプションと COPY の例の Example: COPY from Amazon S3 using a manifest を参照してください。

JDBC フェッチサイズパラメータの設定

デフォルトでは、JDBC ドライバーはクエリに対して一度にすべての結果を収集します。その結果、JDBC 接続で大きな結果セットを取得しようとすると、クライアント側のメモリ不足エラーが発生する可能性があります。クライアントが 1 つのオールオアナッシングの取得ではなくバッチで結果セットを取得できるようにするには、JDBC フェッチサイズパラメータをクライアントアプリケーションで設定します。

> **Note**
>
> フェッチサイズは ODBC ではサポートされません。

最適なパフォーマンスのためには、メモリ不足エラーが発生しない最大の値にフェッチサイズを設定します。フェッチサイズの値を低く設定すると、サーバートリップが増え、それにより実行時間が長くなります。サーバーは、クライアントが結果セット全体を取得するまで、WLM クエリスロットおよび関連メモリを含むリソースを予約します。そうでない場合、クエリはキャンセルされます。フェッチサイズを適切に調整すると、それらのリソースはより迅速に解放され、他のクエリに利用できるようになります。

> **Note**
>
> 大きなデータセットを抽出する必要がある場合は、UNLOAD ステートメントを使用してデータを Amazon S3 に転送することをお勧めします。UNLOAD を使用するときは、コンピューティングノードは並行してデータの転送を高速化します。

JDBC フェッチサイズパラメータの詳細については、PostgreSQL のドキュメントで「Getting results based on a cursor」を参照してください。

ワークロード管理

Amazon Redshift WLM は、自動 WLM または手動 WLM で実行するように設定できます。

Amazon Redshift では、同時実行クエリとユーザーワークロードを管理および優先順位付けして、パフォーマンスとリソース使用率を最適化できます。ワークロード管理 (WLM) を使用すると、キュー、ユーザーグループ、その他のコンストラクトを定義して、さまざまなタイプのクエリやユーザーに割り当てられたリソースを制御できます。

以下のセクションでは、Amazon Redshift のワークロード管理機能の概要、および管理機能の設定とモニタリングについて説明します。

自動 WLM

システムのスループットを最大化し、リソースを効率的に使用するには、Amazon Redshift で、同時実行クエリを自動 WLM で実行するためのリソースの分割方法を管理できます。自動 WLM は、クエリの実行に必要なリソースを管理します。Amazon Redshift は、ディスパッチされたクエリごとに割り当てる同時実行クエリ数とメモリ量を決定します。Amazon Redshift で、同時実行クエリを実行するためにリソースの分割方法を管理する場合は、自動 WLM を使用します。詳細については、「自動 WLM の実装」を参照してください。

同時実行スケーリングや自動 WLM を使用すると、一貫した高速のクエリパフォーマンスで、事実上無制限の同時ユーザーと同時クエリをサポートできます。詳細については、「同時実行スケーリング」を参照してください。

> **Note**
>
> ほとんどの場合では、自動 WLM を使用することをお勧めします。手動 WLM を使用していて、自動 WLM から/に移行する場合は、「手動 WLM から自動 WLM に移行する」を参照してください。

自動 WLM では、キュー内のワークロードのクエリ優先順位を定義できます。クエリの優先度の詳細については、「クエリ優先度」を参照してください。

手動 WLM

複数のセッションやユーザーが同時にクエリを実行している場合があります。一部のクエリがクラスターリソースを長時間消費して、他のクエリのパフォーマンスに影響を与えることがあります。手動

WLM は、特殊なユースケースでこれを管理するのに役立ちます。同時実行をより細かく制御する場合は、手動 WLM を使用してください。

WLM 設定を変更して、実行時間が長いクエリと短いクエリ用に異なるキューを作成することによって、システムパフォーマンスを管理できます。実行時に、ユーザーグループまたはクエリグループに従って、これらのクエリをキューに配信できます。

クエリを実行するユーザーや指定したラベルに基づいて特定のキューにクエリをルーティングするためのルールを設定できます。各キューに割り当てるメモリの量を設定することによって、大きいクエリを他のキューよりメモリの多いキューで実行することもできます。また、実行時間が長いクエリを制限するようにクエリモニタリングルール (QMR) を設定することもできます。詳細については、「手動 WLM を実装する」を参照してください。

> **Note**
>
> 手動 WLM クエリキューのクエリスロットは合計 15 個以下にすることをお勧めします。詳細については、「同時実行レベル」を参照してください。

手動 WLM 設定では、キューに割り当てることができる最大スロット数は 50 であることに注意してください。ただし、これは、自動 WLM 設定で Amazon Redshift クラスターが常に 50 個のクエリを同時に実行するという意味ではありません。これは、必要なメモリやクラスター上の他の種類のリソース割り当てに基づいて変わる可能性があります。

トピック

- WLM システムテーブルとビュー

WLM モードの切り替え

Amazon Redshift コンソールを使用して、自動または手動 WLM を有効にできます。

1. [Switch WLM mode (WLM モードの切り替え)] を選択します。
2. 自動 WLM に設定するには、自動 WLM を選択します。このオプションでは、最大 8 個のキューを使用してクエリを管理します。[メモリ] フィールドと [Concurrency on main (メインでの同時実行数)] フィールドはいずれも、[Auto (自動)] に設定されます。デフォルトでは、クエリの優先度は、[通常] に設定されます。
3. この手動設定を有効にするには、Amazon Redshift コンソールで [手動 WLM] に切り替えます。このオプションでは、管理する対象のキューと、[メモリ] フィールドおよび [Concurrency on main (メインでの同時実行数)] フィールドの値を指定します。手動設定では、クエリキューを最大 8 個設定できます。さらに、これらのキューごとに同時実行できるクエリの数を設定できます。

WLM 設定の変更

WLM の設定を変更する最も簡単な方法は、Amazon Redshift コンソールを使用することです。AWS CLI または Amazon Redshift API を使用することもできます。

クラスターの自動 WLM と手動 WLM を切り替えると、クラスターは `pending reboot` 状態になります。次のクラスターを再起動するまで、変更は有効になりません。

WLM 設定の変更についての詳細は、Amazon Redshift 管理ガイドの「ワークロード管理の設定」を参照してください。

手動 WLM から自動 WLM に移行する

システムのスループットを最大化し、リソースを最も効果的に使用するには、キューに自動 WLM を設定することをお勧めします。手動 WLM から自動 WLM へのスムーズな移行をセットアップするには、次のアプローチを検討してください。

手動 WLM から自動 WLM に移行し、クエリの優先度を使用するには、新しいパラメータグループを作成し、そのパラメータグループをクラスターにアタッチすることをお勧めします。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift パラメータグループ」を参照してください。

> **Important**
>
> パラメータグループを変更したり、手動から自動 WLM に切り替えたりするには、クラスターを再起動する必要があります。詳細については、「WLM の動的設定プロパティと静的設定プロパティ」を参照してください。

3 つの手動 WLM キューがある例を見てみましょう。ETL ワークロード、分析ワークロード、およびデータサイエンスワークロードごとに 1 つ。ETL ワークロードは 6 時間ごとに実行され、分析ワークロードは終日実行されており、データサイエンスワークロードはいつでも急増する可能性があります。手動 WLM では、ビジネスに対する各ワークロードの重要性の理解に基づいて、各ワークロードキューが取得するメモリと同時実行性を指定します。メモリと同時実行性を指定するのは、理解するのが難しいだけでなく、クラスターリソースが静的に分割され、ワークロードのサブセットのみが実行されているときに無駄になります。

クエリの優先度を指定した自動 WLMを使用して、ワークロードの相対的な優先度を示し、前述の問題を回避できます。この例では、以下のステップを行います。

- 新しいパラメータグループを作成し、[Auto WLM] モードに切り替えます。
- ETL ワークロード、分析ワークロード、データサイエンスワークロードの 3 つのワークロードのそれぞれにキューを追加します。手動 WLM モードで使用されたワークロードごとに、同じユーザーグループを使用します。
- ETL ワークロードの優先度を `High`、分析ワークロードは `Normal`、データサイエンスは `Low` に設定します。これらの優先順位は、さまざまなワークロードまたはユーザーグループに対するビジネスの優先順位を反映しています。
- オプションで、ETL ワークロードが 6 時間ごとに実行されている場合でも、これらのキューのクエリで一貫したパフォーマンスが得られるように、分析キューまたはデータサイエンスキューの同時実行スケーリングを有効にします。

クエリの優先度を使用すると、クラスターで分析ワークロードのみが実行されている場合は、システム全体を独占できます。これにより、高いスループットが得られ、システム使用率も向上します。ただし、ETL ワークロードは優先度が高いため、このワークロードが開始されると優先して処理されます。ETL ワークロードの一部として実行されるクエリは、受け入れ時だけでなく、受け入れ後も優先リソース割り当てを得ることができます。結果として、ETL ワークロードは、システムで他に何が実行されているかに関係なく、予測どおりに実行されます。優先度の高いワークロードの予測可能なパフォーマンスには、実行時間が長く、優先度の低い他のワークロードのコストがかかります。

これは、クエリで、重要度が高いクエリが完了するのを待機しているか、優先度の高いクエリを同時に実行している場合は、リソースの割合が小さくなるためです。Amazon Redshift で使用されているスケジューリングアルゴリズムにより、優先度の低いクエリがリソース不足に悩まされることはなく、遅いペースで進行し続けます。

> **Note**
>
> - タイムアウトフィールドは、自動 WLM では使用できません。代わりに、QMR ルールである `query_execution_time` を使用します。詳細については、「WLM クエリモニタリングルール」を参照してください。
> - QMR アクションである HOP は、自動 WLM には使用できません。代わりに、`change priority` アクションを使用します。詳細については、「WLM クエリモニタリングルール」を参照してください。
> - クラスターでは、自動 WLM キューと手動 WLM キューの使用方法が異なるため、設定が混乱する可能性があります。例えば、自動 WLM キューでは優先度プロパティを設定できますが、手動 WLM キューでは設定できません。パラメータグループ内で、自動 WLM キューと手動 WLM キューを混在させないようにしてください。代わりに、自動 WLM に移行するときに新しいパラメータグループを作成します。

自動 WLM の実装

自動ワークロード管理 (WLM) では、Amazon Redshift がクエリの同時実行数とメモリの割り当てを管理します。サービスクラスの識別子 100〜107 を使用して、最大 8 つのキューを作成できます。各キューには優先度があります。詳細については、「クエリ優先度」を参照してください。

自動 WLM は、クエリに必要なリソース量を決定し、ワークロードに基づいて同時実行数を調整します。大量のリソースを必要とするクエリがシステムにある場合 (大きなテーブル間のハッシュ結合など)、同時実行数は減ります。軽いクエリ (挿入、削除、スキャン、単純な集計など) を送信すると、同時実行数は増えます。

自動 WLM は、ショートクエリアクセラレーション (SQA) とは別のものであり、クエリの評価方法が異なります。自動 WLM と SQA は連携して動作し、長時間実行されるリソース集約型のクエリがアクティブな場合でも、短時間実行されるクエリや軽量のクエリを完了させることができます。SQA の詳細については、「ショートクエリアクセラレーション」を参照してください。

Amazon Redshift では、パラメータグループを使用して、自動 WLM を有効にします。

- クラスターでデフォルトのパラメータグループを使用している場合は、Amazon Redshift のクラスターで自動 WLM が有効になります。
- クラスターでカスタムパラメータグループを使用している場合は、自動 WLM を有効にするようにクラスターを設定することができます。自動 WLM の設定用に個別のパラメータグループを作成することをお勧めします。

WLM を設定するには、1 つ以上のクラスターと関連付けることのできるパラメータグループの `wlm_json_configuration` パラメータを編集します。詳細については、「WLM 設定の変更」を参照してください。

クエリキューは、WLM 設定内で定義することができます。デフォルトの WLM 設定にクエリキューを追加することができます (最大合計 8 つのユーザーキュー)。クエリキューごとに以下を設定できます。

- 優先度
- 同時実行スケーリングモード
- ユーザーグループ
- クエリグループ
- クエリのモニタリングルール

優先度

ワークロードでのクエリの相対的な重要度を定義するには、優先度の値を設定します。優先度はキューに指定され、キューに関連付けられたすべてのクエリに継承されます。詳細については、「クエリ優先度」を参照してください。

同時実行スケーリングモード

同時実行スケーリングが有効化された状態で、読み取りと書き込みクエリの同時実行の増加に対応する必要がある場合、Amazon Redshift は新たなクラスター容量を自動的に追加します。クエリをメインクラスターと同時実行スケーリングクラスターのどちらで実行しても、ユーザーには最新のデータが表示されます。

WLM キューを設定することで、どのクエリを同時実行スケーリングクラスターに送信するかを管理します。キューに対して同時実行スケーリングを有効にすると、対象クエリは待機することなく同時実行クラスターに送信されます。詳細については、「同時実行スケーリング」を参照してください。

ユーザーグループ

各ユーザーグループ名を指定するか、ワイルドカードを使用して、一連のユーザーグループをキューに割り当てることができます。リストされたユーザーグループのメンバーがクエリを実行すると、そのクエリは対応するキューで実行されます。キューに割り当てることができるユーザーグループの数に設定された制限はありません。詳細については、「ユーザーグループに基づくクエリのキューへの割り当て」を参照してください。

ユーザーロール

各ユーザーロールを指定するか、ワイルドカードを使用して、一連のユーザーロールをキューに割り当てることができます。リストされたユーザーロールのメンバーがクエリを実行すると、そのクエリは対応するキューで実行されます。キューに割り当てることができるユーザーロールの数に設定された制限はありません。詳細については、「 ユーザーロールに基づくクエリのキューへの割り当て」を参照してください。

クエリグループ

各クエリグループ名を指定するか、ワイルドカードを使用して、一連のクエリグループをキューに割り当てることができます。クエリグループはラベルにすぎません。実行時に、一連のクエリにクエリグループラベルを割り当てることができます。一覧表示されたクエリグループに割り当てられたクエリは、対応するキューで実行されます。キューに割り当てることができるクエリグループの数に設定された制限はありません。詳細については、「クエリグループへのクエリの割り当て」を参照してください。

ワイルドカード

WLM キュー設定でワイルドカードを有効にした場合は、ユーザーグループやクエリグループをキューに個別に割り当てるか、Unix シェル形式のワイルドカードを使用して割り当てることができます。パターンマッチングでは大文字と小文字が区別されません。

例えば、ワイルドカード文字「*」は任意の文字数に一致します。例えば、キューのユーザーグループのリストに dba_* を追加すると、dba_ で始まる名前を持つグループに属する、ユーザーが実行するすべてのクエリは、そのキューに割り当てられます。たとえば、dba_admin や DBA_primary などがあります。ワイルドカード文字「?」は、任意の 1 文字に一致します。したがって、キューにユーザーグループ dba?1 が含まれている場合、dba11 と dba21 は一致しますが、dba12 は一致しません。

デフォルトでは、ワイルドカードは有効になっていません。

クエリのモニタリングルール

クエリモニタリングルールは、WLM キューのメトリクスベースのパフォーマンスの境界を定義し、クエリがこれらの境界を超えた場合のアクションを指定します。例えば、実行時間の短いクエリ専用のキューには、60 秒以上実行されるクエリをキャンセルするルールを作成できます。デザインの不十分なクエリを追跡する目的で、ネステッドループを含むクエリを記録する別のルールを設定することができます。詳細については、「WLM クエリモニタリングルール」を参照してください。

自動 WLM の設定を確認する

自動 WLM が有効になっているかどうかを確認するには、次のクエリを実行します。クエリより 1 行以上が返る場合、自動 WLM は有効になっています。

```
select * from stv_wlm_service_class_config
where service_class >= 100;
```

次のクエリは、各クエリキュー (サービスクラス) を通過したクエリの数を示しています。また、平均実行時間、待機時間が発生しているクエリの数 (90 パーセンタイル値)、平均待機時間を示しています。自動 WLM クエリでは、サービスクラス 100～107 を使用します。

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

自動 WLM で実行されて正常に完了したクエリを確認するには、次のクエリを実行します。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class >= 100 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

クエリ優先度

Amazon Redshift では、ワークロード管理 (WM) を使用して、同時実行クエリとワークロード間でクエリの優先順位付けとリソース割り当てを管理できます。以下のセクションでは、WM クエリキューの設定、メモリ割り当てや同時実行スケーリングなどのキュープロパティの定義、ワークロード要件に合わせた優先度ルールの実装方法について説明します。

すべてのクエリの重要性が同じわけではなく、多くの場合、1 つのワークロードまたはユーザーセットのパフォーマンスが重要になる場合があります。自動 WLM を有効にしている場合は、優先度の値

を設定して、ワークロードでのクエリの相対的な重要度を定義することができます。優先度はキューに指定され、キューに関連付けられたすべてのクエリに継承されます。クエリをキューに関連付けるには、ユーザーグループとクエリグループをキューにマッピングします。次の優先度を設定することができます (優先順位が高 ~ 低の順に表示)。

1. HIGHEST
2. HIGH
3. NORMAL
4. LOW
5. LOWEST

管理者は、これらの優先度を使用して、同じリソースに対して競合する異なる優先度のクエリがある場合に、ワークロードの相対的な重要性を示します。Amazon Redshift は、システムにクエリを許可するとき、およびクエリに割り当てられるリソースの量を決定するときに優先度を使用します。デフォルトでは、クエリは、優先度を NORMAL に設定して実行されます。

追加の優先度 CRITICAL は、HIGHEST よりも優先度が高く、スーパーユーザーが利用できます。この優先度を設定するには、CHANGE_QUERY_PRIORITY 関数、 CHANGE_SESSION_PRIORITY 関数、および CHANGE_USER_PRIORITY 関数を使用できます。これらの関数を使用するアクセス許可をデータベースユーザーに付与するには、ストアドプロシージャを作成し、ユーザーにアクセス許可を付与します。例については、「CHANGE_SESSION_PRIORITY」を参照してください。

> **Note**
>
> 一度に実行できる CRITICAL クエリは 1 つのみです。

抽出、変換、ロード (ETL) ワークロードの優先度が分析ワークロードの優先度よりも高い例を見てみましょう。ETL ワークロードは 6 時間ごとに実行され、分析ワークロードは 1 日中実行されます。クラスターで分析ワークロードのみが実行されている場合は、システム全体が最適化され、最適なシステム使用率で高いスループットが得られます。ただし、ETL ワークロードは優先度が高いため、このワークロードが開始されると優先して処理されます。ETL ワークロードの一部として実行されるクエリは、受け入れ中と、受け入れ後の優先リソース割り当て中に優先度が上がります。結果として、ETL ワークロードは、システムで他に何が実行されているかに関係なく、予測どおりに実行されます。そのため、予測可能なパフォーマンスと、管理者がビジネスユーザーにサービスレベルアグリーメント (SLA) を提示する機能を提供します。

特定のクラスター内では、優先度の高いワークロードの予測可能なパフォーマンスは、優先度の低い他のワークロードのコストになります。優先度の低いワークロードでは、重要の高いクエリが完了するまで待機するため、実行時間が長くなる場合があります。また、優先度の高いクエリを同時に実行している場合は、リソースの割合が小さくなるために実行時間が長くなる場合があります。優先度の低いクエリでリソース不足に悩まされることはありませんが、進行するペースは遅くなります。

前述の例で、管理者は、分析ワークロードの同時実行スケーリングを有効にすることができます。これにより、ETL ワークロードが高い優先度で実行されていても、そのワークロードはスループットを維持できます。

キュー優先度を設定する

自動 WLM を有効にしている場合、各キューには優先度の値があります。クエリは、ユーザーグループとクエリグループに基づいてキューにルーティングされます。キューの優先順位を `NORMAL` に設定して開始します。キューのユーザーグループとクエリグループに関連付けられたワークロードに基づいて、優先度を高くまたは低く設定します。

キューの優先度は、Amazon Redshift コンソールで変更できます。Amazon Redshift コンソールの [Workload Management] (ワークロード管理) ページにキューが表示され、[Priority] (優先度) などのキューのプロパティを編集できるようになります。CLI または API オペレーションを使用して優先度を設定するには、`wlm_json_configuration` パラメータを使用します。詳細については、「Amazon Redshift 管理ガイド」の「ワークロード管理の設定」を参照してください。

次の `wlm_json_configuration` の例では、3 つのユーザーグループ (`ingest`、`reporting`、`analytics`) を定義します。これらのグループのいずれかのユーザーから送信されたクエリは、優先度 (`highest`、`normal`、`low`) のそれぞれで実行されます。

```
[
    {
        "user_group": [
            "ingest"
        ],
        "priority": "highest",
        "queue_type": "auto"
    },
    {
        "user_group": [
            "reporting"
        ],
        "priority": "normal",
```

```
        "queue_type": "auto"
    },
    {
        "user_group": [
            "analytics"
        ],
        "priority": "low",
        "queue_type": "auto",
        "auto_wlm": true
    }
]
```

クエリモニタリングルールを使用してクエリ優先度を変更する

クエリモニタリングルール (QMR) を使用すると、実行中の動作に基づいてクエリの優先度を変更できます。変更するには、アクションに加えて、QMR 述語で priority 属性を指定します。詳細については、「WLM クエリモニタリングルール」を参照してください。

例えば、10 分以上実行される優先度 `high` として分類されたクエリをキャンセルするルールを定義できます。

```
"rules" :[
  {
    "rule_name":"rule_abort",
    "predicate":[
      {
        "metric_name":"query_cpu_time",
        "operator":">",
        "value":600
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"high"
      }
    ],
    "action":"abort"
  }
]
```

もう一方の例では、1 TB を超えるディスクに溢れる現在の優先度 `normal` のクエリのクエリ優先度を `lowest` に変更するルールを定義します。

```
"rules":[
  {
    "rule_name":"rule_change_priority",
    "predicate":[
      {
        "metric_name":"query_temp_blocks_to_disk",
        "operator":">",
        "value":1000000
      },
      {
        "metric_name":"query_priority",
        "operator":"=",
        "value":"normal"
      }
    ],
    "action":"change_query_priority",
    "value":"lowest"
  }
]
```

クエリ優先度をモニタリングする

クエリの待機および実行の優先度を表示するには、stv_wlm_query_state システムテーブルの `query_priority` 列を表示します。

```
query    | service_cl | wlm_start_time             | state            | queue_time |
 query_priority
---------+------------+----------------------------+------------------+------------
+----------------
2673299  | 102        | 2019-06-24 17:35:38.866356 | QueuedWaiting    | 265116     |
 Highest
2673236  | 101        | 2019-06-24 17:35:33.313854 | Running          | 0          |
 Highest
2673265  | 102        | 2019-06-24 17:35:33.523332 | Running          | 0          |
 High
2673284  | 102        | 2019-06-24 17:35:38.477366 | Running          | 0          |
 Highest
2673288  | 102        | 2019-06-24 17:35:38.621819 | Running          | 0          |
 Highest
2673310  | 103        | 2019-06-24 17:35:39.068513 | QueuedWaiting    | 62970      |
 High
```

```
2673303  | 102        | 2019-06-24 17:35:38.968921 | QueuedWaiting    | 162560     |
 Normal
2673306  | 104        | 2019-06-24 17:35:39.002733 | QueuedWaiting    | 128691     |
 Lowest
```

完了したクエリのクエリ優先度をリストするには、stl_wlm_query システムテーブルの `query_priority` 列を参照してください。

```
select query, service_class as svclass, service_class_start_time as starttime,
 query_priority
from stl_wlm_query order by 3 desc limit 10;
```

```

  query   | svclass |          starttime          |    query_priority
---------+---------+----------------------------+----------------------
 2723254 |     100 | 2019-06-24 18:14:50.780094 | Normal
 2723251 |     102 | 2019-06-24 18:14:50.749961 | Highest
 2723246 |     102 | 2019-06-24 18:14:50.725275 | Highest
 2723244 |     103 | 2019-06-24 18:14:50.719241 | High
 2723243 |     101 | 2019-06-24 18:14:50.699325 | Low
 2723242 |     102 | 2019-06-24 18:14:50.692573 | Highest
 2723239 |     101 | 2019-06-24 18:14:50.668535 | Low
 2723237 |     102 | 2019-06-24 18:14:50.661918 | Highest
 2723236 |     102 | 2019-06-24 18:14:50.643636 | Highest
```

ワークロードのスループットを最適化するために、Amazon Redshift はユーザーが送信したクエリの優先順位を変更することがあります。Amazon Redshift は、高度な機械学習アルゴリズムを使用して、この最適化がワークロードに利点をもたらす時期を判断し、次のすべての条件が満たされたときに自動的に適用します。

- 自動 WLM は有効になっています。
- 定義される WLM キューは 1 つだけです。
- クエリの優先順位を設定するクエリモニタリングルール (QMR) が定義されていません。このようなルールには、QMR メトリクス `query_priority` または QMR アクション `change_query_priority` が含まれます。詳細については、「[WLM クエリモニタリングルール](#)」を参照してください。

手動 WLM を実装する

手動 WLM では、WLM 設定を変更して、実行時間が長いクエリと短いクエリ用に異なるキューを作成することによって、システムパフォーマンスとユーザーエクスペリエンスを管理できます。

ユーザーが Amazon Redshift でクエリを実行すると、クエリはクエリキューにルーティングされます。各クエリキューには、いくつかのクエリスロットが含まれています。各キューには、クラスターの使用可能なメモリの一部が割り当てられます。キューのメモリは、キューのクエリスロットに分けられます。Amazon Redshift では、自動 WLM を使用してクエリの同時実行数を管理できます。詳細については、「自動 WLM の実装」を参照してください。

または、クエリキューごとに WLM プロパティを設定できます。このようにして、メモリをスロット間に割り当てる方法と、ランタイムに特定のキューにクエリをルーティングする方法を指定します。実行時間の長いクエリをキャンセルするように WLM プロパティを設定することもできます。

デフォルトでは、Amazon Redshift は次のクエリキューを設定します。

- 1 つのスーパーユーザーキュー

  スーパーユーザーキューは、スーパーユーザー専用に予約されており、設定することはできません。このキューを使用するのは、システムに影響を与えるクエリを実行したり、トラブルシューティング目的でクエリを実行したりする場合に限ります。例えば、ユーザーの実行時間が長いクエリをキャンセルしたり、ユーザーをデータベースに追加したりする必要がある場合にこのキューを使用します。これを使用してルーチンクエリを実行しないでください。このキューはコンソールには表示されませんが、データベースのシステムテーブルに 5 番目のキューとして表示されます。スーパーユーザーキューでクエリを実行するには、ユーザーはスーパーユーザーとしてログインし、事前定義された `superuser` クエリグループを使用してクエリを実行する必要があります。

- 1 つのデフォルトのユーザーキュー

  デフォルトキューは、最初は 5 つのクエリを同時に実行するように設定されています。手動 WLM を使用するときには、デフォルトキューの同時実行数、タイムアウト、メモリ割り当ての各プロパティを変更することはできますが、ユーザーグループまたはクエリグループを指定することはできません。デフォルトキューは、WLM 設定の最後のキューにする必要があります。他のキューにルーティングされないクエリはすべて、デフォルトキューで実行されます。

クエリキューは WLM 設定で定義されます。WLM 設定はパラメータグループの編集可能なパラメータ（`wlm_json_configuration`）であり、1 つ以上のクラスターと関連付けることができます。

詳細については、「Amazon Redshift 管理ガイド」の「ワークロード管理の設定」を参照してください。

デフォルトの WLM 設定にクエリキューを追加することができます (最大合計 8 つのユーザーキュー)。クエリキューごとに以下を設定できます。

- 同時実行スケーリングモード
- 同時実行レベル
- ユーザーグループ
- クエリグループ
- 使用する WLM メモリの割合
- WLM タイムアウト
- WLM クエリキューのホッピング
- クエリのモニタリングルール

同時実行スケーリングモード

同時実行スケーリングが有効化された状態で、読み取りと書き込みクエリの同時実行の増加に対応する必要がある場合、Amazon Redshift は新たなクラスター容量を自動的に追加します。クエリをメインクラスターと同時実行スケーリングクラスターのどちらで実行しても、ユーザーには最新のデータが表示されます。

WLM キューを設定することで、どのクエリを同時実行スケーリングクラスターに送信するかを管理します。キューに対して同時実行スケーリングを有効にすると、対象クエリは待機することなく同時実行クラスターに送信されます。詳細については、「同時実行スケーリング」を参照してください。

同時実行レベル

キューのクエリは、キューに対して定義された WLM クエリスロットの数または同時実行レベルに達するまで、同時に実行されます。その後、以降のキューはキューで待機します。

> **Note**
>
> WLM の同時実行レベルは、クラスターに対して同時実行可能なユーザー接続の数とは異なります。詳細については、「Amazon Redshift 管理ガイド」の「クラスターへの接続」を参照してください。

自動 WLM 設定 (推奨) では、同時実行レベルは [自動] に設定されます。Amazon Redshift はメモリをクエリに動的に割り当てます。その後、同時実行数が決まります。これは、実行中のクエリとキューに登録されたクエリの両方に必要なリソースに基づきます。自動 WLM は設定できません。詳細については、「自動 WLM の実装」を参照してください。

手動 WLM 設定では、Amazon Redshift は各キューに固定量のメモリを静的に割り当てます。キューのメモリは、クエリスロットに均等に分割されます。例として、キューにクラスターのメモリの 20% が割り当てられ、スロットが 10 個ある場合、各クエリにはクラスターのメモリの 2% が割り当てられます。メモリ割り当ては、同時実行されるクエリの数とは関係なく、固定量のままです。このようにメモリ割り当てが固定されているため、スロット数が 5 のときにメモリ内で全体を実行できるクエリは、スロット数が 20 に引き上げられると、中間結果をディスクに書き込むことが必要になる場合があります。この場合、キューのメモリにおける各クエリのシェアは、5 分の 1 から 20 分の 1 に減少します。ディスク I/O が余分に発生すると、パフォーマンスが低下する可能性があります。

すべてのユーザー定義キューの最大スロット数は 50 です。このため、デフォルトキューを含むすべてのキューの合計スロット数が制限されます。制限の対象とならない唯一のキューは、予約済みのスーパーユーザーキューです。

デフォルトでは、手動 WLM キューの同時実行レベルは 5 です。次のような場合は、同時実行レベルを引き上げるとワークロードに良い影響があることがあります。

- 多数の小さいクエリが長時間実行クエリを強制的に待機させられている場合は、より多いスロット数のキューを別に作成し、小さいクエリをそのキューに割り当てます。キューの同時実行レベルが高いと各クエリスロットに割り当てられるメモリは少なくなるものの、クエリが小さい方が必要なメモリが少なくなります。

> **Note**
>
> ショートクエリアクセラレーション (SQA) を有効にすると、WLM によって実行時間が短いクエリが実行時間が長いクエリよりも優先されます。このため、ショートクエリ用の個別のキューは、大部分のワークフローで不要になります。詳細については、「ショートクエリアクセラレーション」を参照してください。

- それぞれが単一のスライス上のデータにアクセスする複数のクエリがある場合は、それらのクエリを同時に実行するように個別の WLM キューを設定します。Amazon Redshift は、同時実行クエリを別々のスライスに割り当てます。これにより、複数のクエリを複数のスライスで並行して実行できます。例えば、クエリが分散キーに関する述語を持つ単純な集計である場合、クエリのデータは単一のスライスに配置されます。

手動 WLM の例

この例は、スロットとメモリの割り当て方法を示す簡単な手動 WLM シナリオです。手動 WLM は、次の 3 つのキューを使用して実装します。

- データインジェストキュー - これは、データを取り込むために設定されます。クラスターのメモリの 20% が割り当てられ、5 つのスロットがあります。その後、キュー内で 5 つのクエリを同時に実行でき、それぞれにメモリの 4% が割り当てられます。
- データサイエンティストキュー - これは、メモリを大量に消費するクエリ用に設計されています。クラスターのメモリの 20% が割り当てられ、5 つのスロットがあります。その後、キュー内で 5 つのクエリを同時に実行でき、それぞれにメモリの 8% が割り当てられます。
- デフォルトキュー - これは組織内のほとんどのユーザー向けに設計されています。これには、一般にクエリの実行時間が短いか中程度であり、複雑ではない営業グループや経理グループも含まれます。クラスターのメモリの 40% が割り当てられ、40 のスロットがあります。このキューでは 40 のクエリを同時に実行でき、各クエリにはメモリの 1% が割り当てられます。すべてのキューの制限は 50 であるため、これは、このキューに割り当てることができる最大スロット数です。

自動 WLM を実行し、15 を超えるクエリを並列実行する必要があるワークロードの場合は、同時実行スケーリングを有効にすることをお勧めします。これは、クエリスロット数を 15 より大きくすると、システムリソースの競合が発生し、単一クラスターの全体的なスループットが制限される可能性があるためです。同時実行スケーリングでは、設定された数の同時実行スケーリングクラスターまで数百のクエリを並列実行できます。同時実行スケーリングクラスターの数は、max_concurrency_scaling_clusters によって制御されます。同時実行スケーリングの詳細については、「同時実行スケーリング」を参照してください。

詳細については、「クエリパフォーマンスの向上」を参照してください。

ユーザーグループ

各ユーザーグループ名を指定するか、ワイルドカードを使用して、一連のユーザーグループをキューに割り当てることができます。リストされたユーザーグループのメンバーがクエリを実行すると、そのクエリは対応するキューで実行されます。キューに割り当てることができるユーザーグループの数に設定された制限はありません。詳細については、「ユーザーグループに基づくクエリのキューへの割り当て」を参照してください。

ユーザーロール

各ユーザーロールを指定するか、ワイルドカードを使用して、一連のユーザーロールをキューに割り当てることができます。リストされたユーザーロールのメンバーがクエリを実行すると、そのクエリは対応するキューで実行されます。キューに割り当てることができるユーザーロールの数に設定された制限はありません。詳細については、「ユーザーロールに基づくクエリのキューへの割り当て」を参照してください。

クエリグループ

各クエリグループ名を指定するか、ワイルドカードを使用して、一連のクエリグループをキューに割り当てることができます。クエリグループはラベルにすぎません。実行時に、一連のクエリにクエリグループラベルを割り当てることができます。一覧表示されたクエリグループに割り当てられたクエリは、対応するキューで実行されます。キューに割り当てることができるクエリグループの数に設定された制限はありません。詳細については、「クエリグループへのクエリの割り当て」を参照してください。

ワイルドカード

WLM キュー設定でワイルドカードを有効にした場合は、ユーザーグループやクエリグループをキューに個別に割り当てるか、Unix シェル形式のワイルドカードを使用して割り当てることができます。パターンマッチングでは大文字と小文字が区別されません。

例えば、ワイルドカード文字「*」は任意の文字数に一致します。例えば、キューのユーザーグループのリストに dba_* を追加すると、dba_ で始まる名前を持つグループに属する、ユーザーが実行するすべてのクエリは、そのキューに割り当てられます。例は、`dba_admin` や `DBA_primary` です。ワイルドカード文字「?」は、任意の 1 文字に一致します。したがって、キューにユーザーグループ dba?1 が含まれている場合、`dba11` と `dba21` は一致しますが、`dba12` は一致しません。

ワイルドカードはデフォルトでオフになっています。

使用する WLM メモリの割合

自動 WLM 設定の場合、メモリの割合は **`auto`** に設定されます。詳細については、「自動 WLM の実装」を参照してください。

手動 WLM 設定の場合、クエリに割り当てる使用可能なメモリの量を指定するには、`WLM Memory Percent to Use` パラメータを設定できます。デフォルトでは、各ユーザー定義キューには、ユー

ザー定義クエリで使用可能なメモリが均等に割り当てられます。例えば、4 つのユーザー定義キューがある場合、各キューには使用可能なメモリの 25 パーセントが割り当てられます。Superuser キューには、独自に割り当てられているメモリがあり、変更できません。割り当て量を変更するには、各キューのメモリの割合を整数で割り当てます (最大で合計 100 パーセント)。未割り当てのメモリは Amazon Redshift によって管理され、処理用に追加メモリをリクエストするキューに一時的に付与できます。

例えば、4 つのキューを設定する場合、20 パーセント、30 パーセント、15 パーセント、15 パーセントのようにメモリを割り当てることができます。残りの 20 パーセントは未割り当てになり、サービスによって管理されます。

WLM タイムアウト

WLM タイムアウト (`max_execution_time`) は廃止されました。代わりに、`query_execution_time` を使用してクエリモニタリングルール (QMR) を作成して、経過したクエリ実行時間を制限します。詳細については、「WLM クエリモニタリングルール」を参照してください。

各キューについて WLM タイムアウト値を設定することで、特定の WLM キュー内でクエリが使用できる時間を制限することができます。タイムアウトパラメータは、クエリをキャンセルまたはホップする前に Amazon Redshift がクエリの実行を待機する時間を、ミリ秒単位で指定します。タイムアウトはクエリの実行時間に基づいていて、キューでの待機時間は含まれません。

WLM は、CREATE TABLE AS (CTAS) ステートメントと読み取り専用クエリ (SELECT ステートメントなど) のホップを試みます。ホップできないクエリはキャンセルされます。詳細については、「WLM クエリキューのホッピング」を参照してください。

WLM タイムアウトは returning 状態に達したクエリには適用されません。クエリの状態を表示するには、STV_WLM_QUERY_STATE システムテーブルを参照してください。COPY ステートメントと、ANALYZE や VACUUM などのメンテナンスオペレーションは、WLM タイムアウトの対象ではありません。

WLM タイムアウトの機能は、statement_timeout 設定パラメータと似ています。相違点は、`statement_timeout` 設定パラメータがクラスター全体に適用されるのに対して、WLM タイムアウトは WLM 設定の単一のキューに固有であることです。

statement_timeout も指定されている場合、statement_timeout および WLM タイムアウト (max_execution_time) の低い方が使用されます。

クエリのモニタリングルール

クエリモニタリングルールは、WLM キューのメトリクスベースのパフォーマンスの境界を定義し、クエリがこれらの境界を超えた場合のアクションを指定します。例えば、実行時間の短いクエリ専用のキューには、60 秒以上実行されるクエリをキャンセルするルールを作成できます。デザインの不十分なクエリを追跡する目的で、ネステッドループを含むクエリを記録する別のルールを設定することができます。詳細については、「WLM クエリモニタリングルール」を参照してください。

WLM クエリキューのホッピング

Amazon Redshift では、WLM (ワークロード管理) のクエリキューホッピングを有効にすることで、ワークロードの同時実行とリソース割り当てを管理できます。この機能を使用すると、リソースが利用可能になったときに、割り当てられたキューから優先度の高いキューに一時的に「ホップ」するクエリが可能になり、全体的なクエリパフォーマンスとシステム使用率が向上します。以下のセクションでは、Amazon Redshift での WLM クエリキューホッピングの設定と使用に関する詳細なガイダンスを提供します。

クエリは、WLM タイムアウトまたはクエリモニタリングルール (QMR) のホップアクションに基づいてホップされる場合があります。クエリは、手動 WLM 設定でのみホップできます。

クエリがホップされると、WLM は、WLM キュー割り当てルールに基づき、次に一致するキューへのクエリのルーティングを試みます。クエリが他のいずれのキュー定義とも一致しない場合、クエリはキャンセルされます。クエリはデフォルトキューに割り当てられません。

WLM タイムアウトのアクション

次の表は、WLM タイムアウト時におけるクエリのタイプ別の動作をまとめたものです。

クエリタイプ	アクション
INSERT、UPDATE、および DELETE	キャンセル
ユーザー定義関数 (UDF)	キャンセル
UNLOAD	キャンセル
COPY	実行を継続
メンテナンスオペレーション	実行を継続

クエリタイプ	アクション
`returning` 状態の読み取り専用クエリ	実行を継続
`running` 状態の読み取り専用クエリ	再割り当てまたは再起動
CREATE TABLE AS (CTAS)、SELECT INTO	再割り当てまたは再起動

WLM タイムアウトキューのホッピング

WLM は、以下のタイプのクエリをタイムアウト時にホップします。

- WLM 状態が `running` の、SELECT ステートメントなどの読み取り専用クエリ。クエリの WLM 状態を確認するには、STV_WLM_QUERY_STATE システムテーブルで STATE 列を表示します。
- CREATE TABLE AS (CTAS) ステートメント。WLM キューのホッピングでは、ユーザー定義とシステム生成の両方の CTAS ステートメントがサポートされます。
- SELECT INTO ステートメント。

WLM タイムアウトの対象でないクエリは、完了するまで元のキューで実行を継続します。以下のタイプのクエリは、WLM タイムアウトの対象外です。

- COPY ステートメント
- ANALYZE や VACUUM などのメンテナンスオペレーション
- WLM 状態が `returning` になった、SELECT ステートメントなどの読み取り専用クエリ。クエリの WLM 状態を確認するには、STV_WLM_QUERY_STATE システムテーブルで STATE 列を表示します。

WLM タイムアウトによるホッピングの対象でないクエリは、タイムアウト時にキャンセルされます。以下のタイプのクエリは、WLM タイムアウトによるホッピングの対象外です。

- INSERT、UPDATE、および DELETE ステートメント
- UNLOAD ステートメント
- ユーザー定義関数 (UDF)

WLM タイムアウト時のクエリの再割り当てと再開

クエリがホップされ一致するキューが見つからない場合、クエリはキャンセルされます。

クエリがホップされ一致するキューが見つかった場合、WLM は新しいキューへのクエリの再割り当てを試みます。クエリを再割り当てすることができない場合は、以下に説明するように、クエリが新しいキューで再開されます。

クエリは以下のすべての条件が満たされた場合にのみ再割り当てされます。

- 一致するキューが見つかった。
- 新しいキューに、クエリを実行できる十分な空きスロットがある。wlm_query_slot_count パラメータが 1 以上の値に設定されている場合、クエリに複数のスロットが必要になる場合があります。
- 新しいキューのメモリ量が、クエリが現在使用しているメモリ量と少なくとも同じである。

再割り当てされたクエリは、新しいキューで実行を継続します。中間の結果は保持されるため、合計実行時間に対する影響は最小限です。

再割り当てできないクエリはキャンセルされ、新しいキューで再開されます。中間の結果は削除されます。クエリはキュー内で待機し、十分なスロット数が利用可能になると実行を開始します。

QMR ホップアクション

次の表は、クエリタイプ別に QMR ホップアクションに基づく動作をまとめたものです。

クエリタイプ	アクション
COPY	実行を継続
メンテナンスオペレーション	実行を継続
ユーザー定義関数 (UDF)	実行を継続
UNLOAD	再割り当てまたは実行を継続
INSERT、UPDATE、および DELETE	再割り当てまたは実行を継続
`returning` 状態の読み取り専用クエリ	再割り当てまたは実行を継続

クエリタイプ	アクション
`running` 状態の読み取り専用クエリ	再割り当てまたは再起動
CREATE TABLE AS (CTAS)、SELECT INTO	再割り当てまたは再起動

QMR によってホップされたクエリが再割り当て、再開、またはキャンセルされるかどうかを確認するには、STL_WLM_RULE_ACTION システムログテーブルにクエリを実行します。

QMR ホップアクションによるクエリの再割り当てと再開

クエリがホップされ一致するキューが見つからない場合、クエリはキャンセルされます。

クエリがホップされ一致するキューが見つかった場合、WLM は新しいキューへのクエリの再割り当てを試みます。クエリを再割り当てすることができない場合は、以下に説明するように、クエリが新しいキューで再開されるか、元のキューで実行を継続します。

クエリは以下のすべての条件が満たされた場合にのみ再割り当てされます。

- 一致するキューが見つかった。
- 新しいキューに、クエリを実行できる十分な空きスロットがある。wlm_query_slot_count パラメータが 1 以上の値に設定されている場合、クエリに複数のスロットが必要になる場合があります。
- 新しいキューのメモリ量が、クエリが現在使用しているメモリ量と少なくとも同じである。

再割り当てされたクエリは、新しいキューで実行を継続します。中間の結果は保持されるため、合計実行時間に対する影響は最小限です。

再割り当てできないクエリは、再開されるか元のキューで実行を継続します。クエリが再開される場合、そのクエリはキャンセルされ、新しいキューで再開されます。中間の結果は削除されます。クエリはキュー内で待機し、十分なスロット数が利用可能になると実行を開始します。

チュートリアル: 手動ワークロード管理 (WLM) キューの設定

Amazon Redshift では、手動ワークロード管理 (WLM) キューを設定して、さまざまなタイプのクエリやユーザーにリソースを優先して割り当てることができます。手動 WLM キューを使用すると、特定のキューに対するメモリと同時実行設定を制御できるため、優先度の低いクエリがシステムを占有するのを防ぎながら、重要なワークロードが必要なリソースを確実に得ることができます。以下のセ

クションでは、ワークロード管理要件を満たすために Amazon Redshift で手動 WLM キューを作成および設定するプロセスについて説明します。

概要

Amazon Redshift では自動ワークロード管理 (WLM) を設定することをお勧めします。自動 WLM の詳細については、「ワークロード管理」を参照してください。ただし、複数の WLM キューを必要とする場合のために、このチュートリアルでは、Amazon Redshift で手動ワークロード管理 (WLM) を設定するプロセスについて説明します。手動 WLM を設定することで、クラスターのクエリパフォーマンスとリソース割り当てを改善できます。

Amazon Redshift はユーザークエリをキューにルーティングして処理します。WLM は、クエリがキューにルーティングされる方法を定義します。デフォルトで、Amazon Redshift にはクエリに使用できるキューが 2 つあります。1 つはスーパーユーザー用で、もう 1 つはユーザー用です。スーパーユーザーキューは設定ができず、一度に 1 つのクエリしか処理できません。このキューはトラブルシューティング目的のみに使用してください。ユーザーキューは最大 5 件のクエリを一度に処理できますが、必要に応じてキューの同時実行レベルを変更することで、この件数を設定できます。

データベースに対して数人のユーザーがクエリを実行する場合は、別の設定の方が効率的なことがあります。例えば、一部のユーザーが VACUUM のように大量のリソースを使う操作を実行する場合、レポートのようにリソースをあまり使わないクエリに対して悪影響が生じることがあります。このような場合は、キューを追加して、異なるワークロード用に設定することを検討してください。

予測時間: 75 分

推定コスト: 50 セント

前提条件

Amazon Redshift クラスター、サンプル TICKIT データベース、Amazon Redshift RSQL クライアントツールが必要です。これらをまだセットアップしていない場合は、「Amazon Redshift 入門ガイド」と「Amazon Redshift RSQL」を参照してください。

セクション

- セクション 1: デフォルトキュー処理の動作の理解
- セクション 2: WLM のクエリキュー設定の変更
- セクション 3: ユーザーグループとクエリグループに基づいてクエリをキューにルーティング
- セクション 4: wlm_query_slot_count を使用してキューの同時実行レベルを一時的に変更
- セクション 5: リソースのクリーンアップ

セクション 1: デフォルトキュー処理の動作の理解

手動 WLM の設定を開始する前に、Amazon Redshift におけるキュー処理のデフォルト動作を理解しておくと役に立ちます。このセクションでは、いくつかのシステムテーブルから情報を返すデータベースビューを 2 つ作成します。その後、いくつかのテストクエリを実行して、クエリがデフォルトでどのようにルーティングされるかを確認します。システムテーブルの詳細については、「システムテーブルとビューのリファレンス」を参照してください。

ステップ 1: WLM_QUEUE_STATE_VW ビューを作成する

このステップでは、WLM_QUEUE_STATE_VW というビューを作成します。このビューは、次のシステムテーブルから情報を返します。

- STV_WLM_CLASSIFICATION_CONFIG
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE

チュートリアル全体でこのビューを使用して、WLM 設定の変更後にキューがどうなるかをモニタリングします。次の表は WLM_QUEUE_STATE_VW ビューが返すデータについて説明しています。

列	説明
キュー	キューを表す行に関連付けられた番号。キュー番号によってデータベースでのキューの順序が決まります。
description	特定のユーザーグループでのみ使用できるか、特定のクエリグループでのみ使用できるか、またはあらゆるタイプのクエリで使用できるかを示す値。
slots	キューに割り当てられたスロットの数。
mem	キューに割り当てられているメモリの量 (スロットあたりの MB 単位)。
max_execution_time	クエリの実行が許可されている時間。これを過ぎるとクエリは終了します。
ユーザー_*	ユーザーグループに一致させるために、ワイルドカード文字の使用が WLM 設定で許可されるかどうかを示す値。

列	説明
query_*	クエリグループに一致させるために、ワイルドカード文字の使用が WLM 設定で許可されるかどうかを示す値。
queued	キューで処理を待機中のクエリの数。
executing	現在実行中のクエリの数。
executed	実行されたクエリの数。

WLM_QUEUE_STATE_VW ビューを作成するには

1. Amazon Redshift RSQL を開き、TICKIT サンプルデータベースに接続します。このデータベースがない場合は、「前提条件」を参照してください。
2. 次のクエリを実行して、WLM_QUEUE_STATE_VW ビューを作成します。

```
create view WLM_QUEUE_STATE_VW as
select (config.service_class-5) as queue
, trim (class.condition) as description
, config.num_query_tasks as slots
, config.query_working_mem as mem
, config.max_execution_time as max_time
, config.user_group_wild_card as "user_*"
, config.query_group_wild_card as "query_*"
, state.num_queued_queries queued
, state.num_executing_queries executing
, state.num_executed_queries executed
from
STV_WLM_CLASSIFICATION_CONFIG class,
STV_WLM_SERVICE_CLASS_CONFIG config,
STV_WLM_SERVICE_CLASS_STATE state
where
class.action_service_class = config.service_class
and class.action_service_class = state.service_class
and config.service_class > 4
order by config.service_class;
```

3. 次のクエリを実行して、ビューに含まれる情報を表示します。

```
select * from wlm_queue_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (querytype:any)                           |     5 | 836 |        0 |  false |
 false   |      0 |         1 |      160
```

ステップ 2: WLM_QUERY_STATE_VW ビューを作成する

このステップでは、WLM_QUERY_STATE_VW というビューを作成します。このビューは STV_WLM_QUERY_STATE システムテーブルから情報を返します。

チュートリアル全体でこのビューを使用して、実行中のクエリをモニタリングします。次の表は WLM_QUERY_STATE_VW ビューが返すデータについて説明しています。

列	説明
query	クエリ ID。
キュー	キューの数。
slot_count	クエリに割り当てられたスロットの数。
start_time	クエリが開始された時刻。
state	実行などのクエリの状態。
queue_time	クエリがキューに入ってからの時間 (マイクロ秒)。
exec_time	クエリが実行されている時間 (マイクロ秒)。

WLM_QUERY_STATE_VW ビューを作成するには

1. RSQL で次のクエリを実行して、WLM_QUERY_STATE_VW ビューを作成します。

```
create view WLM_QUERY_STATE_VW as
select query, (service_class-5) as queue, slot_count, trim(wlm_start_time) as
 start_time, trim(state) as state, trim(queue_time) as queue_time, trim(exec_time) as
 exec_time
from stv_wlm_query_state;
```

2. 次のクエリを実行して、ビューに含まれる情報を表示します。

```
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 1249 |     1 |          1 | 2014-09-24 22:19:16 | Executing | 0          | 516
```

ステップ 3: テストクエリを実行する

このステップでは、RSQL で複数の接続からクエリを実行し、システムテーブルを確認して、クエリがどのようにルーティングされて処理されたかを判断します。

このステップでは、RSQL ウィンドウを 2 つ開いておく必要があります。

- RSQL ウィンドウ 1 では、このチュートリアルで既に作成したビューを使用して、キューとクエリの状態をモニタリングするクエリを実行します。
- RSQL ウィンドウ 2 では、RSQL ウィンドウ 1 に表示される結果を変更する実行時間が長いクエリを実行します。

テストクエリを実行するには

1. RSQL ウィンドウを 2 つ開きます。既にウィンドウを 1 つ開いている場合は、2 つ目のウィンドウを開くだけでかまいません。どちらの接続にも同じユーザーアカウントを使用できます。
2. RSQL ウィンドウ 1 で、次のクエリを実行します。

```
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 1258 |     1 |          1 | 2014-09-24 22:21:03 | Executing | 0          | 549
```

このクエリは自己参照的な結果を返します。現在実行中のクエリはこのビューからの SELECT ステートメントです。このビューに対するクエリは少なくとも 1 つの結果を常に返します。次のステップで実行時間が長いクエリを開始した後に表示される結果とこの結果を比較します。

3. RSQL ウィンドウ 2 で、TICKIT サンプルデータベースからクエリを実行します。このクエリは約 1 分間実行されるため、その間に WLM_QUEUE_STATE_VW ビューと以前に作成した WLM_QUERY_STATE_VW ビューの結果を確認できます。場合によっては、クエリの実行時間が短くて両方のビューに対してクエリを実行できないことがあります。このような場合は、`l.listid` でフィルターの値を増やし、実行時間を延長できます。

> **Note**
>
> クエリの実行時間を短縮し、システムパフォーマンスを向上させるために、Amazon Redshift は特定の種類のクエリの結果をリーダーノード上のメモリにキャッシュします。結果のキャッシュが有効になると、その後のクエリはより高速に実行されます。クエリが高速で実行されることを防ぐには、現行のセッションで結果のキャッシュを無効にします。

現在のセッションに対する結果のキャッシュをオフにするには、以下にあるとおり、[enable_result_cache_for_session](#) パラメータを `off` に設定します。

```
set enable_result_cache_for_session to off;
```

RSQL ウィンドウ 2 で、次のクエリを実行します。

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <
 100000;
```

4. RSQL ウィンドウ 1 で、WLM_QUEUE_STATE_VW と WLM_QUERY_STATE_VW をクエリし、その結果を以前の結果と比較します。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+------------------------------------------+-------+-----+----------+--------
+---------+--------+----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (querytype:any)                           |     5 | 836 |        0 |  false |
 false   |      0 |         2 |      163

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+--------------------+-----------+-----------
+-----------
 1267 |     1 |          1 | 2014-09-24 22:22:30 | Executing | 0          | 684
 1265 |     1 |          1 | 2014-09-24 22:22:36 | Executing | 0          | 4080859
```

以前のクエリとこのステップでの結果との間には次の違いがあることに注意してください。

- 現在は WLM_QUERY_STATE_VW に 2 行あります。1 つの結果は、このビューに対して SELECT 操作を実行する自己参照的クエリです。2 つ目の結果は、以前のステップの長時間実行されるクエリです。
- WLM_QUEUE_STATE_VW の executing 列の値は、1 から 2 に増えました。この列エントリは、キューで 2 つのクエリが実行中であることを意味します。
- executed 列の値は、キューのクエリを実行するたびに増加します。

WLM_QUEUE_STATE_VW ビューは、キューの全体像と各キューで処理中のクエリの数を把握するのに便利です。WLM_QUERY_STATE_VW ビューは、現在実行中の個々のクエリについてその詳細を把握するのに便利です。

セクション 2: WLM のクエリキュー設定の変更

キューのデフォルトの仕組みが理解できたため、次は手動 WLM を使用してクエリキューを設定する方法について説明します。このセクションでは、クラスターの新しいパラメータグループを作成して設定します。ユーザーキューを 2 つ追加で作成して、クエリのユーザーグループまたはクエリグループラベルに基づいてクエリを受け付けるように、これらのキューを設定します。この 2 つのキューのどちらにもルーティングされないクエリは、実行時にデフォルトキューにルーティングされます。

パラメータグループの手動 WLM 設定を作成するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで、[Configurations] (設定) を選択し、次に [Workload management] (ワークロード管理) を選択して [Workload management] (ワークロード管理) ページを表示します。
3. [作成] を選択して [パラメータグループの作成] ウィンドウを表示します。
4. [Parameter group name] (パラメータグループ名) と[Description] (説明)の両方で **WLMTutorial** を入力し、[Create] (作成) をクリックしてパラメータグループを作成します。

> **Note**
>
> [パラメータグループ名] は作成時にすべて小文字に変換されます。

5. [ワークロード管理] ページで、パラメータグループ **wlmtutorial** を選択し、[パラメータ] と [ワークロード管理] のタブのある詳細ページを表示します。
6. [ワークロード管理] タブを開いていることを確認し、[Switch WLM mode (WLM モードの切り替え)] を選択して [Concurrency settings (同時実行設定)] ウィンドウを表示します。
7. [Manual WLM (手動 WLM)] を選択してから [保存] を選択し、手動 WLM に切り替えます。
8. [Edit workload queues (ワークロードキューの編集)] を選択します。
9. [キューの追加] を 2 度選択して、2 つのキューを追加します。現在 [キュー 1]、[キュー 2]、[デフォルトキュー] の 3つのキューがある状態です。
10. それぞれのキューの情報を次のように入力します。

    • [Queue 1] (キュー 1) では、[Memory (%)] (メモリ (%)) に **30**、[Concurrency on main] (メインの同時実行性)に **2**、[Query groups] (クエリグループ) に **test** を入力します。その他の設定はデフォルト値のままにしておきます。

- [Queue 2] (キュー 2) では、[Memory (%)] (メモリ (%)) に **40**、[Concurrency on main] (メインの同時実行性) に **3**、[User groups] (ユーザーグループ) に **admin** を入力します。その他の設定はデフォルト値のままにしておきます。
- [デフォルトキュー] には変更を加えないでください。WLM は、未割り当てのメモリをデフォルトキューに割り当てます。

11. 設定を保存するには [保存] を選択します。

次に、手動 WLM 設定を持つパラメータグループをクラスターに関連付けます。

パラメータグループをクラスターの手動 WLM 設定に関連付けるには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューから [Clusters] (クラスター) を選択し、次に [Clusters] (クラスター) を選択してクラスターのリストを表示します。
3. クラスターの詳細を表示するには、`examplecluster` などのクラスターを選択します。次に、[Properties] (プロパティ) タブを選択して、そのクラスターのプロパティを表示します。
4. [Database configurations] (データベース構成) セクションで、[Edit] (編集)、[Edit parameter group] (パラメータグループの編集) を選択して、パラメータグループのウィンドウを表示します。
5. [Parameter groups] (パラメータグループ) で、以前に作成した **wlmtutorial** パラメータグループを選択します。
6. [Save changes] (変更を保存) を選択して、パラメータグループを関連付けます。

   クラスターは変更されたパラメータグループで変更されます。ただし、変更をデータベースにも適用するには、クラスターを再起動する必要があります。
7. クラスターを選択してから、[Actions] (アクション) の [Reboot] (再起動) を選択します。

クラスターが再起動されると、ステータスは [利用可能] に戻ります。

セクション 3: ユーザーグループとクエリグループに基づいてクエリをキューにルーティング

クラスターを新しいパラメータグループに関連付けて WLM を設定しました。次に、いくつかのクエリを実行して Amazon Redshift がどのようにクエリをキューにルーティングして処理するかを確認します。

ステップ 1: データベースのクエリキュー設定を表示する

まず、データベースに WLM が正常に設定されていることを確認します。

クエリキュー設定を表示するには

1. RSQL を開き、次のクエリを実行します。クエリは「ステップ 1: WLM_QUEUE_STATE_VW ビューを作成する」で作成した WLM_QUEUE_STATE_VW ビューを使用します。クラスターを再起動する前にセッションをデータベースに接続済みである場合は、再接続する必要があります。

```
select * from wlm_queue_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        0
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |        0
```

この結果を、「ステップ 1: WLM_QUEUE_STATE_VW ビューを作成する」で受け取った結果と比較します。キューが 2 つ追加されていることがわかります。キュー 1 は test クエリグループのキューになり、キュー 2 は admin ユーザーグループのキューになっています。

キュー 3 はデフォルトキューになっています。リストの最後のキューは常にデフォルトキューです。これは、クエリでユーザーグループやクエリグループが指定されていない場合、デフォルトでクエリがルーティングされる先のキューです。

2. 次のクエリを実行して、クエリがキュー 3 で実行されることを確認します。

```
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2144 |     3 |          1 | 2014-09-24 23:49:59 | Executing | 0          | 550430
```

ステップ 2: クエリグループキューを使ってクエリを実行する

クエリグループキューを使ってクエリを実行するには

1. 次のクエリを実行して、クエリを `test` クエリグループにルーティングします。

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. 他の RSQL ウィンドウから、次のクエリを実行します。

```
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2168 |     1 |          1 | 2014-09-24 23:54:18 | Executing | 0          | 6343309
 2170 |     3 |          1 | 2014-09-24 23:54:24 | Executing | 0          | 847
```

クエリは、test クエリグループ (今はキュー 1) にルーティングされました。

3. キューの状態表示ですべてを選択します。

```
select * from wlm_queue_state_vw;
```

次のような結果が表示されます。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
```

```
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |          0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |          1 |        0
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |          0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |          1 |        0
```

4. 今度は、クエリグループをリセットして、実行時間が長いクエリを再び実行します。

```
reset query_group;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

5. ビューに対するクエリを実行して、結果を確認します。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |          0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |          0 |        1
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |          0 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |          2 |        5

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2186 |     3 |          1 | 2014-09-24 23:57:52 | Executing | 0          | 649
 2184 |     3 |          1 | 2014-09-24 23:57:48 | Executing | 0          | 4137349
```

結果として、クエリが今は再びキュー 3 で実行されています。

ステップ 3: データベースユーザーとグループを作成する

このキューでクエリを実行する前に、データベースにユーザーグループを作成して、このユーザーグループにユーザーを追加する必要があります。次に、新しいユーザーの認証情報を使って RSQL にログインして、クエリを実行します。データベースユーザーを作成するには、管理ユーザーのようなスーパーユーザーとしてクエリを実行する必要があります。

新しいデータベースユーザーとユーザーグループを作成するには

1. RSQL ウィンドウで次のコマンドを実行して、データベースに `adminwlm` という名前の新しいデータベースユーザーを作成します。

```
create user adminwlm createuser password '123Admin';
```

2. 続いて、次のコマンドを実行して、新しいユーザーグループを作成し、新しい `adminwlm` ユーザーをそのユーザーグループに追加します。

```
create group admin;
alter group admin add user adminwlm;
```

ステップ 4: ユーザーグループキューを使ってクエリを実行する

次に、クエリを実行し、それをユーザーグループキューにルーティングします。これを行うのは、実行するクエリのタイプを処理するように設定されたキューにクエリをルーティングする場合です。

ユーザーグループキューを使ってクエリを実行するには

1. RSQL ウィンドウ 2 で、次のクエリを実行して、`adminwlm` アカウントに切り替え、そのユーザーとしてクエリを実行します。

```
set session authorization 'adminwlm';
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. RSQL ウィンドウ 1 で、次のクエリを実行して、クエリがルーティングされるクエリキューを確認します。

```
select * from wlm_query_state_vw;
select * from wlm_queue_state_vw;
```

   結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        1
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         1 |        0
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |         1 |        8

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2202 |     2 |          1 | 2014-09-25 00:01:38 | Executing | 0          | 4885796
 2204 |     3 |          1 | 2014-09-25 00:01:43 | Executing | 0          | 650
```

このクエリが実行されたキューはキュー 2 (admin ユーザーキュー) です。このユーザーとしてログインしてクエリを実行すると、別のクエリグループを指定しない限り、クエリは常にキュー 2 で実行されます。選択されるキューは、キューの割り当てルールによって異なります。詳細については、「WLM キュー割り当てルール」を参照してください。

3. 次に、RSQL ウィンドウ 2.から次のクエリを実行します。

```
set query_group to test;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. RSQL ウィンドウ 1 で、次のクエリを実行して、クエリがルーティングされるクエリキューを確認します。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+-----------+----------
```

```
   0 | (super user) and (query group: superuser) |     1 | 357 |         0 |  false |
 false   |      0 |          0 |        0
   1 | (query group: test)                       |     2 | 627 |         0 |  false |
 false   |      0 |          1 |        1
   2 | (suser group: admin)                      |     3 | 557 |         0 |  false |
 false   |      0 |          0 |        1
   3 | (querytype:any)                           |     5 | 250 |         0 |  false |
 false   |      0 |          1 |       10

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2218 |     1 |          1 | 2014-09-25 00:04:30 | Executing | 0          | 4819666
 2220 |     3 |          1 | 2014-09-25 00:04:35 | Executing | 0          | 685
```

5. 処理が完了したら、クエリグループをリセットします。

```
reset query_group;
```

セクション 4: wlm_query_slot_count を使用してキューの同時実行レベルを一時的に変更

ときどき、特定のクエリのために一時的にリソースが余分に必要になることがあります。その場合は、wlm_query_slot_count 設定を使って、クエリキューでスロットが割り当てられる方法を一時的に変更することができます。スロットは、クエリを処理するために使用されるメモリと CPU の単位です。データベースで VACUUM 操作を実行する場合など、クラスターのリソースを大量に使用するクエリをたまに実行する場合に、スロット数を一時的に変更できます。

クエリのタイプによっては、ユーザーが wlm_query_slot_count を設定することが必要になる場合があります。そのような場合は、WLM 設定を調整し、クエリのニーズにより適したキューをユーザーに提供することを検討します。スロット数を使用して同時実行レベルを一時的に変更する方法の詳細については、「wlm_query_slot_count」を参照してください。

ステップ 1: wlm_query_slot_count を使用して同時実行レベルを一時的に変更する

このチュートリアルの目的に合わせて、実行時間が長い同じ SELECT クエリを実行します。そのクエリは `adminwlm` ユーザーとして実行し、wlm_query_slot_count を使ってクエリで使用可能なスロット数を増やします。

wlm_query_slot_count を使用して同時実行レベルを一時的に変更するには

1. クエリの制限を増やして、WLM_QUERY_STATE_VW ビューをクエリして結果を確認する時間を十分に確保します。

```
set wlm_query_slot_count to 3;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

2. ここで、管理者ユーザーを使用して WLM_QUERY_STATE_VW にクエリを実行し、クエリの動作を確認します。

```
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2240 |     2 |          1 | 2014-09-25 00:08:45 | Executing | 0          | 3731414
 2242 |     3 |          1 | 2014-09-25 00:08:49 | Executing | 0          | 596
```

クエリのスロット数が 3 であることがわかります。この数は、クエリが 3 つのスロットをすべて使ってクエリを処理していること、キューのリソースがすべてそのクエリに割り当てられていることを意味します。

3. 今度は、次のクエリを実行します。

```
select * from WLM_QUEUE_STATE_VW;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |         0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      0 |         0 |        4
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |         1 |        3
```

```
    3 | (querytype:any)                           |     5 | 250 |          0 |   false |
 false   |      0 |           1 |        25
```

wlm_query_slot_count 設定は、現在のセッションに対してのみ有効です。そのセッションが期限切れになった場合、または、別のユーザーがクエリを実行する場合は、WLM 設定が使用されます。

4. スロット数をリセットして、テストを再実行します。

```
reset wlm_query_slot_count;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |          0 |   false |
 false   |      0 |           0 |         0
    1 | (query group: test)                       |     2 | 627 |          0 |   false |
 false   |      0 |           0 |         2
    2 | (suser group: admin)                      |     3 | 557 |          0 |   false |
 false   |      0 |           1 |         2
    3 | (querytype:any)                           |     5 | 250 |          0 |   false |
 false   |      0 |           1 |        14

query | queue | slot_count | start_time          | state     | queue_time | exec_time
------+-------+------------+---------------------+-----------+------------
+-----------
 2260 |     2 |          1 | 2014-09-25 00:12:11 | Executing | 0          | 4042618
 2262 |     3 |          1 | 2014-09-25 00:12:15 | Executing | 0          | 680
```

ステップ 2: 別のセッションからクエリを実行する

次に、別のセッションからクエリを実行します。

別のセッションからクエリを実行するには

1. RSQL ウィンドウ 1 および 2 で、次を実行して、テストクエリグループを使用します。

```
set query_group to test;
```

2. RSQL ウィンドウ 1 で、実行時間が長い次のクエリを実行します。

```
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

3. RSQL ウィンドウ 1 で実行時間が長いクエリがまだ進行中の間に、以下を実行します。これらのコマンドは、キューのすべてのスロットを使用するようにスロット数を増やした後で、実行時間が長いクエリの実行を開始します。

```
set wlm_query_slot_count to 2;
select avg(l.priceperticket*s.qtysold) from listing l, sales s where l.listid <40000;
```

4. 3 つ目の RSQL ウィンドウを開いて、ビューをクエリし、結果を確認します。

```
select * from wlm_queue_state_vw;
select * from wlm_query_state_vw;
```

結果の例は次のとおりです。

```
query | description                               | slots | mem | max_time | user_* |
 query_* | queued | executing | executed
------+-------------------------------------------+-------+-----+----------+--------
+---------+--------+----------+----------
    0 | (super user) and (query group: superuser) |     1 | 357 |        0 |  false |
 false   |      0 |        0 |        0
    1 | (query group: test)                       |     2 | 627 |        0 |  false |
 false   |      1 |        1 |        2
    2 | (suser group: admin)                      |     3 | 557 |        0 |  false |
 false   |      0 |        0 |        3
    3 | (querytype:any)                           |     5 | 250 |        0 |  false |
 false   |      0 |        1 |       18

query | queue | slot_count | start_time          | state          | queue_time |
 exec_time
------+-------+------------+---------------------+----------------+------------
+-----------
 2286 |     1 |          2 | 2014-09-25 00:16:48 | QueuedWaiting  | 3758950    | 0
 2282 |     1 |          1 | 2014-09-25 00:16:33 | Executing      | 0          |
 19335850
```

```
  2288 |     3 |           1 | 2014-09-25 00:16:52 | Executing      | 0            | 666
```

最初のクエリではキュー 1 に割り当てられたスロットの 1 つを使用してクエリを実行していることに注意してください。さらに、キュー内で待機しているキューが 1 つあることにも注意してください (queued が 1、state が QueuedWaiting)。最初のクエリが完了すると、2 つ目のクエリの実行が開始されます。このように実行されるのは、両方のクエリが test クエリグループにルーティングされていて、2 つ目のクエリは十分なスロット数が使用可能になるまで処理の開始を待機する必要があるためです。

セクション 5: リソースのクリーンアップ

クラスターが実行されている限り料金が発生し続けます。このチュートリアルを完了したら、Amazon Redshift 入門ガイドの「ステップ 8: 環境をリセットする」のステップに従って、環境を以前の状態に戻します。

WLM の詳細については、「ワークロード管理」を参照してください。

同時実行スケーリング

同時実行スケーリング機能を使用すると、一貫した高速のクエリパフォーマンスで、数千の同時ユーザーと同時クエリをサポートできます。同時実行スケーリングが有効になっている場合、Amazon Redshift は自動的に新たなクラスターキャパシティーを追加し、読み取りと書き込み両方でクエリの増加に対応します。クエリをメインクラスターと同時実行スケーリングクラスターのどちらで実行しても、ユーザーには最新のデータが表示されます。

WLM キューを設定することで、どのクエリを同時実行スケーリングクラスターに送信するかを管理できます。同時実行スケーリングを有効にすると、対象となるクエリはキュー内に待機することなく、同時実行スケーリングクラスターに送信されるようになります。

同時実行スケーリングクラスターは、実際に実行した時間分のみ課金されます。料金の発生する仕組みや最低料金など、料金の詳細については、「同時実行スケーリングの料金」を参照してください。

トピック

- 同時実行スケーリング機能
- 同時実行スケーリングに関する制限
- 同時実行スケーリングの AWS リージョン
- 同時実行スケーリングの候補

- 同時実行スケーリングキューの設定
- 同時実行スケーリングのモニタリング
- 同時実行スケーリングシステムビュー

同時実行スケーリング機能

WLM キューで同時実行スケーリングを有効にすると、このスケーリングは、ダッシュボードクエリなどの読み取りオペレーションのために機能します。また、データの取り込みや処理のためのステートメントなど、一般的に使用される書き込みオペレーションにおいても機能します。

書き込み操作のための同時実行スケーリング機能

同時実行スケーリングは、抽出、変換、ロード (ETL) ステートメントなど、頻繁に使用される書き込みオペレーションをサポートしています。書き込み操作用の同時実行スケーリングは、多数のリクエストを受信しているクラスターに、一貫した応答時間を維持させたい場合に特に有用です。これにより、メインクラスター上のリソースについて競合を起こしている、書き込み操作のスループットが向上します。

同時実行スケーリングでは、COPY、INSERT、DELETE、UPDATE、CREATE TABLE AS (CTAS) の各ステートメントをサポートしています。さらに、同時実行スケーリングは、集計を使用しない MV のマテリアライズドビュー更新をサポートしています。他のデータ操作言語 (DML) ステートメントとデータ定義言語 (DDL) ステートメントはサポートされていません。サポートされていない WRITE ステートメント (CREATE without TABLE AS など) が、サポートされている WRITE ステートメントの前に明示的なトランザクションに含まれる場合、その WRITE ステートメントは同時実行スケーリングクラスターでは実行されません。

同時実行スケーリングのクレジットを計上すると、このクレジットは、読み取りと書き込み両方のオペレーションに適用されます。

同時実行スケーリングに関する制限

Amazon Redshift で同時実行スケーリングを使用する際の制限事項を以下に示します。

- インターリーブソートキーを使用するテーブルに対するクエリはサポートされていません。
- 一時テーブルに対するクエリはサポートされていません。
- 制限のあるネットワークまたは仮想プライベートクラウド (VPC) 構成で保護されている、外部リソースにアクセスするクエリはサポートされません。
- Python のユーザー定義関数 (UDF) と Lambda UDF を含むクエリはサポートされていません。

- システムテーブル、PostgreSQL のカタログテーブル、またはバックアップ用ではないテーブルにアクセスするクエリはサポートされていません。
- 制限付き IAM ポリシーのアクセス許可が設定されている場合、外部リソースにアクセスする COPY クエリや UNLOAD クエリはサポートされません。これには、Amazon S3 バケットや DynamoDB テーブルなどのリソース、またはソースに適用されるアクセス許可が含まれます。IAM ソースの例は以下のとおりです。
  - `aws:sourceVpc` – ソース VPC。
  - `aws:sourceVpce` – ソース VPC エンドポイント。
  - `aws:sourceIp` – ソース IP アドレス。

  場合によっては、リソースまたはソースのいずれかを制限するアクセス許可を削除して、リソースにアクセスする COPY クエリや UNLOAD クエリが同時実行スケーリングクラスターに送信されるようにする必要があります。

  リソースポリシーの詳細については、「AWS Identity and Access Management ユーザーガイド」の「ポリシータイプ」と、「バケットポリシーを使用した VPC エンドポイントからのアクセスコントロール」を参照してください。
- 書き込みオペレーションの Amazon Redshift の同時実行スケーリングでは、CREATE TABLE や ALTER TABLE などの DDL (書き込み) オペレーションはサポートしていません。
- COPY コマンドでの ANALYZE の使用はサポートされていません。
- DISTSTYLE に ALL が設定されているターゲットテーブルに対する書き込みオペレーションは、サポートしていません。
- 以下のファイル形式からの COPY はサポートされていません。
  - Parquet
  - ORC
- ID 列を持つテーブルに対する書き込みオペレーションはサポートしていません。
- Amazon Redshift は、Amazon Redshift RA3 ノードのみで、書き込みオペレーションの同時実行スケーリングをサポートしています。他のノードタイプでは、書き込みオペレーションの同時実行スケーリングはサポートされていません。

同時実行スケーリングの AWS リージョン

Amazon Redshift では、同時実行スケーリングを使用して、Redshift クラスター全体の同時実行ワークロードの需要を管理できます。このトピックでは、Amazon Redshift で同時実行スケーリングを使用できるリージョンについて詳しく説明します。

同時実行スケーリングは、以下の AWS リージョンでご利用になれます。

- 米国東部 (バージニア北部) リージョン (us-east-1)
- 米国東部 (オハイオ) リージョン (us-east-2)
- 米国西部 (北カリフォルニア) リージョン (us-west-1)
- 米国西部 (オレゴン) リージョン (us-west-2)
- アジアパシフィック (ムンバイ) リージョン (ap-south-1)
- アジアパシフィック (ソウル) リージョン (ap-northeast-2)
- アジアパシフィック (シンガポール) リージョン (ap-southeast-1)
- アジアパシフィック (シドニー) リージョン (ap-southeast-2)
- アジアパシフィック (東京) リージョン (ap-northeast-1)
- カナダ (中部) リージョン (ca-central-1)
- 中国 (北京) リージョン (cn-north-1)
- 中国 (寧夏) リージョン (cn-northwest-1)
- 欧州 (フランクフルト) リージョン (eu-central-1)
- 欧州 (アイルランド) リージョン (eu-west-1)
- 欧州 (ロンドン) リージョン (eu-west-2)
- 欧州 (パリ) リージョン (eu-west-3)
- 欧州 (ストックホルム) リージョン (eu-north-1)
- 欧州 (チューリッヒ) リージョン (eu-central-2)
- 欧州 (スペイン) リージョン (eu-south-2)
- 南米 (サンパウロ) リージョン (sa-east-1)
- AWS GovCloud (米国東部)

同時実行スケーリングの候補

Amazon Redshift では、クエリ処理をスケールアウトして、同時実行クエリを高速化できます。次のトピックでは、Amazon Redshift で同時実行スケーリングにルーティングするクエリを決定するために使用する基準について説明します。

メインクラスターが次の要件を満たしている場合にのみ、クエリは同時実行クラスターにルーティングされます。

- EC2-VPC プラットフォーム。
- ノードタイプは dc2.8xlarge、dc2.large、ra3.large、ra3.xlplus、ra3.4xlarge、ra3.16xlarge のいずれかである必要があります。書き込みオペレーションの同時実行スケーリングは、Amazon Redshift RA3 ノードのみでサポートされています。
- ノードタイプとして ra3.xlplus、ra3.4xlarge、または ra3.16xlarge を使用するクラスターの最大コンピューティングノード数は 32 個です。また、メインクラスターのノード数は、クラスターの元の作成時には 32 ノード以下である必要があります。例えば、クラスターに現在 20 個のノードがあるが、元々 40 個で作成された場合は、同時実行スケーリングの要件を満たしません。逆に、DC2 クラスターに現在 40 個のノードがあるが、元々 20 個で作成されている場合は、同時実行スケーリングの要件を満たします。
- シングルノードクラスターではありません。

同時実行スケーリングキューの設定

Amazon Redshift では、同時実行スケーリングを設定することで、同時実行リソースとシステムリソースを管理できます。同時実行スケーリングキューを使用すると、同時に実行できるクエリまたはユーザーセッションの数を制限できます。次のセクションでは、Amazon Redshift で同時実行スケーリングキューを有効にして、同時クエリとユーザーセッションを効果的に処理する方法について説明します。

クエリを同時実行スケーリングクラスターにルーティングするには、ワークロードマネージャ (WLM) キューで同時実行スケーリングを有効化します。キューの同時実行スケーリングを有効にするには、[同時実行スケーリングモード] の値に [自動] を設定します。

同時実行スケーリングが有効なキューにルーティングされたクエリの数がキューの同時実行数を超えると、キャパシティが手動で設定されていても、自動で決定されても、対象クエリが同時実行スケーリングクラスターに送信されます。メインクラスターでキュースロットが利用可能になると、クエリはメインクラスターにルーティングされ、実行されます。他の WLM キューと同様に、ユーザーグループに基づいて、またはクエリグループラベルをクエリに付けることで、あるいは [キューへのクエリの割り当て] で定義した照合条件に従って、クエリを同時実行スケーリングキューにルーティングします。WLM クエリモニタリングルール を定義してクエリをルーティングすることもできます。例えば、所要時間が 5 秒を超えるすべてのクエリを同時実行スケーリングキューにルーティングできます。キューイングの動作は、自動 WLM と手動 WLM のどちらを使用しているかによって異なる可能性があることに注意してください。詳細については、「自動 WLM の実装」または「手動 WLM の実装」を参照してください。

同時実行スケーリングクラスターのデフォルト数は 1 です。使用できる同時実行スケーリングクラスターの数は、max_concurrency_scaling_clusters によって制御されます。

同時実行スケーリングのモニタリング

Amazon Redshift では、同時実行スケーリングをモニタリングおよび管理して、データウェアハウスワークロードのパフォーマンスとコスト効率を最適化できます。同時実行スケーリングを使用すると、Amazon Redshift はワークロードの需要が増加すると自動的にクラスター容量を追加し、需要が減少するとその容量を削除します。次のセクションでは、Amazon Redshift クラスターの同時実行スケーリングのモニタリングに関するガイダンスを提供します。

メインクラスターと同時実行スケーリングクラスターのどちらでクエリが実行されているかを確認するには、Amazon Redshift コンソールをで [クラスター] に移動してクラスターを選択します。次に、[クエリのモニタリング] タブと [ワークロードの同時実行] を選択すると、実行中のクエリとキューに入っているクエリに関する情報が表示されます。

実行時間を確認するには、STL_QUERY テーブルをクエリし、concurrency_scaling_status 列でフィルタリングします。次のクエリは、同時実行スケーリングクラスターで実行されたクエリとメインクラスターで実行されたクエリのキュー時間と実行時間を比較します。

```
SELECT w.service_class AS queue
, CASE WHEN q.concurrency_scaling_status = 1 THEN 'concurrency scaling cluster' ELSE
 'main cluster' END as concurrency_scaling_status
, COUNT( * ) AS queries
, SUM( q.aborted ) AS aborted
, SUM( ROUND( total_queue_time::NUMERIC / 1000000,2) ) AS queue_secs
, SUM( ROUND( total_exec_time::NUMERIC / 1000000,2) ) AS exec_secs
FROM stl_query q
JOIN stl_wlm_query w
USING (userid,query)
WHERE q.userid > 1
AND q.starttime > '2019-01-04 16:38:00'
AND q.endtime < '2019-01-04 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

必要に応じて、starttime および endtime の値を調整します。

同時実行スケーリングシステムビュー

Amazon Redshift では、同時実行スケーリングシステムビューを使用して、クラスター内の同時実行スケーリングアクティビティをモニタリングおよび管理できます。次のセクションでは、Amazon Redshift 環境で同時実行スケーリングを効果的に活用するために、これらのシステムビューをクエリし、結果を解釈する方法について説明します。

プレフィックス SVCS のある一連のシステムビューは、メインクラスターと同時実行スケーリングクラスターの両方におけるクエリに関するシステムログテーブルの詳細を提供します。

以下のビューには、対応する STL ビューや SVL ビューと同様の情報があります。

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_STREAM_SEGS

以下のビューは同時実行スケーリングに固有のものです。

- SVCS_CONCURRENCY_SCALING_USAGE

同時実行スケーリングの詳細については、「Amazon Redshift 管理ガイド」で次のトピックを参照してください。

- 同時実行スケーリングデータの表示
- クエリ実行中のクラスターパフォーマンスの表示
- クエリの詳細の表示

ショートクエリアクセラレーション

ショートクエリアクセラレーション (SQA) は、実行時間が短い一部のクエリを、実行時間が長いクエリよりも優先します。SQA では実行時間が短いクエリを専用領域で実行します。このため SQA クエリは、実行時間が長いクエリをキューで待機するよう強制されません。SQA は、実行時間が短

く、ユーザー定義のキュー内にあるクエリのみを優先します。SQA によって実行時間が短いクエリの実行開始が早くなり、ユーザーへの結果表示も早くなります。

SQA を有効にすると、短いクエリの実行に割り当てられるワークロード管理 (WLM) キューを減らすことができます。さらに、キュー内のスロットに対する実行時間が長いクエリとショートクエリの競合が不要になるため、WLM キューが使用するクエリスロットの数を少なく設定できます。同時実行数が減るとクエリのスループットが向上し、大部分のワークロードに関するシステム全体のパフォーマンスも向上します。

CREATE TABLE AS (CTAS) ステートメントと読み取り専用クエリ (SELECT ステートメントなど) は SQA の対象です。

Amazon Redshift は、機械学習アルゴリズムを使用して対象のクエリを 1 つひとつ分析し、クエリの実行時間を予測します。デフォルトでは、WLM は、クラスターのワークロードの分析に基づいて、SQA 最大実行時間の値を動的に割り当てます。または、固定値 (1～20 秒) を指定します。クエリの予測実行時間が、定義済みまたは動的に割り当てられた SQA の最大ランタイムよりも短く、クエリが WLM キューで待機している場合、SQA はクエリを WLM キューから分離し、優先的に実行するようにスケジュールします。クエリの実行時間が SQA の最大実行時間より長い場合、WLM は、WLM キュー割り当てルールに基づき、最初の一致する WLM キューにクエリを移動します。クエリのパターンを SQA が学習するため、時間が経つほど予測精度は向上します。

SQA は、デフォルトのパラメータグループおよびすべての新しいパラメータグループに対してデフォルトで有効になっています。Amazon Redshift コンソールで SQA を無効にするには、パラメータグループの WLM 設定を編集し、[Enable short query acceleration (ショートクエリアクセラレーションの有効化)] を選択解除します。ベストプラクティスとして、システムパフォーマンス全体を最適な状態に維持するためには、WLM クエリのスロット数を 15 以下にすることをお勧めします。WLM 設定の変更については、「Amazon Redshift 管理ガイド」の「ワークロード管理の設定」を参照してください。

ショートクエリの最大実行時間

SQA を有効にすると、WLM は、ショートクエリの最大実行時間をデフォルトで動的に設定します。SQA の最大実行時間の動的設定を保持することをお勧めします。デフォルト設定を上書きするには、固定値 (1～20 秒) を指定します。

場合によっては、SQA の最大実行時間値に別の値を使用して、システムのパフォーマンスを向上させることを検討する場合があります。そのような場合は、ワークロードを分析して、実行時間の短いクエリの最大実行時間を見極めます。以下のクエリは、クエリの最大実行時間を約 70 パーセンタイル値で返します。

```
select least(greatest(percentile_cont(0.7)
within group (order by total_exec_time / 1000000) + 2, 2), 20)
from stl_wlm_query
where userid >= 100
and final_state = 'Completed';
```

ワークロードにとって有効な最大実行時間の値を特定したら、ワークロードに大幅な変更が発生しない限りは、その値を変更する必要はありません。

SQA のモニタリング

SQA が有効になっているかどうか確認するには、以下のクエリを実行します。クエリが 1 行を返した場合、SQA は有効です。

```
select * from stv_wlm_service_class_config
where service_class = 14;
```

次のクエリは、各クエリキュー (サービスクラス) を通過したクエリの数を示しています。また、平均実行時間、待機時間が発生しているクエリの数 (90 パーセンタイル値)、平均待機時間を示しています。SQA クエリはサービスクラス 14 で使用します。

```
select final_state, service_class, count(*), avg(total_exec_time),
percentile_cont(0.9) within group (order by total_queue_time), avg(total_queue_time)
from stl_wlm_query where userid >= 100 group by 1,2 order by 2,1;
```

SQA によって選択され正常に完了したクエリを特定するには、以下のクエリを実行します。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Completed'
order by b.query desc limit 5;
```

SQA で選択されたがタイムアウトしたクエリを特定するには、以下のクエリを実行します。

```
select a.queue_start_time, a.total_exec_time, label, trim(querytxt)
from stl_wlm_query a, stl_query b
where a.query = b.query and a.service_class = 14 and a.final_state = 'Evicted'
order by b.query desc limit 5;
```

削除されたクエリ、およびより一般的にはクエリに対して実行できるルールベースのアクションの詳細については、WLM クエリモニタリングルール を参照してください。

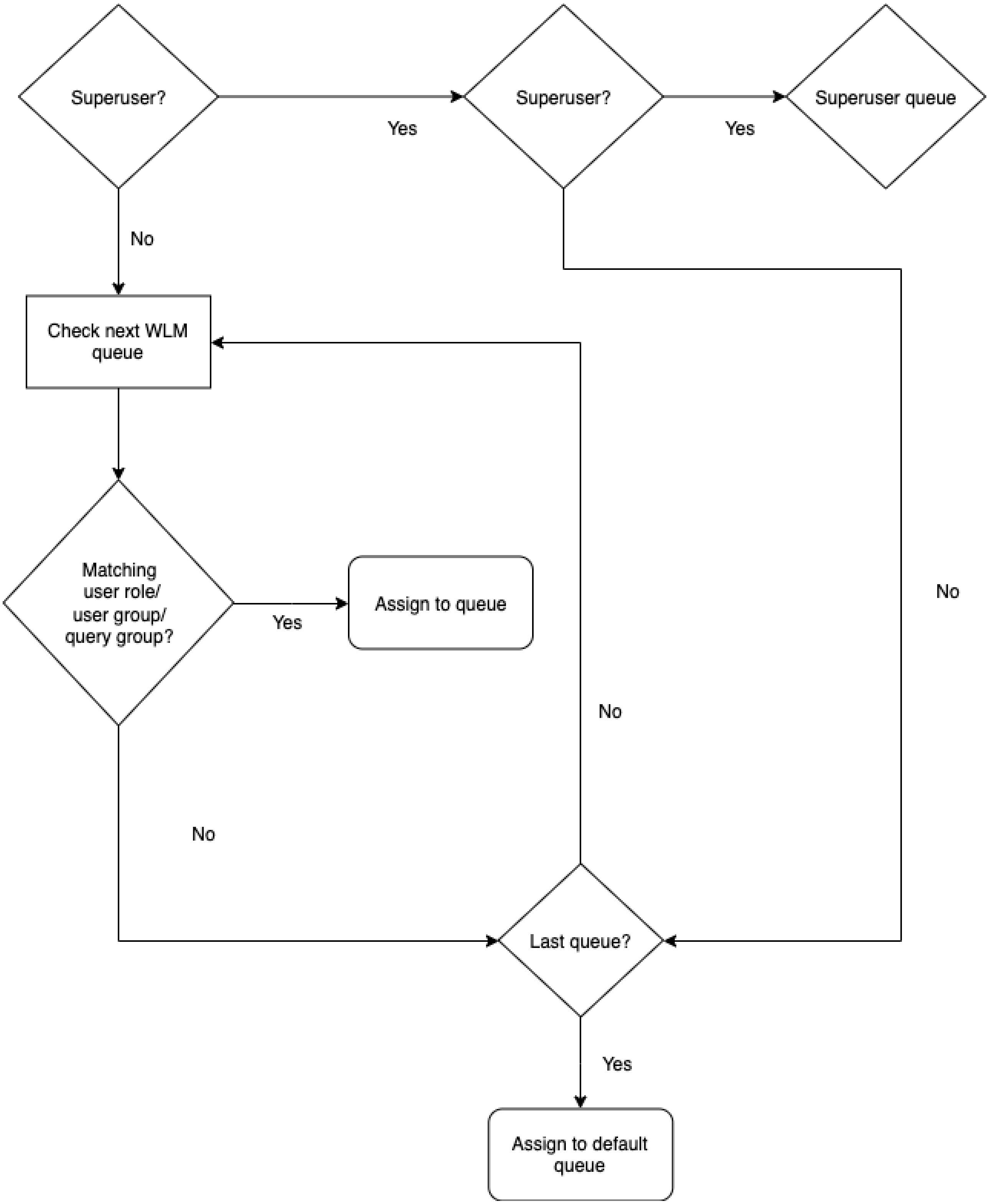
WLM キュー割り当てルール

Amazon Redshift では、ワークロード管理 (WLM) 設定でキュー割り当てルールを定義することで、メモリと CPU リソースのユーザークエリへの割り当てを制御できます。次のセクションでは、効率的なリソース配分を実現し、Amazon Redshift のさまざまなワークロードのサービスレベルアグリーメントを満たすための WLM キュー割り当てルールの作成と管理について説明します。

ユーザーがクエリを実行するときに、WLM は WLM キュー割り当てルールに基づいて、最初に一致するキューにクエリを割り当てます。

1. ユーザーがスーパーユーザーとしてログインし、superuser というラベルのクエリグループでクエリを実行すると、クエリはスーパーユーザーキューに割り当てられます。
2. ユーザーがロールの一部で、リストされているユーザーグループに属する場合、またはリストされているクエリグループ内でクエリを実行する場合、クエリは最初に一致するキューに割り当てられます。
3. クエリがいずれの条件にも一致しない場合、クエリは WLM 設定で最後のキューとして定義されているデフォルトキューに割り当てられます。

次の図にこれらのルールの仕組みを示します。

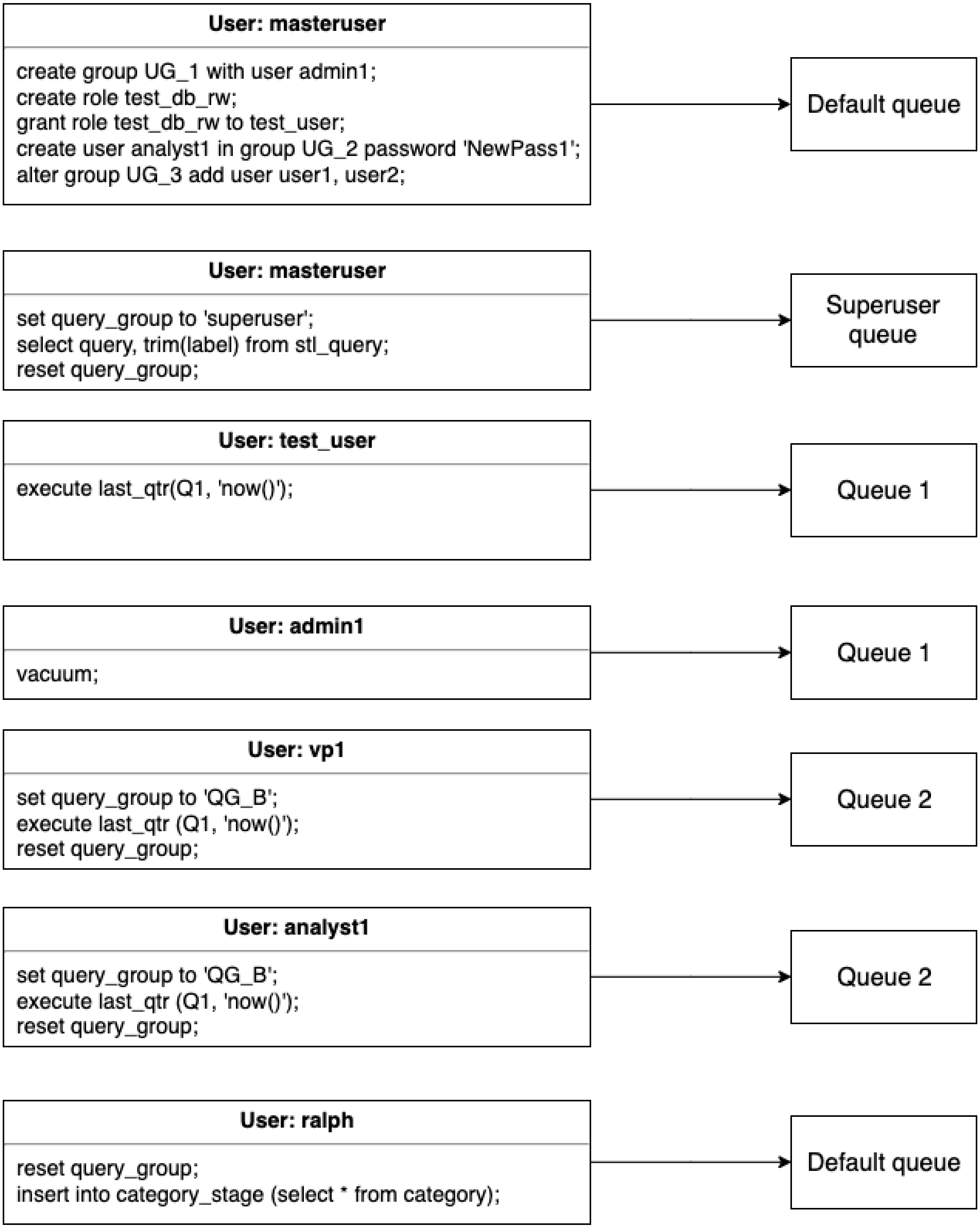


キュー割り当ての例

次の表に、superuser キューと 4 つのユーザー定義キューを持つ WLM 設定を示します。

キュー	Concurrency	ユーザーロール	ユーザーグループ	クエリグループ
スーパーユーザー	1			スーパーユーザー
1	5	test_db_rw	UG_1	
2	5			QG_B
3	5		UG_2	QG_C
デフォルト	5			

次の図は、ユーザーグループとクエリグループに従って、前のテーブルでクエリがキューに割り当てられる方法を示します。実行時にクエリをユーザーグループとクエリグループに割り当てる方法の詳細については、後の「キューへのクエリの割り当て」を参照してください。



この例では、WLM は次の割り当てを行います。

1. ステートメントの最初のセットでは、ユーザーをユーザーグループに割り当てるための 3 つの方法を示します。ステートメントは、ユーザー `adminuser` によって実行されます。このユーザーは、いずれかの WLM キューにリストされているユーザーグループのメンバーではありません。クエリグループは設定されないため、ステートメントはデフォルトキューにルーティングされます。
2. ユーザー `adminuser` はスーパーユーザーであり、クエリグループは `'superuser'` に設定されるため、クエリはスーパーユーザーキューに割り当てられます。
3. ユーザー `test_user` は、キュー 1 にリストされたロール `test_db_rw` が割り当てられるため、クエリはキュー 1 に割り当てられます。
4. ユーザー `admin1` はキュー 1 にリストされたユーザーグループのメンバーであるため、クエリはキュー 1 に割り当てられます。
5. ユーザー `vp1` は、リストされたユーザーグループのメンバーではありません。クエリグループは `'QG_B'` に設定されているため、クエリはキュー 2 に割り当てられます。
6. ユーザー `analyst1` はキュー 3 にリストされたユーザーグループのメンバーですが、`'QG_B'` はキュー 2 に一致するため、クエリはキュー 2 に割り当てられます。
7. ユーザー `ralph` は、リストされたユーザーグループのメンバーではなく、クエリグループがリセットされたため、一致するキューはありません。クエリはデフォルトキューに割り当てられます。

キューへのクエリの割り当て

Amazon Redshift では、ワークロードの同時実行を管理し、クエリをキューに割り当てることでクエリに優先順位を付けることができます。キューを使用すると、重要度の低いクエリより重要なクエリが優先されるように、さまざまなタイプのクエリやユーザーにメモリや CPU などのリソースを割り当てることができます。以下のセクションでは、定義した基準に基づいてキューを作成し、プロパティを設定して、受信クエリを割り当てる方法について説明します。

次の例では、ユーザーロール、ユーザーグループ、クエリグループに従ってクエリをキューに割り当てます。

ユーザーロールに基づくクエリのキューへの割り当て

ユーザーがロールに割り当てられ、そのロールがキューにアタッチされている場合、そのユーザーによって実行されたクエリはそのキューに割り当てられます。次の例では、`sales_rw` という名前のユーザーロールを作成し、そのロールにユーザー `test_user` を割り当てます。

```
create role sales_rw;
grant role sales_rw to test_user;
```

1 つのロールを別のロールに明示的に付与することで、2 つのロールのアクセス許可を組み合わせることもできます。ネストされたロールをユーザーに割り当てると、両方のロールのアクセス許可がユーザーに付与されます。

```
create role sales_rw;
create role sales_ro;
grant role sales_ro to role sales_rw;
grant role sales_rw to test_user;
```

クラスターでロールが付与されているユーザーのリストを表示するには、SVV_USER_GRANTS テーブルにクエリを実行します。クラスターでロールが付与されているロールのリストを表示するには、SVV_ROLE_GRANTS テーブルにクエリを実行します。

```
select * from svv_user_grants;
select * from svv_role_grants;
```

ユーザーグループに基づくクエリのキューへの割り当て

ユーザーグループ名がキューの定義にリストされている場合、そのユーザーグループのメンバーによって実行されたキューは、対応するキューに割り当てられます。次の例では、ユーザーグループを作成し、SQL コマンド CREATE USER、CREATE GROUP、および ALTER GROUP を使用してユーザーをグループに割り当てます。

```
create group admin_group with user admin246, admin135, sec555;
create user vp1234 in group ad_hoc_group password 'vpPass1234';
alter group admin_group add user analyst44, analyst45, analyst46;
```

クエリグループへのクエリの割り当て

適切なクエリグループにクエリを割り当てることで、実行時にクエリをキューに割り当てることができます。クエリグループを開始するには、SET コマンドを使用します。

```
SET query_group TO group_label
```

ここで、*group_label* は、WLM 設定にリストされているクエリグループのラベルです。

SET query_group コマンドの後で実行するすべてのクエリは、クエリグループをリセットするか、現在のログインセッションを終了するまで、指定されたクエリグループのメンバーとして実行されます。Amazon Redshift オブジェクトの設定とリセットの詳細については、SQL コマンドリファレンスの「SET」および「RESET」を参照してください。

指定するクエリグループラベルは現在の WLM 設定に含まれている必要があります。それ以外の場合、SET query_group コマンドはクエリキューに対して効果がありません。

TO 句で定義されたラベルはクエリログにキャプチャされるので、このラベルをトラブルシューティングに使用できます。query_group 設定パラメータの詳細については、設定リファレンスの「query_group」を参照してください。

次の例では、クエリグループ 'priority' の一部として 2 つのクエリを実行し、クエリグループをリセットします。

```
set query_group to 'priority';
select count(*)from stv_blocklist;
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

Superuser キューへのクエリの割り当て

superuser キューにクエリを割り当てるには、スーパーユーザーとして Amazon Redshift にログオンし、superuser グループでクエリを実行します。終了したら、クエリグループをリセットし、以降のクエリが superuser キューで実行されないようにします。

次の例では、2 つのコマンドを superuser キューで実行するように割り当てます。

```
set query_group to 'superuser';

analyze;
vacuum;
reset query_group;
```

スーパーユーザーのリストを表示するには、PG_USER システムカタログテーブルをクエリします。

```
select * from pg_user where usesuper = 'true';
```

WLM の動的設定プロパティと静的設定プロパティ

WLM 設定プロパティは動的または静的のいずれかです。動的なプロパティは、クラスターを再起動することなくデータベースに適用できますが、静的プロパティで変更を有効にするには、クラスターの再起動が必要です。ただし、動的プロパティと静的プロパティを同時に変更する場合は、すべてのプロパティの変更を有効にするためにクラスターを再起動する必要があります。これは、変更されるプロパティが動的か静的かは関係ありません。

動的プロパティが適用されている場合でも、クラスターのステータスは `modifying` です。自動 WLM および手動 WLM 間の切り替えは静的な変更であり、クラスターの再起動が必要です。

次の表は、自動 WLM または手動 WLM を使用するときに動的または静的である WLM プロパティを示しています。

WLM プロパティ	自動 WLM	手動 WLM
クエリグループ	動的	静的
Query Group Wildcard	動的	静的
ユーザーグループ	動的	静的
User Group Wildcard	動的	静的
ユーザーロール	動的	静的
ユーザーロールワイルドカード	動的	静的
メインでの同時実行数	該当しません	動的
同時実行スケーリングモード	動的	動的
ショートクエリアクセラレーションの有効化	該当しません	動的
ショートクエリの最大実行時間	動的	動的
使用するメモリの割合	該当しません	動的

WLM プロパティ	自動 WLM	手動 WLM
タイムアウト	該当しません	動的
優先度	動的	該当しません
キューの追加または削除	動的	静的

クエリモニタリングルール (QMR) を変更すると、クラスターを変更しなくても自動的に変更が行われます。

> **Note**
>
> 手動 WLM を使用しており、タイムアウトの値が変更された場合、その新しい値は、値が変更された後に実行を開始する任意のクエリに適用されます。使用する同時実行またはメモリの割合を変更すると、Amazon Redshift は新しい設定に動的に変更されます。したがって、現在実行中のクエリは変更の影響を受けません。詳細については、「WLM の動的なメモリの割り当て」を参照してください。

トピック

WLM の動的なメモリ割り当て

各キューでは、WLM によりキューの同時実行レベルと同じ数のクエリスロットが作成されます。クエリスロットに割り当てられるメモリ量は、キューに割り当てられたメモリをスロットカウントで割った割合と同じです。メモリの割り当てや同時実行数を変更すると、Amazon Redshift は新しい WLM 設定への移行を動的に管理します。したがって、アクティブなクエリは、現在割り当てられているメモリ量を使用して完了まで実行できます。同時に、Amazon Redshift はメモリの合計使用量が使用可能なメモリの 100% を超えないようにします。

ワークロードマネージャは、次のプロセスを使用して移行を管理します。

1. WLM は、新しい各クエリスロットへのメモリ割り当てを再計算します。

2. クエリスロットが実行中のクエリによりアクティブに使用されていない場合、WLM によりスロットが削除され、新しいスロットがメモリを使用できるようになります。
3. クエリスロットがアクティブに使用されている場合、WLM はクエリが終了するまで待機します。
4. アクティブなクエリが完了すると、空のスロットが削除され、関連するメモリが解放されます。
5. 1 つ以上のスロットを追加するのに必要なメモリが使用可能になると、新しいスロットが追加されます。
6. 変更時に実行されていたすべてのクエリが完了すると、スロットカウントは新しい同時実行レベルと同じになり、新しい WLM 設定への移行が完了します。

実際、変更時に実行されていたクエリは、引き続き元のメモリ割り当てを使用します。変更時にキューに入っていたクエリは、新しく利用可能になったスロットにルーティングされます。

移行プロセス中に WLM の動的プロパティが変更された場合、WLM はすぐに現在の状態から新しい設定への移行を開始します。移行のステータスを表示するには、STV_WLM_SERVICE_CLASS_CONFIG システムテーブルにクエリを実行します。

動的な WLM の例

Amazon Redshift では、動的な WLM (ワークロード管理) を使用して、Amazon Redshift クラスター全体のワークロード分散とリソース割り当てを自動的に管理できます。動的な WLM は、ワークロードの需要に基づいてメモリ割り当てを動的に調整するワークロード管理 (WLM) 設定の例であり、最適な同時実行とパフォーマンスを可能にします。次のセクションでは、Amazon Redshift クラスターの動的な WLM の実装と設定について詳しく説明します。

次の動的プロパティを使用して、キューが 2 つ存在するクラスター WLM が設定されているとします。

キュー	Concurrency	% Memory to Use
1	4	50%
2	4	50%

ここでは、クラスターがクエリ処理に 200 GB のメモリを使用できるとします (この値は任意の数値であり、あくまでも例です)。次の式に示すように、各スロットには 25 GB 割り当てられます。

```
(200 GB * 50% ) / 4 slots  = 25 GB
```

次に、次の動的プロパティを使用するように WLM を変更します。

キュー	Concurrency	% Memory to Use
1	3	75%
2	4	25%

次の式に示すように、キュー 1 内の各スロットの新しいメモリ割り当ては 50 GB です。

```
(200 GB * 75% ) / 3 slots = 50 GB
```

新しい設定の適用時にクエリ A1、A2、A3、A4 が実行中であり、クエリ B1、B2、B3、B4 がキューに入っているとします。WLM は、クエリスロットを次のように動的に再設定します。

ステップ	実行中のクエリ	現在のスロットカウント	目標スロットカウント	割り当て済みメモリ	利用可能なメモリ
1	A1、A2、A3、A4	4	0	100 GB	50 GB
2	A2、A3、A4	3	0	75 GB	75 GB
3	A3、A4	2	0	50 GB	100 GB
4	A3、A4、B1	2	1	100 GB	50 GB
5	A4、B1	1	1	75 GB	75 GB
6	A4、B1、B2	1	2	125 GB	25 GB
7	B1、B2	0	2	100 GB	50 GB
8	B1、B2、B3	0	3	150 GB	0 GB

1. WLM は、各クエリスロットへのメモリ割り当てを再計算します。最初は、キュー 1 には 100 GB 割り当てられていました。新しいキューの合計割り当ては 150 GB のため、新しいキューはすぐに 50 GB 使用可能になります。キュー 1 が 4 つのスロットを使用するようになり、新しい同時実行レベルは 3 つのスロットになったため、新しいスロットは追加されません。
2. 1 つのクエリが終了したら、スロットは削除され、25 GB が解放されます。キュー 1 のスロットは 3 つ、使用可能なメモリは 75 GB になりました。新しい設定では新しいスロットごとに 50 GB 必要ですが、新しい同時実行レベルは 3 つのスロットのため、新しいスロットは追加されません。
3. 2 つ目のクエリが終了したら、スロットは削除され、25 GB が解放されます。キュー 1 のスロットは 2 つになり、空きメモリは 100 GB になります。
4. 空きメモリ 50 GB を使用して新しいスロットが追加されます。キュー 1 のスロットは 3 つになり、空きメモリは 50 GB になります。キューに入っているクエリを新しいスロットにルーティングできるようになりました。
5. 3 つ目のクエリが完了すると、スロットは削除され、25 GB が解放されます。キュー 1 のスロットは 2 つになり、空きメモリは 75 GB になります。
6. 空きメモリ 50 GB を使用して新しいスロットが追加されます。キュー 1 のスロットは 3 つになり、空きメモリは 25 GB になります。キューに入っているクエリを新しいスロットにルーティングできるようになりました。
7. 4 つ目のクエリが終了したら、スロットは削除され、25 GB が解放されます。キュー 1 のスロットは 2 つになり、空きメモリは 50 GB になります。
8. 空きメモリ 50 GB を使用して新しいスロットが追加されます。キュー 1 のスロットは、それぞれ 50 GB のスロットが 3 つになり、使用可能なすべてのメモリが割り当てられました。

移行が完了し、キューに入ったクエリがすべてのクエリスロットを使用できるようになります。

WLM クエリモニタリングルール

Amazon Redshift ワークロード管理では、クエリモニタリングルールは、WLM キューのメトリクスベースのパフォーマンスの境界を定義し、クエリがこれらの境界を超えた場合のアクションを指定します。例えば、実行時間の短いクエリ専用のキューには、60 秒以上実行されるクエリをキャンセルするルールを作成できます。デザインの不十分なクエリを追跡する目的で、ネステッドループを含むクエリを記録する別のルールを設定することができます。

ワークロード管理 (WLM) 構成の一部としてクエリモニタリングルールを定義します。1 つのキューに対して最大 25 個のルールを定義できます。ルールの総数はキュー全体で最大 25 個に制限されて

います。各ルールには最大 3 つの条件または述語と 1 つのアクションが含まれます。述語は、メトリクス、比較条件 (=, <, or >)、および値で構成されています。いずれかのルールのすべての述語が満たされると、ルールのアクションがトリガーされます。ルールアクションは、前述のようにログ、ホップ、中止を指定できます。

指定のキューのルールは、そのキューで実行中のクエリにのみ適用されます。ルールは他のルールに依存しません。

WLM は 10 秒ごとにメトリクスを評価します。同じ期間に複数のルールがトリガーされると、WLM は最も重大なアクションのルールを選択します。2 つのルールのアクションの重要度が同じ場合、WLM はルール名に基づいてアルファベット順にルールを実行します。アクションがホップまたは中止の場合、アクションは記録され、クエリはキューから削除されます。アクションがログ場合、キューはキューで実行されます。WLM はルールごとにクエリあたり 1 つのログアクションを開始します。キューに他のルールが含まれている場合、これらのルールは引き続き有効です。アクションがホップで、クエリが別のキューにルーティングされる場合、新しいキューのルールが適用されます。特定のクエリに対して実行されるクエリのモニタリングと追跡アクションの詳細については、ショートクエリアクセラレーション にあるサンプルコレクションを参照してください。

ルールの述語がすべて満たされると、WLM は STL_WLM_RULE_ACTION システムテーブルに行を書き込みます。さらに、Amazon Redshift は、現在実行されているクエリのクエリメトリクスを STV_QUERY_METRICS に記録します。完了したクエリのメトリクスは STL_QUERY_METRICS に保存されます。

クエリモニタリングルールの定義

WLM 構成の一部としてクエリモニタリングルールを作成します。このルールは、クラスターのパラメータグループ定義の一部として定義します。

ルールは AWS Management Consoleを使用して作成するか、JSON でプログラム的に作成できます。

> **Note**
>
> プログラム的にルールを作成することを選択した場合は、コンソールを使用して、パラメータグループ定義に含める JSON を生成することを強くおすすめします。詳細については、Amazon Redshift 管理ガイドの「コンソールを使用してクエリモニタリングルールを作成または変更する」および「AWS CLI によるパラメータ値を設定する」を参照してください。

クエリモニタリングルールを定義するには、次の要素を指定します。

- ルール名 – ルール名は WLM 設定内で一意である必要があります。ルール名には最大で 32 文字の英数字または下線を使用できます。スペースまたは疑問符を含めることはできません。キューごとの 25 個までルールを指定できます。すべてのキューでのルールは合計 25 個までです。
- 1 つ以上の述語 – ルールごとに最大 3 つのルールを定義できます。いずれかのルールのすべての述語が満たされると、関連付けられたアクションがトリガーされます。述語はメトリクス名、演算子 (=、<、または >)、および値で定義されます。例:「`query_cpu_time > 100000`」。メトリクスのリストと各メトリクスの値については、このセクションの Amazon Redshift のクエリモニタリングメトリクスのプロビジョニング を参照してください。
- アクション – 複数のルールがトリガーされると、WLM は最も重大なアクションのルールを選択します。想定されるアクションの重大度は昇順で次のようになります。
  - ログ – STL_WLM_RULE_ACTION システムテーブルにクエリに関する情報を記録します。ログレコードを書き込むだけの場合はログアクションを使用します。WLM は最大でクエリごと、ルールごとにログを 1 つ作成します。ログアクションの後、他のルールは有効な状態を維持し、WLM はひき続きクエリをモニタリングします。
  - ホップ (手動 WLM でのみ利用可能) – アクションを記録して、次に一致するキューにクエリをホップします。一致するキューがない場合、クエリはキャンセルされます。QMR は、CREATE TABLE AS (CTAS) ステートメントと読み取り専用クエリ (SELECT ステートメントなど) のみホップします。詳細については、「WLM クエリキューのホッピング」を参照してください。
  - 中断 – アクションをログに記録してクエリをキャンセルします。QMR は、COPY ステートメント、および ANALYZE や VACUUM などのメンテナンスオペレーションを停止しません。
  - 優先度の変更 (自動 WLM でのみ使用可能) – クエリの優先度を変更します。

クエリのランタイムを制限するには、WLM タイムアウトを使用する代わりにクエリモニタリングルールを作成することをお勧めします。例えば、次の JSON スニペットに示すように、`max_execution_time` を 50,000 ミリ秒に設定できます。

```
"max_execution_time": 50000
```

代わりに同等のクエリモニタリングルールを定義することをお勧めします。次の例は、`query_execution_time` を 50 秒に設定するクエリモニタリングルールを示しています。

```
"rules":
[
    {
```

```
        "rule_name": "rule_query_execution",
        "predicate": [
            {
                "metric_name": "query_execution_time",
                "operator": ">",
                "value": 50
            }
        ],
        "action": "abort"
    }
]
```

クエリモニタリングルールを作成または変更するステップについては、Amazon Redshift 管理ガイドの「コンソールを使用してクエリモニタリングルールを作成または変更する」および「wlm_json_configuration パラメータのプロパティ」を参照してください。

クエリモニタリングルールの詳細は次のトピックで確認できます。

- Amazon Redshift のクエリモニタリングメトリクスのプロビジョニング
- クエリモニタリングルールテンプレート
- コンソールを用いたルールの作成
- ワークロード管理の設定
- クエリのモニタリングルールのシステムテーブルとビュー

Amazon Redshift のクエリモニタリングメトリクスのプロビジョニング

次のテーブルに、クエリモニタリングルールで使用されるメトリクスを説明します (これらのメトリクスは、STV_QUERY_METRICS および STL_QUERY_METRICS システムのテーブルに保存されるメトリクスとは異なります)。

指定のメトリクスについて、パフォーマンスしきい値はクエリレベルまたはセグメントレベルのいずれかで追跡されます。セグメントとステップの詳細については、「クエリプランと実行ワークフロー」を参照してください。

> **Note**
>
> WLM タイムアウト パラメータは、クエリモニタリングルールとは異なります。

メトリクス	名前	説明
CPU 時間のクエリ	`query_cpu_time`	クエリに使用される CPU 時間 (秒)。CPU time は Query execution time とは異なります。 有効な値は 0～999,999 です。
ブロック読み取り	`query_blocks_read`	クエリによって読み取られた 1 MB データブロックの数。 有効な値は 0～1,048,575 です。
行数のスキャン	`scan_row_count`	スキャンステップの行数。行数は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前およびユーザー定義のクエリフィルタが適用される前に出力された合計行数を表します。 有効な値は 0～999,999,999,999,999 です。
クエリ実行時間	`query_execution_time`	経過したクエリ実行時間 (秒)。キューでの待機時間は実行時間に含まれません。 有効な値は 0～86,399 です。
クエリキュー時間	`query_queue_time`	キューでの待機に費やされた時間（秒単位）。 有効な値は 0～86,399 です。
CPU の使用	`query_cpu_usage_percent`	クエリが使用する CPU 容量の割合。 有効な値は 0～6,399 です。
ディスクへのメモリ	`query_temp_blocks_to_disk`	中間結果の書き込みに使用される一時ディスク容量 (1 MB ブロック)。 有効な値は 0～319,815,679 です。

メトリクス	名前	説明
CPU スキュー	cpu_skew	任意のスライスの最大 CPU 使用量とすべてのスライスの平均 CPU 使用量の比率。このメトリクスはセグメントレベルで定義されます。 有効な値は 0～99 です。
I/O スキュー	io_skew	任意のスライスの最大ブロック読み取り (I/O) とすべてのスライスの平均ブロック読み取りの比率。このメトリクスはセグメントレベルで定義されます。 有効な値は 0～99 です。
結合した行	join_row_count	結合ステップで処理された行数。 有効な値は 0～999,999,999,999,999 です。
ネストしたループ結合行数	nested_loop_join_row_count	ネストしたループ結合行数の行数。 有効な値は 0～999,999,999,999,999 です。
返される行数	return_row_count	クエリによって返された行の数。 有効な値は 0～999,999,999,999,999 です。
セグメント実行時間	segment_execution_time	単一セグメントで経過した実行時間 (秒)。サンプリングエラーを回避または減少するには、segment_execution_time > 10 をルールに含めます。 有効な値は 0～86,388 です。
Spectrum スキャン行数	spectrum_scan_row_count	Amazon Redshift Spectrum クエリにスキャンされる Amazon S3 内のデータ行数。 有効な値は 0～999,999,999,999,999 です。

メトリクス	名前	説明
Spectrum スキャンサイズ	spectrum_scan_size_mb	Amazon Redshift Spectrum クエリにスキャンされる Amazon S3 内のデータサイズ (MB)。 有効な値は 0～999,999,999,999,999 です。
クエリ優先度	query_priority	クエリの優先度です。 有効な値は、HIGHEST、HIGH、NORMAL、LOW、LOWEST です。大なり (>) 演算子と小なり (<) 演算子を使用して query_priority を比較する場合、HIGHEST は HIGH より大きく、HIGH は NORMAL より大きい、などがあります。

> **Note**
>
> - ホップアクションは、query_queue_time 述語ではサポートされていません。つまり、query_queue_time 述語が満たされたときにホップするように定義されたルールは無視されます。
> - 短いセグメントの実行時間は、一部のメトリクスで io_skew や query_cpu_usage_percent などのサンプリングエラーとなる場合があります。サンプリングエラーを回避または減少するには、セグメントの実行時間をルールに含めます。最良な開始ポイントは、segment_execution_time > 10 です。

SVL_QUERY_METRICS ビューは、完了したクエリのメトリクスを示します。SVL_QUERY_METRICS_SUMMARY ビューは、完了したクエリのメトリクスの最大値を示します。このビューの値は、クエリのモニタリングルールを定義するしきい値を決定する際に役立ちます。

Amazon Redshift Serverless のクエリモニタリングメトリクス

次のテーブルでは、Amazon Redshift Serverless のクエリモニタリングルールで使用されるメトリクスについて説明します。

メトリクス	名前	説明
CPU 時間のクエリ	max_query_cpu_time	クエリに使用される CPU 時間 (秒)。CPU time は Query execution time とは異なります。 有効な値は 0～999,999 です。
ブロック読み取り	max_query_blocks_read	クエリによって読み取られた 1 MB データブロックの数。 有効な値は 0～1,048,575 です。
行数のスキャン	max_scan_row_count	スキャンステップの行数。行数は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前およびユーザー定義のクエリフィルタが適用される前に出力された合計行数を表します。 有効な値は 0～999,999,999,999,999 です。
クエリ実行時間	max_query_execution_time	経過したクエリ実行時間 (秒)。キューでの待機時間は実行時間に含まれません。クエリが設定された実行時間を超えると、Amazon Redshift Serverless はクエリを中止します。 有効な値は 0～86,399 です。
クエリキュー時間	max_query_queue_time	キューでの待機に費やされた時間 (秒単位)。 有効な値は 0～86,399 です。
CPU の使用	max_query_cpu_usage_percent	クエリが使用する CPU 容量の割合。 有効な値は 0～6,399 です。
ディスクへのメモリ	max_query_temp_blocks_to_disk	中間結果の書き込みに使用される一時ディスク容量 (1 MB ブロック)。 有効な値は 0～319,815,679 です。

メトリクス	名前	説明
結合した行	max_join_row_count	結合ステップで処理された行数。 有効な値は 0～999,999,999,999,999 です。
ネストしたループ結合行数	max_nested_loop_join_row_count	ネストしたループ結合行数の行数。 有効な値は 0～999,999,999,999,999 です。

> **Note**
>
> - ホップアクションは、max_query_queue_time 述語ではサポートされていません。つまり、max_query_queue_time 述語が満たされたときにホップするように定義されたルールは無視されます。
> - 短いセグメントの実行時間は、一部のメトリクスで max_io_skew や max_query_cpu_usage_percent などのサンプリングエラーとなる場合があります。

クエリモニタリングルールテンプレート

Amazon Redshift コンソールを使用してルールを追加すると、事前定義されたテンプレートからルールを作成することを選択できます。Amazon Redshift は、述語のセットを含む新しいルールを作成し、述語にデフォルト値で入力します。デフォルトのアクションはログです。述語とアクションはユースケースに合わせて変更できます。

次の表に利用可能なテンプレートの一覧を示します。

テンプレート名	述語	説明
ネストされたループ結合	nested_loop_join_row_count > 100	ネストしたループ結合は、不完全な結合述語を示すことがあります。この場合、高確率でリターンセットが非常に大きくなります (デカルト積)。少ない行数を使用して潜在的なランナウェイクエリを早期に検出します。

テンプレート名	述語	説明
クエリが多くの行を返す	`return_row_count > 1000000`	キューを実行中のシンプルで短いクエリ専用にすると、多くの行数を返すクエリを見つけるルールが含まれる可能性があります。テンプレートはデフォルトで 100 万行を使用します。システムによっては、100 万行で多いとみなされます。またより大きなシステムでは、10 億行で多いとみなされます。
多くの行がある結合	`join_row_count > 1000000000`	結合ステップの行数が以上に多い場合は、フィルタの制限を厳格化することが必要な可能性があります。テンプレートはデフォルトで 10 億行を使用します。素早くシンプルなクエリを目的にしたアドホック (ワンタイム) キューの場合は、この行数を小さくすることができます。
中間結果を書き込むときの高いディスク使用	`query_temp_blocks_to_disk > 100000`	現在実行中のクエリが制限を超えてシステム RAM を使用する場合、クエリ実行エンジンは中間結果をディスク (スピルされたメモリ) に書き込みます。通常、この条件は不正なクエリの結果です。また、この不正なクエリはディスク容量の大部分を使用します。ディスク使用量の許容可能なしきい値は、クラスターノードのタイプと数に応じて異なります。テンプレートは、デフォルトで 100,000 ブロックまたは 100 GB を使用します。小さなクラスターでは少ない数を使用できます。
高い I/O スキューで長時間実行されているクエリ	`segment_execution_time > 120` および `io_skew > 1.30`	I/O スキューは、1 つのノードスライスの I/O 料金が他のスライスよりもかなり高い場合に発生します。開始点として、1.30 のスキュー (平均 1.3 倍) は高いとみなされます。I/O スキューが高いことが必ず問題であるわけではありませんが、クエリの実行が長時間におよぶなどの状態と組み合わさる場合は、分散スタイルやソートキーに問題がある可能性があります。

クエリのモニタリングルールのシステムテーブルとビュー

ルールの述語がすべて満たされると、WLM は STL_WLM_RULE_ACTION システムテーブルに行を書き込みます。この行には、ルールをトリガーしたクエリとその結果のアクションに関する詳細が含まれます。

さらに、Amazon Redshift では、次のシステムテーブルとビューにクエリメトリクスが記録されます。

- STV_QUERY_METRICS テーブルは、現在実行中のクエリのメトリクスを表示します。
- STL_QUERY_METRICS テーブルは完了したクエリのメトリクスを記録します。
- SVL_QUERY_METRICS ビューは、完了したクエリのメトリクスを示します。
- SVL_QUERY_METRICS_SUMMARY ビューは、完了したクエリのメトリクスの最大値を示します。

WLM システムテーブルとビュー

WLM は、内部で定義されている WLM サービスクラスに従ってクエリキューを設定します。Amazon Redshift は、WLM 設定で定義されたキューとともに、これらのサービスクラスに従っていくつかの内部キューを作成します。キューおよびサービスクラスという用語は、通常、システムテーブルでは同じ意味で使用されます。スーパーユーザーキューは、サービスクラス 5 を使用します。ユーザー定義キューは、サービスクラス 6 以上を使用します。

クエリ、キュー、およびサービスクラスは、WLM に固有のシステムテーブルを使用して確認できます。次の操作を実行するには、次のテーブルに対してクエリを実行します。

- ワークロードマネージャーによって追跡されているクエリと、割り当てられているリソースを確認する。
- クエリの割り当て先のキューを確認する。
- 現在、ワークロードマネージャーによって追跡されているクエリのステータスを確認する。

テーブル名	説明
STL_WLM_ERROR	WLM 関連エラーイベントのログを含みます。
STL_WLM_QUERY	WLM によって追跡されているクエリをリストします。

テーブル名	説明
STV_WLM_CLASSIFICATION_CONFIG	WLM 用の現在の分類ルールを表示します。
STV_WLM_QUERY_QUEUE_STATE	クエリキューの現在の状態を記録します。
STV_WLM_QUERY_STATE	WLM によって追跡されているクエリの現在の状態のスナップショットを提供します。
STV_WLM_QUERY_TASK_STATE	クエリタスクの現在の状態を含みます。
STV_WLM_SERVICE_CLASS_CONFIG	WLM のサービスクラス設定を記録します。
STV_WLM_SERVICE_CLASS_STATE	サービスクラスの現在の状態を表示します。
STL_WLM_RULE_ACTION	レコードは、ユーザー定義のキューに関連付けられた WLM クエリモニタリングルールによって生じたアクションの詳細を示します。
STV_WLM_QMR_CONFIG	WLM クエリモニタリングルール (QMR) の構成を記録します。

システムテーブルのクエリを追跡するには、タスク ID を使用します。次の例に、最近送信されたユーザークエリのタスク ID を取得する方法を示します。

```
select task from stl_wlm_query where exec_start_time =(select max(exec_start_time) from
 stl_wlm_query);

task
------
137
(1 row)
```

次の例では、現在実行中あるいは複数のサービスクラス (キュー) を待機中のクエリを表示しています。このクエリは、Amazon Redshift の全体的な同時実行ワークロードを追跡するのに役立ちます。

```
select * from stv_wlm_query_state order by query;


xid |task|query|service_| wlm_start_  |  state   |queue_ | exec_
    |    |     |class   | time        |          |time   | time
----+----+-----+--------+-------------+---------+-------+--------
2645| 84 | 98  | 3      | 2010-10-... |Returning|   0   | 3438369
2650| 85 | 100 | 3      | 2010-10-... |Waiting  |   0   | 1645879
2660| 87 | 101 | 2      | 2010-10-... |Executing|   0   | 916046
2661| 88 | 102 | 1      | 2010-10-... |Executing|   0   | 13291
(4 rows)
```

WLM サービスクラス ID

次の表は、サービスクラスに割り当てられている ID の一覧です。

ID	サービスクラス
1 ~ 4	将来の利用のために予約されています。
5	スーパーユーザーキューで使用します。
6 ~ 13	WLM 設定で定義されている手動 WLM キューで使用します。
14	ショートクエリアクセラレーションで使用します。
15	Amazon Redshift で実行されるメンテナンスアクティビティ用に予約されています。
100 ~ 107	auto_wlm が true の場合に自動 WLM キューで使用します。

データベースセキュリティ

データベースセキュリティは、各データベースオブジェクトに対するアクセス権限を付与するユーザーを制御することによって管理します。ユーザーにはロールまたはグループを割り当てることができ、ユーザー、ロール、またはグループに付与するアクセス許可によって、アクセスできるデータベースオブジェクトが決まります。

トピック

データベースオブジェクトに対するアクセス権限は、ユーザーまたはロールに付与したアクセス許可に応じて異なります。データベースセキュリティの機能方法について、以下のガイドラインにまとめてあります。

- デフォルトでは、オブジェクト所有者のみにアクセス許可が付与されます。
- Amazon Redshift データベースユーザーは、データベースに接続できる名前付きユーザーです。ユーザーへのアクセス許可の付与には、明示的方法 (アカウントに直接割り当てる) と、暗黙的方法 (アクセス許可を付与するグループのメンバーにする) の 2 つがあります。
- グループとはユーザーが集合したものであり、これにアクセス許可を一括して割り当てることで、セキュリティの管理を合理的に行えます。
- スキーマは、データベーステーブルおよびその他のデータベースオブジェクトの集合です。スキーマは、ファイルシステムディレクトリに似ていますが、ネストできない点が異なります。ユーザーには、単一のスキーマまたは複数のスキーマに対するアクセス権限を付与できます。

さらに、Amazon Redshift は以下の機能を採用しているため、どのユーザーがどのデータベースオブジェクトにアクセスできるかをより細かく制御できます。

- ロールベースのアクセスコントロール (RBAC) により、アクセス許可をロールに割り当て、ロールをユーザーに適用できるため、大規模なユーザーグループのアクセス許可を制御できます。グループとは異なり、ロールは他のロールからアクセス許可を継承できます。

  行レベルセキュリティ (RLS) により、選択した行へのアクセスを制限するポリシーを定義して、それらのポリシーをユーザーまたはグループに適用できます。

  動的データマスキング (DDM) は、クエリランタイムにデータを変換することでデータをさらに保護するため、ユーザーは機密情報を公開することなくデータにアクセスできます。

セキュリティ実装の例については、「ユーザーおよびグループのアクセス権限の管理例」を参照してください。

データ保護の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のセキュリティ」を参照してください。

Amazon Redshift セキュリティの概要

Amazon Redshift データベースのセキュリティは、他の種類の セキュリティとは異なります。Amazon Redshift には、このセクションで説明しているデータベースセキュリティに加えて、以下のセキュリティ管理といった特徴が用意されています。

- サインイン認証情報 – Amazon Redshift の AWS マネジメントコンソールへのアクセスは、AWS アカウントのアクセス許可によって管理されます。詳細については、「サインイン認証情報」を参照してください。
- アクセス管理 — 特定の Amazon Redshift リソースへのアクセスを管理するには、AWS Identity and Access Management (IAM) アカウントを定義します。詳細については、Amazon Redshift のリソースに対するアクセスの制御を参照してください。
- クラスターセキュリティグループ – 他のユーザーに Amazon Redshift クラスターへのインバウンドアクセス権を付与するには、クラスターセキュリティグループを定義し、それをクラスターに関連付けます。詳細については、「Amazon Redshift クラスターセキュリティグループ」を参照してください。

- VPC – 仮想ネットワーク環境を使用してクラスターへのアクセスを保護するには、Amazon Virtual Private Cloud (VPC) でクラスターを起動します。詳細については、「Virtual Private Cloud (VPC) でクラスターを管理する」を参照してください。
- クラスターの暗号化 – ユーザーが作成したすべてのテーブル内のデータを暗号化するには、クラスターの起動時に、そのクラスターの暗号化を有効にします。詳細については、「Amazon Redshift クラスター」を参照してください。
- SSL 接続 – SQL クライアントとクラスター間の接続を暗号化するには、Secure Sockets Layer (SSL) 暗号化を使用します。詳細については、「SSL を使用してクラスターに接続する」を参照してください。
- ロードデータ暗号化 – テーブルロードデータファイルを Amazon S3 にアップロードするときに暗号化するには、サーバー側の暗号化またはクライアント側の暗号化を使用できます。サーバー側で暗号化されたデータからロードする場合、Amazon S3 が透過的に復号を処理します。クライアント側で暗号化されたデータからロードする場合、Amazon Redshift の COPY コマンドによって、テーブルをロードするときにデータが復号されます。詳細については、「Amazon S3 への暗号化されたデータのアップロード」を参照してください。
- 転送中のデータ – AWS クラウド内で転送されるデータを保護するために、Amazon Redshift では、COPY、UNLOAD、バックアップ、および復元オペレーションを実行する際、ハードウェアでアクセラレートされた SSL を使用して、Amazon S3 または Amazon DynamoDB と通信します。
- 列レベルのアクセスコントロール – Amazon Redshift でデータに対して列レベルのアクセスを制御するには、ビューベースのアクセスコントロールを実装したり、別のシステムを使用したりせずに、列レベルの許可ステートメントや revoke ステートメントを使用します。
- 行レベルのセキュリティ制御 — Amazon Redshift のデータに対して行レベルのセキュリティ制御を行うには、ポリシーで定義された行へのアクセスを制限するポリシーを作成してロールまたはユーザーにアタッチします。

データベースユーザーのデフォルトのアクセス許可

データベースオブジェクトを作成したユーザーは、その所有者になります。デフォルトでは、スーパーユーザーまたはオブジェクトの所有者のみが、そのオブジェクトに対するクエリや変更を実行でき、またアクセス許可を付与できます。ユーザーがオブジェクトを使用するには、そのユーザー、またはそのユーザーが属するグループに、必要なアクセス許可を付与する必要があります。データベースのスーパーユーザーは、そのデータベースの所有者と同じアクセス許可を持ちます。

Amazon Redshift でサポートされるアクセス許可は、SELECT、INSERT、UPDATE、DELETE、REFERENCES、CREATE、TEMPORARY、およ

び USAGE です。異なるタイプのオブジェクトには、それぞれ異なるアクセス許可が関連付けられます。Amazon Redshift でサポートされるデータベースオブジェクトのアクセス許可については、GRANT コマンドを参照してください。

オブジェクトを変更または破棄する許可を持つのは、所有者のみです。

デフォルトでは、データベースの PUBLIC スキーマに対して、すべてのユーザーが CREATE と USAGE のアクセス許可を持ちます。データベースの PUBLIC スキーマ内に、オブジェクトをユーザーが作成することを禁止するには、REVOKE コマンドを使用して、そのアクセス許可を削除します。

以前付与したアクセス許可を取り消す場合は、REVOKE コマンドを使用します。オブジェクト所有者が持つ (DROP、GRANT、REVOKE などの) アクセス許可は暗黙的であり、付与したり取り消したりすることはできません。オブジェクトの所有者は、自身が持つ通常の (例えば、テーブルを所有者自身と他のユーザーに対して読み取り専用にするなどの) アクセス許可を取り消すことができます。スーパーユーザーは、GRANT コマンドであるか REVOKE コマンドであるかを問わず、すべてのアクセス許可を保持します。

スーパーユーザー

すべてのデータベースについて、データベースのスーパーユーザーは、そのデータベースの所有者と同様のアクセス許可を保持します。

クラスターの起動時に作成する管理者ユーザーと呼ばれるユーザーは、スーパーユーザーです。

スーパーユーザーを作成できるのは、スーパーユーザーのみです。

Amazon Redshift のシステムテーブルおよびシステムビューは、スーパーユーザーのみが表示できるか、すべてのユーザーが表示できます。「スーパーユーザーが表示できるビュー」として指定されているシステムテーブルおよびシステムビューを照会できるのは、スーパーユーザーのみです。詳細については、「SYS モニタリングビュー」を参照してください。

スーパーユーザーは、すべてのカタログテーブルを表示できます。詳細については、システムカタログテーブル を参照してください。

データベーススーパーユーザーは、すべての許可チェックをバイパスします。スーパーユーザーは、GRANT コマンドであるか REVOKE コマンドであるかを問わず、すべてのアクセス許可を保持します。スーパーユーザーロールを使用するときには、注意が必要です。ほとんどの作業をスーパーユーザー以外のロールで行うことをお勧めします。より制限の厳しい権限を持つ管理者ロールを作成

できます。ロールの作成の詳細については、「ロールベースのアクセスコントロール (RBAC)」を参照してください。

新しいデータベーススーパーユーザーを作成するには、スーパーユーザーとしてデータベースにログオンし、CREATEUSER のアクセス許可を使用して、CREATE USER コマンドまたは ALTER USER コマンドを実行します。

```
CREATE USER adminuser CREATEUSER PASSWORD '1234Admin';
ALTER USER adminuser CREATEUSER;
```

スーパーユーザーを作成、変更、または削除するには、ユーザーの管理と同じコマンドを使用します。詳細については、「ユーザーの作成、変更、および削除」を参照してください。

ユーザー

データベースユーザーの作成および管理は、Amazon Redshift SQL コマンドの CREATE USER および ALTER USER により行えます。また、Amazon Redshift での JDBC または ODBC 用カスタムドライバーを使用すると、SQL クライアントを設定できます。これらのドライバーは、データベースのログオンプロセスの一部として、データベースユーザーや仮パスワードを作成するプロセスを実行します。

ドライバーは AWS Identity and Access Management (IAM) 認証に基づいてデータベースユーザーを認証します。既に AWS の外部でユーザー ID を管理している場合、Security Assertion Markup Language (SAML) 2.0 に準拠した ID プロバイダー (IdP) を使用して、Amazon Redshift リソースへのアクセスを管理できます。IAM ロールを使用して IdP と AWS の設定を行い、フェデレーションユーザーが一時データベース認証情報を生成して、Amazon Redshift データベースにログオンすることを許可します。詳細については、「IAM 認証を使用したデータベースユーザー認証情報の生成」を参照してください。

Amazon Redshift ユーザーを作成および削除できるのは、データベーススーパーユーザーのみです。Amazon Redshift にログオンしようとするユーザーに対しては、認証が行われます。ユーザーは、データベースやデータベースオブジェクト (例えばテーブルなど) を所有できます。また、これらのオブジェクトに対するアクセス許可を、他のユーザーやグループおよびスキーマに付与することで、各オブジェクトにアクセスできるユーザーを管理します。CREATE DATABASE 権限を持つユーザーは、データベースを作成し、それらのデータベースに許可を付与できます。スーパーユーザーは、すべてのデータベースに対し、データベース所有者と同じアクセス許可を保持します。

ユーザーの作成、変更、および削除

データベースユーザーは、データウェアハウスクラスター全体にわたって (データベースごとではなく) グローバルです。

- ユーザーを作成するには、CREATE USER コマンドを使用します。
- スーパーユーザーを作成するには、CREATEUSER オプションを指定しながら CREATE USER コマンドを使用します。
- 既存のユーザーを削除するには、DROP USER コマンドを使用します。
- ユーザーを変更する (例: パスワードを変更する) には、ALTER USER コマンドを使用します。
- ユーザーのリストを表示するには、PG_USER カタログテーブルに対しクエリを実行します。

```
select * from pg_user;

  usename    | usesysid | usecreatedb | usesuper | usecatupd |  passwd  | valuntil |
 useconfig
------------+----------+-------------+----------+-----------+----------+----------
+-----------
 rdsdb      |        1 | t           | t        | t         | ******** |          |
 masteruser |      100 | t           | t        | f         | ******** |          |
 dwuser     |      101 | f           | f        | f         | ******** |          |
 simpleuser |      102 | f           | f        | f         | ******** |          |
 poweruser  |      103 | f           | t        | f         | ******** |          |
 dbuser     |      104 | t           | f        | f         | ******** |          |
(6 rows)
```

グループ

グループとは、そこで関連付けられている任意のアクセス許可が付与されているユーザーの集合を言います。グループを使用してアクセス許可を割り当てることができます。例えば、営業、管理、サポートなど、さまざまなグループを作成して、各グループ内のユーザーに、その業務に必要なデータへの適切なアクセス権限を付与できます。アクセス許可はグループレベルで付与または取り消すことができ、その変更は (スーパーユーザーを除く) グループのすべてのメンバーに適用されます。

すべてのユーザーグループを表示するには、次のように PG_GROUP システムカタログテーブルを照会します。

```
select * from pg_group;
```

例えば、すべてのデータベースユーザーをグループ別に一覧表示するには、次の SQL を実行します。

```
SELECT u.usesysid
,g.groname
,u.usename
FROM pg_user u
LEFT JOIN pg_group g ON u.usesysid = ANY (g.grolist)
```

グループの作成、変更、および削除

グループの作成、変更、削除ができるのは、スーパーユーザーだけです。

以下のアクションを実行できます。

- グループを作成するには、CREATE GROUP コマンドを使用します。
- ユーザーを既存のグループに追加するには、または既存のグループから削除するには、ALTER GROUP コマンドを使用します。
- グループを削除するには、DROP GROUP コマンドを使用します。このコマンドはグループのみを削除し、そのメンバーユーザーは削除しません。

ユーザーおよびグループのアクセス権限の管理例

この例では、ユーザーグループとユーザーを作成した上で、ウェブアプリケーションクライアントに接続する Amazon Redshift データベースへのさまざまなアクセス許可を付与しています。この例では、3 つのユーザーグループが想定されています。ウェブアプリケーションの一般ユーザー、ウェブアプリケーションのパワーユーザー、およびウェブ開発者の 3 グループです。

1. ユーザーを割り当てるグループを作成します。以下の一連のコマンドで、3 つのユーザーグループを作成します。

   ```
   create group webappusers;

   create group webpowerusers;

   create group webdevusers;
   ```

2. 複数のデータベースユーザーを作成し、それぞれに異なるアクセス許可を付与した上で、グループに追加します。

a. 2 つのユーザーを作成し、それらを WEBAPPUSERS グループに追加します。

```
create user webappuser1 password 'webAppuser1pass'
in group webappusers;

create user webappuser2 password 'webAppuser2pass'
in group webappusers;
```

b. ウェブデベロッパーユーザーを作成して WEBDEVUSERS グループに追加します。

```
create user webdevuser1 password 'webDevuser2pass'
in group webdevusers;
```

c. スーパーユーザーを作成します。このユーザーには、他のユーザーを作成できる管理権限が与えられます。

```
create user webappadmin  password 'webAppadminpass1'
createuser;
```

3. スキーマを作成し、ウェブアプリケーションで使用されるデータベーステーブルに関連付けてから、そのスキーマに対するアクセス権限をさまざまなユーザーグループに付与します。

a. WEBAPP スキーマを作成します。

```
create schema webapp;
```

b. WEBAPPUSERS グループに USAGE のアクセス許可を付与します。

```
grant usage on schema webapp to group webappusers;
```

c. WEBPOWERUSERS グループに USAGE のアクセス許可を付与します。

```
grant usage on schema webapp to group webpowerusers;
```

d. WEBDEVUSERS グループに ALL のアクセス許可を付与します。

```
grant all on schema webapp to group webdevusers;
```

これで、基本的なユーザーおよびグループがセットアップされました。ユーザーおよびグループを変更できるようになりました。

4. 例えば、次のコマンドは、WEBAPPUSER1 の search_path パラメータを変更します。

```
alter user webappuser1 set search_path to webapp, public;
```

SEARCH_PATH は、スキーマが指定されていない簡潔な名前でデータベースオブジェクト (テーブルや関数など) が参照されたときにスキーマを検索する順序を指定します。

5. また、グループを作成した後で、ユーザーをグループに追加することもできます。例えば、次のように WEBAPPUSER2 を WEBPOWERUSERS グループに追加します。

```
alter group webpowerusers add user webappuser2;
```

スキーマ

データベースには、1 つ以上の名前付きスキーマが含まれています。データベース内の各スキーマには、テーブルや、その他の種類の名前付きオブジェクトが含まれています。デフォルトでは、データベースに、PUBLIC という名前の 1 つのスキーマが含まれています。スキーマを使用すると、共通の名前でデータベースオブジェクトをグループ化できます。スキーマは、ファイルシステムディレクトリに似ていますが、ネストできない点が異なります。

同じデータベース内の異なるスキーマ内で同一のデータベースオブジェクト名を使用でき、競合は生じません。例えば、MY_SCHEMA と YOUR_SCHEMA の両方に、MYTABLE という名前のテーブルを含めることができます。必要なアクセス許可が付与されたユーザーであれば、データベース内の複数のスキーマにわたりオブジェクトにアクセスできます。

デフォルトでは、オブジェクトは、データベースの検索パスに含まれる最初のスキーマ内に作成されます。詳細については、このセクションで後述する「検索パス」を参照してください。

スキーマは、組織において、またマルチユーザー環境での並行処理問題において以下のように有用です。

- 多数のデベロッパーが相互に妨害せずに同じデータベース内で作業できる。
- データベースオブジェクトの論理グループを編成して、より管理しやすくする。
- アプリケーションで使用されるオブジェクトを専用のスキーマに入れる機能をアプリケーションに追加する。これにより、それらのオブジェクトの名前と、別のアプリケーションで使用されるオブジェクトの名前とが競合しなくなる。

検索パス

検索パスは search_path パラメータで定義します。このパラメータに、スキーマ名をカンマで区切ってリストします。検索パスは、オブジェクト (テーブルや関数など) がスキーマ修飾子なしの簡潔な名前で参照されたときにスキーマを検索する順序を指定します。

ターゲットスキーマを指定せずにオブジェクトを作成した場合は、検索パスにリストされている最初のスキーマにオブジェクトが追加されます。同じ名前の複数のオブジェクトが異なるスキーマ内に存在する場合、スキーマが指定されていないオブジェクト名は、その名前のオブジェクトを含む検索パス内の最初のスキーマを参照します。

現行セッションのデフォルトスキーマを変更するには、SET コマンドを使用します。

詳細については、「設定リファレンス」の「search_path」の説明を参照してください。

スキーマの作成、変更、および削除

あらゆるユーザーがスキーマを作成でき、所有するスキーマを変更または削除できます。

以下のアクションを実行できます。

- スキーマを作成するには、CREATE SCHEMA コマンドを使用します。
- スキーマの所有者を変更するには、ALTER SCHEMA コマンドを使用します。
- スキーマおよびそのオブジェクトを削除するには、DROP SCHEMA コマンドを使用します。
- スキーマ内にテーブルを作成するには、schema_name.table_name という形式でテーブルを作成します。

すべてのスキーマのリストを表示するには、PG_NAMESPACE システムカタログテーブルをクエリします。

```
select * from pg_namespace;
```

スキーマに属するテーブルのリストを表示するには、PG_TABLE_DEF システムカタログテーブルをクエリします。たとえば、次のクエリは PG_CATALOG スキーマのテーブルのリストを返します。

```
select distinct(tablename) from pg_table_def
where schemaname = 'pg_catalog';
```

スキーマベースのアクセス許可。

スキーマベースのアクセス許可は、スキーマ所有者により以下のように決定されます。

- デフォルトでは、データベースの PUBLIC スキーマに対して、すべてのユーザーが CREATE と USAGE のアクセス許可を持ちます。データベースの PUBLIC スキーマ内に、オブジェクトをユーザーが作成することを禁止するには、REVOKE コマンドを使用して、そのアクセス許可を削除します。
- オブジェクトの所有者により USAGE のアクセス許可が付与されたユーザーでない限り、ユーザーは、自らが所有していないスキーマに含まれるオブジェクトにアクセスできません。
- 別のユーザーが作成したスキーマに対する CREATE のアクセス許可が付与されたユーザーは、そのスキーマ内にオブジェクトを作成できます。

ロールベースのアクセスコントロール (RBAC)

ロールベースのアクセスコントロール (RBAC) を使用して、Amazon Redshift でのデータベース許可を管理することで、Amazon Redshift 内のセキュリティに関する許可の管理を簡略化できます。ユーザーが実行できる操作を、広範な、または詳細なレベルの両方で制御することで、機密データへのアクセスを保護できます。また、通常はスーパーユーザーに制限されるタスクへのユーザーによるアクセスを制御することもできます。異なるロールに異なるアクセス許可を割り当て、それらを異なるユーザーに割り当てることで、ユーザーアクセスをより細かく制御できます。

ロールが割り当てられたユーザーは、そのロールが指定し承認したタスクのみ実行できます。例えば、CREATE TABLE および DROP TABLE 許可を持つロールが割り当てられたユーザーについては、これらのタスクの実行のみが承認されます。作業で必要とするデータに対する各ユーザーアクセスを制御するには、セキュリティに関するさまざまなレベルの許可を、それぞれのユーザーに付与します。

RBAC では、対象となるオブジェクトのタイプに関係なく、ロールの要件に基づいて、最小許可の原則をユーザーに適用します。アクセス許可の付与と取り消しはロールレベルで実行され、個々のデータベースオブジェクトにおいてアクセス許可を更新する必要はありません。

RBAC では、スーパーユーザーのアクセスが必要となるコマンドを実行するための、アクセス許可を持つロールを作成できます。これらの許可を含むロールで承認されているのであれば、ユーザーはこの種類のコマンドを実行できます。同様に、特定のコマンドへのアクセスを制限するロールを作成し、その中で承認されたスーパーユーザーまたはユーザーのいずれかに対して、ロールを割り当てることもできます。

Amazon Redshift RBAC の仕組みについては、以下の Amazon Redshift のロールベースアクセスコントロール (RBAC) の導入をご覧ください。

ロール階層

ロールとは、ユーザーまたは別のロールに割り当てることができるアクセス許可の集合です。ロールには、システムまたはデータベースへのアクセス許可を割り当てることができます。ユーザーは、割り当てられたロールのアクセス許可を継承します。

RBAC では、ネストされたロールをユーザーに付与できます。ロールは、ユーザーとロールの両方に付与することが可能です。ユーザーにロールを付与すると、このロールに含まれるすべてのアクセス許可が、そのユーザーに対し承認されます。ユーザーにロール r1 を付与するとすれば、r1 が持つ権限がそのユーザーでも承認されます。このユーザーには r1 からのアクセス許可が与えられると同時に、すでに保持していた既存の許可も維持されます。

あるロール (r1) を別のロール (r2) に付与する場合、r1 のすべてのアクセス許可が r2 においても承認されます。さらに、r2 を別のロール (r3) に付与すると、r3 には r1 と r2 の許可を組み合わせた許可が付与されます。このロール階層では、r1 からの許可が r2 に継承されています。Amazon Redshift は、アクセス許可とともに各ロールの認可を伝播します。r1 を r2 に付与し、r2 を r3 に付与すると、3 つのロールのすべてのアクセス許可が r3 において承認されます。したがって、ユーザーに r3 を付与することで、このユーザーには 3 つのロールからのすべてのアクセス許可が付与されます。

Amazon Redshift では、ロールの認可サイクルを作成することはできません。ロールの認可サイクルは、ネストされたロールがロール階層で前の段階にあるロールに割り当てられる場合に発生します。例えば、ここで r3 を r1 に割り当てる場合がこれに相当します。IAM ロールを作成し、ロールの割り当てを管理する方法の詳細については「RBAC でのロールの管理」を参照してください。

ロールの割り当て

CREATE ROLE のアクセス許可を持つスーパーユーザーおよび通常のユーザーは、CREATE ROLE ステートメントを使用してロールを作成できます。スーパーユーザーとロールの管理者は、GRANT ROLE ステートメントを使用して、他のユーザーにロールを付与できます。これらのユーザー (管理者) は、REVOKE ROLE ステートメントを使用して他のユーザーが持つロールを取り消したり、DROP ROLE ステートメントを使用してロールを削除したりできます。ロール管理者には、ロール所有者と ADMIN OPTION のアクセス許可を持つロールを付与されたユーザーが含まれます。

ロールの付与と取り消しを行えるのは、スーパーユーザーまたはロールの管理者のみです。1 つまたは複数のロールやユーザーに対して、1 つ以上のロールを付与または取り消すことができます。GRANT ROLE ステートメントで WITH ADMIN OPTION オプションを使用して、付与されたすべてのロールの管理オプションを、その付与のすべての対象者に提供します。

Amazon Redshift は、複数のロールの付与や、複数のユーザーを付与の対象にすることを含め、ロール割り当てのさまざまな組み合わせをサポートしています。WITH ADMIN オプションは、ユーザーにのみ適用され、ロールには適用されません。同様に、付与の対象者からロールと管理としての承認を削除するには、WITH ADMIN OPTION オプションを指定しながら REVOKE ROLE ステートメントを実行します。ADMIN OPTION を指定した場合には、管理としての承認のみがロールから取り消されます。

次の例では、`sample_role2` ロールでの管理としての承認を `user2` から取り消しています。

```
REVOKE ADMIN OPTION FOR sample_role2 FROM user2;
```

IAM ロールを作成し、ロールの割り当てを管理する方法の詳細については「RBAC でのロールの管理」を参照してください。

Amazon Redshift でのシステム定義のロール

Amazon Redshift には、特定のアクセス許可が定義されたシステム定義のロールがいくつか用意されています。システム固有のロールには `sys:` がプレフィックスされます。適切なアクセス権を持つユーザーのみが、システム定義のロールを変更したり、カスタムなシステム定義のロールを作成したりできます。カスタムなシステム定義のロールでは、`sys:` のプレフィックスを使用することはできません。

次の表に、ロールとそのアクセス許可を一覧でまとめています。

ロール名	説明		
`sys:monitor`	このロールには、カタログまたはシステムテーブルへのアクセス許可が付与されています。		
`sys:operator`	このロールには、カタログまたはシステムテーブルへのアクセス、分析、バキューム処理、またはクエリのキャンセルを		

ロール名	説明
	行うための許可が付与されています。
sys:dba	このロールには、スキーマの作成、テーブルの作成、スキーマの削除、テーブルの削除、およびテーブルの切り捨てを行うための許可が付与されています。これには、ストアドプロシージャの作成または置換、プロシージャの削除、関数の作成または置換、外部関数の作成または置換、ビューの作成、およびビューの削除を行うための許可が付与されています。同時にこのロールでは、sys: operator ロールからすべてのアクセス許可を継承しています。
sys:superuser	このロールには、RBAC でのシステムへのアクセス許可 で定義されているすべてのサポート対象システムアクセス許可があります。

ロール名	説明		
sys:secadmin	• このロールが持つアクセス許可により、ユーザーの作成、ユーザーの変更、ユーザーの削除、ロールの作成、ロールの削除、およびロールの付与を行うことができます。 • このロールには、リレーションの RLS をオンまたはオフにするアクセス許可と、RLS および DDM ポリシー (作成、ドロップ、アタッチ、デタッチ、変更) を管理するアクセス許可があります。また、このロールにはデフォルトで、EXPLAIN RLS、IGNORE RLS、EXPLAIN MASKING アクセス許可が付与されていることにも注意してください。 • このロールは、そのためのアクセス許可が明示的に付与された場合にの		

ロール名	説明		
	み、ユーザのテーブルにアクセスできます。		

データ共有のためのシステム定義のロールとユーザー

Amazon Redshift は、データ共有とデータ共有コンシューマーに対応する内部使用を目的としたロールとユーザーを作成します。各内部ロール名とユーザー名には、予約された名前空間プレフィックス `ds:` が付いています。それには以下の形式があります。

名前	説明		
`ds:`*sharename*	データ共有に対応するシステムロール。		
`ds:`*sharename_consumer*	データ共有コンシューマーに対応するシステムロール。		

データ共有ロールは、データ共有ごとに作成されます。このロールは、データ共有に現在付与されているすべてのアクセス許可を保持しています。データ共有ユーザーは、データ共有コンシューマーごとに作成されます。このユーザーには、単一のデータ共有ロールへのアクセス許可が付与されます。複数のデータ共有に追加されたコンシューマーでは、データ共有ごとにデータ共有ユーザーが作成されます。

データ共有が適切に機能するには、このようなユーザーとロールが必要です。これらのユーザーとロールは変更または削除することはできません。また、顧客が実行するタスクにアクセスしたり、使用したりすることはできません。このような場合、無視しても問題ありません。データ共有の詳細については、「Amazon Redshift でのクラスター間のデータの共有」を参照してください。

> **Note**
>
> `ds:` プレフィックスを使用してユーザー定義のロールまたはユーザーを作成することはできません。

RBAC でのシステムへのアクセス許可

以下は、ロールに対して付与または取り消すことができるシステムへのアクセス許可の一覧です。

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
CREATE ROLE	• スーパーユーザー。 • CREATE ROLE のアクセス許可を持つユーザー。		
DROP ROLE	• スーパーユーザー。 • ロールを作成したユーザー、または WITH ADMIN OPTION の許可を持つロールが付与されたロールの所有者ユーザー。		
「ユーザーの作成」	• スーパーユーザー。 • CREATE USER のアクセス許可を持つユーザー。これらのユーザーはスーパーユーザーを作成できません。		
DROP USER	• スーパーユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
	• DROP USER のアクセス許可を持つユーザー。		
ALTER USER	• スーパーユーザー。 • ALTER USER のアクセス許可を持つユーザー。これらのユーザーは、ユーザーをスーパーユーザーに変更したり、スーパーユーザーをユーザーに変更したりすることはできません。 • 自分のパスワードを変更しようとしている現在のユーザー。		
CREATE SCHEMA	• スーパーユーザー。 • CREATE SCHEMA のアクセス許可を持つユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
DROP SCHEMA	• スーパーユーザー。 • DROP SCHEMA のアクセス許可を持つユーザー。 • スキーマの所有者。		
ALTER DEFAULT PRIVILEGES	• スーパーユーザー。 • ALTER DEFAULT PRIVILEGES のアクセス許可を持つユーザー。 • 独自のデフォルトのアクセス許可を変更しているユーザー。 • 自身によるアクセスが許可されているスキーマでの、アクセス許可を設定しているユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です
ACCESS CATALOG	• スーパーユーザー。 • ACCESS CATALOG アクセス許可を持つユーザー。
ACCESS SYSTEM TABLE	• スーパーユーザー。 • ACCESS SYSTEM TABLE のアクセス許可を持つユーザー。
CREATE TABLE	• スーパーユーザー。 • CREATE TABLE のアクセス許可を持つユーザー。 • スキーマに対し CREATE 関数を実行することが許可されたユーザー。

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
DROP TABLE	• スーパーユーザー。 • DROP TABLE のアクセス許可を持つユーザー。 • スキーマに対し USAGE のアクセス許可を持つテーブル所有者。		
ALTER TABLE	• スーパーユーザー。 • ALTER TABLE のアクセス許可を持つユーザー。 • スキーマに対し USAGE のアクセス許可を持つテーブル所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		

CREATE OR REPLACE FUNCTION	• CREATE FUNCTION の場合:
	  • スーパーユーザー。
	  • CREATE 関数または REPLACE 関数を実行することが許可されたユーザー。
	  • 言語に対し USAGE 関数を実行することが許可されたユーザー。
	• REPLACE FUNCTION の場合:
	  • スーパーユーザー。
	  • CREATE 関数または REPLACE 関数を実行することが許可されたユーザー。
	  • 関数の所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
CREATE OR REPLACE EXTERNAL FUNCTION	• スーパーユーザー。 • CREATE OR REPLACE EXTERNAL FUNCTION のアクセス許可を持つユーザー。		
DROP FUNCTION	• スーパーユーザー。 • DROP FUNCTION のアクセス許可を持つユーザー。 • 関数の所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		

CREATE OR REPLACE PROCEDURE	• CREATE PROCEDURE の場合:
	  • スーパーユーザー。
	  • CREATE プロシージャまたは REPLACE プロシージャを実行することが許可されたユーザー。
	  • 言語に対し USAGE 関数を実行することが許可されたユーザー。
	• REPLACE PROCEDURE の場合:
	  • スーパーユーザー。
	  • CREATE プロシージャまたは REPLACE プロシージャを実行することが許可されたユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
	• プロシージャの所有者。		
DROP PROCEDURE	• スーパーユーザー。 • DROP PROCEDURE のアクセス許可を持つユーザー。 • プロシージャの所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
CREATE OR REPLACE VIEW	• CREATE VIEW の場合: 　• スーパーユーザー。 　• ビューを CREATE もしくは REPLACE することが許可されたユーザー。 　• スキーマに対し CREATE 関数を実行することが許可されたユーザー。 • REPLACE VIEW の場合: 　• スーパーユーザー。 　• ビューを CREATE もしくは REPLACE することが許可されたユーザー。 　• ビューの所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
DROP VIEW	• スーパーユーザー。 • DROP VIEW のアクセス許可を持つユーザー。 • ビューの所有者。		
CREATE MODEL	• スーパーユーザー。 • システムのアクセス許可 CREATE MODEL を持つ (CREATE MODEL における関連付けを読み取ることが可能な) ユーザー。 • CREATE MODEL のアクセス許可を持つユーザー。		
DROP MODEL	• スーパーユーザー。 • DROP MODEL のアクセス許可を持つユーザー。 • モデルの所有者。 • スキーマの所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
CREATE DATASHARE	• スーパーユーザー。 • CREATE DATASHARE のアクセス許可を持つユーザー。 • データベースの所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
ALTER DATASHARE	• スーパーユーザー。 • ALTER DATASHARE のアクセス許可を持つユーザー。 • データ共有に対する ALTER または ALL の許可を持つユーザー • 特定のオブジェクトをデータ共有に追加するユーザーには、そのオブジェクトに対するアクセス許可が必要です。これを行うユーザーは、オブジェクトの所有者であるか、オブジェクトに対する SELECT、USAGE、もしくは ALL のアクセス許可が付与されている必要があります。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
DROP DATASHARE	• スーパーユーザー。 • DROP DATASHARE のアクセス許可を持つユーザー。 • データベースの所有者。		
ライブラリを作成する	• スーパーユーザー。 • CREATE LIBRARY のアクセス許可を持つユーザー、または指定した言語での許可が付与されたユーザー。		
DROP LIBRARY	• スーパーユーザー。 • DROP LIBRARY のアクセス許可を持つユーザー。 • ライブラリの所有者。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
ANALYZE	• スーパーユーザー。 • ANALYZE のアクセス許可を持つユーザー。 • 関連付けの所有者。 • テーブルの共有先であるデータベース所有者。		
CANCEL	• 自身によるクエリをキャンセルしているスーパーユーザー。 • ユーザーによるクエリをキャンセルしているスーパーユーザー。 • CANCEL のアクセス許可を持ち、ユーザーによるクエリをキャンセルしているユーザー。 • 自身によるクエリをキャンセルしているユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
TRUNCATE TABLE	• スーパーユーザー。 • TRUNCATE TABLE のアクセス許可を持つユーザー。 • テーブルの所有者。		
VACUUM	• スーパーユーザー。 • VACUUM のアクセス許可を持つユーザー。 • テーブルの所有者。 • テーブルの共有先であるデータベース所有者。		
IGNORE RLS	• スーパーユーザー。 • `sys:secadmin` ロール内のユーザー。		
EXPLAIN RLS	• スーパーユーザー。 • `sys:secadmin` ロール内のユーザー。		

Command	コマンドを実行するには、次のいずれかの方法によるアクセス許可が必要です		
EXPLAIN MASKING	• スーパーユーザー。 • `sys:secadmin` ロール内のユーザー。		

データベースオブジェクトへのアクセス許可

システムへのアクセス許可とは別に、Amazon Redshift には、データベースオブジェクトへのアクセス許可が含まれており、アクセスのオプションを定義できます。この定義には、テーブルとビューからのデータの読み取り、データの書き込み、テーブルの作成、テーブルの削除など各機能のオプションが含まれます。詳細については、「GRANT」を参照してください。

RBAC を使用すると、システムへのアクセス許可による場合と同様に、データベースオブジェクトへのアクセス許可をロールに割り当てることができます。その後、このロールをユーザーに割り当て、そのユーザーでシステムへのアクセスを認可し、データベースへのアクセスを認可できます。

RBAC での ALTER DEFAULT PRIVILEGES

指定したユーザーによって今後作成されるオブジェクトに対して、デフォルトで適用するアクセス許可のセットを定義するには、ALTER DEFAULT PRIVILEGES ステートメントを使用します。デフォルトでは、ユーザーは自分のデフォルトのアクセス許可のみ変更できます。RBAC では、ロールに対してデフォルトのアクセス許可を設定できます。詳細については、ALTER DEFAULT PRIVILEGES コマンドを参照してください。

RBAC を使用すると、システムへのアクセス許可と同様に、データベースオブジェクトのアクセス許可をロールに割り当てることができます。その上で、ロールをユーザーに割り当てたり、システムおよび (あるいは) データベースに対するアクセスを許可したりできます。

RBAC でのロールの使用に関する考慮事項

RBAC データを使用する際には、以下の点を考慮してください。

- Amazon Redshift では、ロール認証のサイクルは使用できません。r1 を r2 に付与した後に、r2 を r1 に付与することはできません。
- RBAC は、ネイティブの Amazon Redshift オブジェクトと Amazon Redshift Spectrum テーブルの両方で使用できます。
- Amazon Redshift の管理者は、RBAC を有効にして使用開始するために、クラスターを最新のメンテナンスパッチにアップグレードします。
- ロールを作成できるのは、スーパーユーザーと、システムへのアクセス許可 CREATE ROLE を持つユーザーのみです。
- スーパーユーザーとロールの管理者のみが、ロールを変更または削除できます。
- ロール名をユーザー名と同じにすることはできません。
- ロール名に「:/\n」などの無効な文字を含めることはできません。
- ロール名に、PUBLIC などの予約語を使用することはできません。
- ロール名は、デフォルトロールの予約済みプレフィックス sys: で始めることはできません。
- RESTRICT パラメータを持ち、さらに別のロールに付与されているロールは削除できません。デフォルトの設定は RESTRICT です。削除しようとしたロールが別のロールを継承している場合、Amazon Redshift はエラーをスローします。
- ロールに対する管理者としてのアクセス許可を持たないユーザーは、ロールを付与または取り消すことはできません。
- RBAC は、システムテーブルおよびビューでは完全にはサポートされていません。システムテーブルおよびビューの RBAC アクセス許可は、アップグレード、ダウングレード、またはサイズ変更の際に保持されません。Amazon Redshift でのシステム定義のロール を使用して、システムテーブルおよびビューのアクセス許可を管理することをお勧めします。システムテーブルの詳細については、「システムテーブルとビューのリファレンス」を参照してください。

RBAC でのロールの管理

以下のコマンドを使用して、以下の操作を実行します。

- ロールを作成するには、CREATE ROLE コマンドを使用します。
- ロールの名前を変更したり、ロールの所有者を変更するには、ALTER ROLEコマンドを使用します。
- ロールを削除するには、DROP ROLE コマンドを使用します。
- ユーザーにロールを付与するには、GRANT コマンドを使用します。
- ユーザーからロールを取り消すには、REVOKE コマンドを使用します。

- ロールにシステムへのアクセス許可を付与するには、GRANT コマンドを使用します。
- ロールからシステムへのアクセス許可を取り消すには、REVOKE コマンドを使用します。

クラスターまたはワークグループ内のロールのリストを表示するには、「SVV_ROLES」を参照してください。

チュートリアル: RBAC でのロールの作成とクエリ

RBAC では、スーパーユーザーのアクセスが必要となるコマンドを実行するための、アクセス許可を持つロールを作成できます。これらの許可を含むロールで承認されているのであれば、ユーザーはこの種類のコマンドを実行できます。

このチュートリアルでは、作成するデータベースでアクセス許可を管理するのにロールベースのアクセスコントロール (RBAC) を使用できます。次に、データベースに接続し、2 つの異なるロールからデータベースにクエリを実行して RBAC の機能をテストします。

データベースをクエリするために作成して使用する 2 つのロールは `sales_ro` と `sales_rw` です。`sales_ro` ロールを作成し、`sales_ro` ロールを持つユーザーとしてデータをクエリします。`sales_ro` ユーザーは SELECT コマンドのみを使用でき、UPDATE コマンドは使用できません。そのため、`sales_rw` ロールを作成し、`sales_rw` ロールを持つユーザーとしてデータをクエリします。`sales_rw` ユーザーは SELECT コマンドと UPDATE コマンドは使用できます。

また、特定のコマンドへのアクセスを制限するロールを作成し、スーパーユーザーまたはユーザーのいずれかに対して、ロールを割り当てることもできます。

タスク

- ステップ 10: 読み取り/書き込みユーザーとしてテーブルを切り捨てる
- RBAC のシステム関数 (オプション)
- RBAC のシステムビュー (オプション)
- RBAC で行レベルセキュリティを使用する (オプション)

前提条件

- TICKIT サンプルデータベースがロードされた Amazon Redshift クラスターまたはサーバーレスワークグループを作成します。サーバーレスワークグループを作成するには、「Redshift Serverless データウェアハウスの使用を開始」を参照してください。クラスターを作成するには、「Amazon Redshift クラスターのサンプルを作成する」を参照してください。TICKIT サンプルデータの詳細については、「サンプルデータベース」を参照してください。
- スーパーユーザーまたはロール管理者のアクセス許可を持つユーザーへのアクセスがあります。ロールの付与または取り消しを行えるのは、スーパーユーザーまたはロールの管理者のみです。RBAC に必要なアクセス許可の詳細については、「RBAC でのシステムへのアクセス許可」を参照してください。
- 「RBAC でのロールの使用に関する考慮事項」を確認します。

ステップ 1: 管理者ユーザーを作成する

このチュートリアル用にセットアップするには、データベース管理者ロールを作成し、このステップでデータベース管理者ユーザーに割り当てます。データベース管理者はスーパーユーザーまたはロール管理者として作成する必要があります。

Amazon Redshift クエリエディタ v2 のすべてのクエリを実行します。

1. 管理者ロール db_admin を作成するには、次の例を使用します。

```
CREATE ROLE db_admin;
```

2. 次の例を使用して、dbadmin という名前のデータベースユーザーを作成します。

```
CREATE USER dbadmin PASSWORD 'Test12345';
```

3. sys:dba という名前のシステム定義ロールを db_admin ロールに付与するには、次の例を使用します。sys:dba ロールが付与されると、dbadmin ユーザーはスキーマとテーブルを作成できます。詳細については、「Amazon Redshift でのシステム定義のロール」を参照してください。

ステップ 2: スキーマをセットアップする

このステップでは、データベース管理者としてデータベースに接続します。次に、2 つのスキーマを作成し、それらにデータを追加します。

1. クエリエディタ v2 を使用して dbadmin ユーザーとして dev データベースに接続します。データベースへの接続の詳細については、「クエリエディタ v2 の操作」を参照してください。
2. セールスおよびマーケティングのデータベーススキーマを作成するには、次の例を使用します。

```
CREATE SCHEMA sales;
CREATE SCHEMA marketing;
```

3. セールススキーマのテーブルに値を作成して挿入するには、次の例を使用します。

```
CREATE TABLE sales.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO sales.cat(SELECT * FROM category);

CREATE TABLE sales.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
);
INSERT INTO sales.dates(SELECT * FROM date);

CREATE TABLE sales.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
starttime timestamp
);
```

```
INSERT INTO sales.events(SELECT * FROM event);

 CREATE TABLE sales.sale(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO sales.sale(SELECT * FROM sales);
```

4. マーケティングスキーマのテーブルに値を作成して挿入するには、次の例を使用します。

```
CREATE TABLE marketing.cat(
catid smallint,
catgroup varchar(10),
catname varchar(10),
catdesc varchar(50)
);
INSERT INTO marketing.cat(SELECT * FROM category);

CREATE TABLE marketing.dates(
dateid smallint,
caldate date,
day char(3),
week smallint,
month char(5),
qtr char(5),
year smallint,
holiday boolean
);
INSERT INTO marketing.dates(SELECT * FROM date);

CREATE TABLE marketing.events(
eventid integer,
venueid smallint,
catid smallint,
dateid smallint,
eventname varchar(200),
```

```
starttime timestamp
);
INSERT INTO marketing.events(SELECT * FROM event);

CREATE TABLE marketing.sale(
marketingid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp
);
INSERT INTO marketing.sale(SELECT * FROM marketing);
```

ステップ 3: 読み取り専用ユーザーを作成する

このステップでは、読み取り専用ロールとその読み取り専用ロールのセールスアナリストユーザーを作成します。セールスアナリストがコミッションが最も高かったイベントを検索するという割り当てられたタスクを実行するには、セールススキーマのテーブルへの読み取り専用アクセスのみが必要です。

1. dbadmin ユーザーとしてデータベースに接続します。
2. sales_ro ロールを作成するには、次の例を使用します。

```
CREATE ROLE sales_ro;
```

3. セールスアナリストユーザーを作成するには、次の例を使用します。

```
CREATE USER salesanalyst PASSWORD 'Test12345';
```

4. sales_ro ロールに、セールススキーマのオブジェクトの使用と選択のアクセス許可を付与するには、次の例を使用します。

```
GRANT USAGE ON SCHEMA sales TO ROLE sales_ro;
GRANT SELECT ON ALL TABLES IN SCHEMA sales TO ROLE sales_ro;
```

5. セールスアナリストユーザーに sales_ro ロールを付与するには、次の例を使用します。

```
GRANT ROLE sales_ro TO salesanalyst;
```

ステップ 4: 読み取り専用ユーザーとしてデータをクエリする

このステップでは、セールスアナリストユーザーがセールススキーマからデータをクエリします。次に、セールスアナリストユーザーはテーブルの更新とマーケティングスキーマのテーブルの読み取りを試みます。

1. セールスアナリストユーザーとしてデータベースに接続します。
2. 最もコミッションが高い 10 件のセールスを検索するには、次の例を使用します。

```
SET SEARCH_PATH TO sales;
SELECT DISTINCT events.dateid, sale.commission, cat.catname
FROM sale, events, dates, cat
WHERE events.dateid=dates.dateid AND events.dateid=sale.dateid AND events.catid =
 cat.catid
ORDER BY 2 DESC LIMIT 10;

+--------+------------+----------+
| dateid | commission | catname  |
+--------+------------+----------+
|   1880 |     1893.6 | Pop      |
|   1880 |     1893.6 | Opera    |
|   1880 |     1893.6 | Plays    |
|   1880 |     1893.6 | Musicals |
|   1861 |       1500 | Plays    |
|   2003 |       1500 | Pop      |
|   1861 |       1500 | Opera    |
|   2003 |       1500 | Plays    |
|   1861 |       1500 | Musicals |
|   1861 |       1500 | Pop      |
+--------+------------+----------+
```

3. セールススキーマでイベントテーブルから 10 件のイベントを選択するには、次の例を使用します。

```
SELECT * FROM sales.events LIMIT 10;

+---------+---------+-------+--------+--------------------+---------------------+
| eventid | venueid | catid | dateid |     eventname      |      starttime      |
```

```
+---------+---------+-------+--------+--------------------+---------------------+
|    4836 |      73 |     9 |   1871 | Soulfest           | 2008-02-14 19:30:00 |
|    5739 |      41 |     9 |   1871 | Fab Faux           | 2008-02-14 19:30:00 |
|     627 |     229 |     6 |   1872 | High Society       | 2008-02-15 14:00:00 |
|    2563 |     246 |     7 |   1872 | Hamlet             | 2008-02-15 20:00:00 |
|    7703 |      78 |     9 |   1872 | Feist              | 2008-02-15 14:00:00 |
|    7903 |      90 |     9 |   1872 | Little Big Town    | 2008-02-15 19:30:00 |
|    7925 |     101 |     9 |   1872 | Spoon              | 2008-02-15 19:00:00 |
|    8113 |      17 |     9 |   1872 | Santana            | 2008-02-15 15:00:00 |
|     463 |     303 |     8 |   1873 | Tristan und Isolde | 2008-02-16 19:00:00 |
|     613 |     236 |     6 |   1873 | Pal Joey           | 2008-02-16 15:00:00 |
+---------+---------+-------+--------+--------------------+---------------------+
```

4. eventid 1 のイベント名を更新するには、次の例を実行します。この例では、セールスアナリストユーザーはセールススキーマのイベントテーブルに対する SELECT アクセス許可しか持っていないため、「アクセス許可が拒否されました」というエラーが発生します。イベントテーブルを更新するには、sales_ro ロールに UPDATE 権限を付与する必要があります。テーブルを更新するアクセス許可の付与の詳細については、「GRANT の UPDATE パラメータ」を参照してください。UPDATE コマンドの詳細については、「UPDATE」を参照してください。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

5. マーケティングスキーマでイベントテーブルからすべてを選択するには、次の例を使用します。この例では、セールスアナリストユーザーはセールススキーマのイベントテーブルに対する SELECT アクセス許可しか持っていないため、「アクセス許可が拒否されました」というエラーが発生します。マーケティングスキーマのイベントテーブルからデータを選択するには、マーケティングスキーマのイベントテーブルに対する SELECT アクセス許可を sales_ro ロールに付与する必要があります。

```
SELECT * FROM marketing.events;

               ERROR: permission denied for schema marketing
```

ステップ 5: 読み取り/書き込みユーザーを作成する

このステップでは、セールススキーマのデータ処理のための抽出、変換、ロード (ETL) パイプラインの構築を担当するセールスエンジニアに読み取り専用アクセスが与えられますが、タスクを実行するための読み取り/書き込みアクセスが後に付与されます。

1. dbadmin ユーザーとしてデータベースに接続します。
2. セールススキーマで sales_rw ロールを作成するには、次の例を使用します。

```
CREATE ROLE sales_rw;
```

3. セールスエンジニアユーザーを作成するには、次の例を使用します。

```
CREATE USER salesengineer PASSWORD 'Test12345';
```

4. sales_ro ロールを割り当てることで、sales_rw ロールにセールススキーマのオブジェクトの使用と選択のアクセス許可を付与するには、次の例を使用します。Amazon Redshift でロールがアクセス許可を継承する方法の詳細については、「ロール階層」を参照してください。

```
GRANT ROLE sales_ro TO ROLE sales_rw;
```

5. セールスエンジニアユーザーに sales_rw ロールを付与するには、次の例を使用します。

```
GRANT ROLE sales_rw TO salesengineer;
```

ステップ 6: 継承された読み取り専用ロールのあるユーザーとしてデータをクエリする

このステップでは、セールスエンジニアユーザーが読み取りアクセス許可が付与される前にイベントテーブルを更新しようとします。

1. セールスエンジニアユーザーとしてデータベースに接続します。
2. セールスエンジニアユーザーは、セールススキーマのイベントテーブルからデータを正常に読み取ることができます。セールススキーマでイベントテーブルから eventid 1 のイベントを選択するには、次の例を使用します。

```
SELECT * FROM sales.events where eventid=1;

+---------+---------+-------+--------+-----------------+---------------------+
| eventid | venueid | catid | dateid |    eventname    |      starttime      |
```

```
+---------+---------+-------+--------+-----------------+---------------------+
|       1 |     305 |     8 |   1851 | Gotterdammerung | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+-----------------+---------------------+
```

3. マーケティングスキーマでイベントテーブルからすべてを選択するには、次の例を使用します。セールスエンジニアユーザーにはマーケティングスキーマのテーブルに対するアクセス許可がないため、このクエリは結果としてアクセス許可拒否エラーになります。マーケティングスキーマのイベントテーブルからデータを選択するには、マーケティングスキーマのイベントテーブルに対する SELECT アクセス許可を sales_rw ロールに付与する必要があります。

```
SELECT * FROM marketing.events;

ERROR: permission denied for schema marketing
```

4. eventid 1 のイベント名を更新するには、次の例を実行します。この例では、セールスエンジニアユーザーにはセールススキーマのイベントテーブルに対する選択アクセス許可しかないため、アクセス許可拒否エラーが発生します。イベントテーブルを更新するには、sales_rw ロールに UPDATE のアクセス許可を付与する必要があります。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;

ERROR: permission denied for relation events
```

ステップ 7: 読み取り/書き込みロールに更新および挿入アクセス許可を付与する

sales_rw ロールに対するアクセス許可を更新して挿入します。

1. dbadmin ユーザーとしてデータベースに接続します。
2. sales_rw ロールに UPDATE、INSERT、DELETE アクセス許可を付与するには、次の例を使用します。

```
GRANT UPDATE, INSERT, ON ALL TABLES IN SCHEMA sales TO role sales_rw;
```

ステップ 8: 読み取り/書き込みユーザーとしてデータをクエリする

このステップでは、各自のロールに挿入アクセス許可と更新アクセス許可が付与された後に、セールスエンジニアがテーブルを正常に更新します。次に、セールスエンジニアはイベントテーブルの分析とバキュームを試みますが、失敗します。

1. セールスエンジニアユーザーとしてデータベースに接続します。
2. eventid 1 のイベント名を更新するには、次の例を実行します。

```
UPDATE sales.events
SET eventname = 'Comment event'
WHERE eventid = 1;
```

3. 前のクエリで行った変更を表示するには、次の例を使用してセールススキーマでイベントテーブルから eventid 1 のイベントを選択します。

```
SELECT * FROM sales.events WHERE eventid=1;

+---------+---------+-------+--------+---------------+---------------------+
| eventid | venueid | catid | dateid |   eventname   |      starttime      |
+---------+---------+-------+--------+---------------+---------------------+
|       1 |     305 |     8 |   1851 | Comment event | 2008-01-25 14:30:00 |
+---------+---------+-------+--------+---------------+---------------------+
```

4. セールススキーマで更新されたイベントテーブルを分析するには、次の例を使用します。この例では、セールスエンジニアユーザーに必要なアクセス許可がなく、セールススキーマのイベントテーブルの所有者でもないため、アクセス許可拒否エラーが発生します。イベントテーブルを分析するには、GRANT コマンドを使用して ANALYZE のアクセス許可を sales_rw ロールに付与する必要があります。ANALYZE コマンドの詳細については、「ANALYZE」を参照してください。

```
ANALYZE sales.events;

               ERROR: skipping "events" --- only table or database owner can analyze
```

5. 更新されたイベントテーブルをバキュームするには、次の例を使用します。この例では、セールスエンジニアユーザーに必要なアクセス許可がなく、セールススキーマのイベントテーブルの所有者でもないため、アクセス許可拒否エラーが発生します。イベントテーブルをバキュームするには、GRANT コマンドを使用して VACUUM のアクセス許可を sales_rw ロールに付与する必要があります。VACUUM コマンドの詳細については、「VACUUM」を参照してください。

```
VACUUM sales.events;

ERROR: skipping "events" --- only table or database owner can vacuum it
```

ステップ 9: 管理者ユーザーとしてデータベース内のテーブルを分析してバキュームする

このステップでは、dbadmin ユーザーがすべてのテーブルを分析してバキュームします。ユーザーにはこのデータベースに対する管理者アクセス許可があるため、これらのコマンドを実行できます。

1. dbadmin ユーザーとしてデータベースに接続します。
2. セールススキーマでイベントテーブルを分析するには、次の例を使用します。

```
ANALYZE sales.events;
```

3. セールススキーマでイベントテーブルをバキュームするには、次の例を使用します。

```
VACUUM sales.events;
```

4. マーケティングスキーマでイベントテーブルを分析するには、次の例を使用します。

```
ANALYZE marketing.events;
```

5. マーケティングスキーマでイベントテーブルをバキュームするには、次の例を使用します。

```
VACUUM marketing.events;
```

ステップ 10: 読み取り/書き込みユーザーとしてテーブルを切り捨てる

このステップでは、セールスエンジニアユーザーがセールススキーマのイベントテーブルを切り捨てようとしますが、dbadmin ユーザーから切り捨てアクセス許可が付与された場合にのみ成功します。

1. セールスエンジニアユーザーとしてデータベースに接続します。
2. セールススキーマのイベントテーブルからすべての行を削除するには、次の例を使用します。この例では、セールスエンジニアユーザーに必要なアクセス許可がなく、セールススキーマのイベントテーブルの所有者でもないため、エラーが発生します。イベントテーブルを切り捨てるに

は、GRANT コマンドを使用して TRUNCATE のアクセス許可を sales_rw ロールに付与する必要があります。TRUNCATE コマンドの詳細については、「TRUNCATE」を参照してください。

```
TRUNCATE sales.events;

ERROR: must be owner of relation events
```

3. dbadmin ユーザーとしてデータベースに接続します。
4. テーブル切り捨て特権を sales_rw ロールを付与するには、次の例を使用します。

```
GRANT TRUNCATE TABLE TO role sales_rw;
```

5. クエリエディタ v2 を使用してセールスエンジニアユーザーとしてデータベースに接続します。
6. セールススキーマでイベントテーブルから 10 件のイベントを読み取るには、次の例を使用します。

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+-----------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname           |      starttime
    |
+---------+---------+-------+--------+-----------------------------
+---------------------+
|       1 |     305 |     8 |   1851 | Comment event               | 2008-01-25
 14:30:00 |
|       2 |     306 |     8 |   2114 | Boris Godunov               | 2008-10-15
 20:00:00 |
|       3 |     302 |     8 |   1935 | Salome                      | 2008-04-19
 14:30:00 |
|       4 |     309 |     8 |   2090 | La Cenerentola (Cinderella) | 2008-09-21
 14:30:00 |
|       5 |     302 |     8 |   1982 | Il Trovatore                | 2008-06-05
 19:00:00 |
|       6 |     308 |     8 |   2109 | L Elisir d Amore            | 2008-10-10
 19:30:00 |
|       7 |     309 |     8 |   1891 | Doctor Atomic               | 2008-03-06
 14:00:00 |
|       8 |     302 |     8 |   1832 | The Magic Flute             | 2008-01-06
 20:00:00 |
|       9 |     308 |     8 |   2087 | The Fly                     | 2008-09-18
 19:30:00 |
```

```
|      10 |     305 |     8 |   2079 | Rigoletto                     | 2008-09-10
 15:00:00 |
+---------+---------+-------+--------+-----------------------------
+---------------------+
```

7. セールススキーマでイベントテーブルを切り捨てるには、次の例を使用します。

```
TRUNCATE sales.events;
```

8. セールススキーマで更新されたイベントテーブルからデータを読み取るには、次の例を使用します。

```
SELECT * FROM sales.events ORDER BY eventid LIMIT 10;

+---------+---------+-------+--------+-----------------------------
+---------------------+
| eventid | venueid | catid | dateid |          eventname          |      starttime
    |
+---------+---------+-------+--------+-----------------------------
+---------------------+
```

マーケティングスキーマに対して読み取り専用ロールおよび読み取り/書き込みロールを作成する (オプション)

このステップでは、マーケティングスキーマに対して読み取り専用ロールおよび読み取り/書き込みロールを作成します。

1. dbadmin ユーザーとしてデータベースに接続します。
2. マーケティングスキーマの読み取り専用ロールと読み取り/書き込みロールを作成するには、次の例を使用します。

```
CREATE ROLE marketing_ro;

CREATE ROLE marketing_rw;

GRANT USAGE ON SCHEMA marketing TO ROLE marketing_ro, ROLE marketing_rw;

GRANT SELECT ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_ro;

GRANT ROLE marketing_ro TO ROLE marketing_rw;
```

```
GRANT INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA marketing TO ROLE marketing_rw;

CREATE USER marketinganalyst PASSWORD 'Test12345';

CREATE USER marketingengineer PASSWORD 'Test12345';

GRANT ROLE marketing_ro TO marketinganalyst;

GRANT ROLE marketing_rw TO marketingengineer;
```

RBAC のシステム関数 (オプション)

Amazon Redshift には、追加のグループまたは role_is_member_of ロールおよび user_is_member_of ロールのユーザーメンバーシップおよびロールメンバーシップに関する情報を提供します。これらの関数は、スーパーユーザーと一般ユーザーが利用できます。スーパーユーザーはすべてのロールメンバーシップを確認できます。一般ユーザーは、アクセスが付与されているロールのメンバーシップのみを確認できます。

role_is_member_of 関数を使用するには

1. セールスエンジニアユーザーとしてデータベースに接続します。
2. sales_rw ロールが sales_ro ロールのメンバーかどうかを確認するには、次の例を使用します。

```
SELECT role_is_member_of('sales_rw', 'sales_ro');

+-------------------+
| role_is_member_of |
+-------------------+
| true              |
+-------------------+
```

3. sales_ro ロールが sales_rw ロールのメンバーかどうかを確認するには、次の例を使用します。

```
SELECT role_is_member_of('sales_ro', 'sales_rw');

+-------------------+
| role_is_member_of |
+-------------------+
| false             |
+-------------------+
```

user_is_member_of 関数を使用するには

1. セールスエンジニアユーザーとしてデータベースに接続します。
2. 次の例では、セールスアナリストユーザーのユーザーメンバーシップを確認しようとしています。セールスエンジニアはセールスアナリストへのアクセスがないため、このクエリはエラーになります。このコマンドを正常に実行するには、セールスアナリストユーザーとしてデータベースに接続し、例を使用します。

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');

ERROR
```

3. スーパーユーザーとしてデータベースに接続します。
4. スーパーユーザーとして接続しているときにセールスアナリストユーザーのメンバーシップを確認するには、次の例を使用します。

```
SELECT user_is_member_of('salesanalyst', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+
```

5. dbadmin ユーザーとしてデータベースに接続します。
6. セールスエンジニアのメンバーシップを確認するには、次の例を使用します。

```
SELECT user_is_member_of('salesengineer', 'sales_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| true              |
+-------------------+

SELECT user_is_member_of('salesengineer', 'marketing_ro');

+-------------------+
| user_is_member_of |
+-------------------+
| false             |
```

```
+--------------------+

SELECT user_is_member_of('marketinganalyst', 'sales_ro');

+--------------------+
| user_is_member_of |
+--------------------+
| false             |
+--------------------+
```

RBAC のシステムビュー (オプション)

ロール、ユーザーへのロールの割り当て、ロール階層、およびロールによるデータベースオブジェクトの特権を表示するには、Amazon Redshift のシステムビューを使用します。これらのビューは、スーパーユーザーと一般ユーザーが利用できます。スーパーユーザーはすべてのロールの詳細を確認できます。一般ユーザーは、アクセスが付与されているロールの詳細のみを確認できます。

1. クラスターで明示的にロールが付与されたユーザーの一覧を表示するには、次の例を使用します。

```
SELECT * FROM svv_user_grants;
```

2. クラスターで明示的にロールが付与されたロールの一覧を表示するには、次の例を使用します。

```
SELECT * FROM svv_role_grants;
```

システムビューの全リストについては、「SVV メタデータビュー」を参照してください。

RBAC で行レベルセキュリティを使用する (オプション)

機密データに対するきめ細かなアクセス制御を行うには、行レベルセキュリティ (RLS) を使用します。RLS の詳細については、「行レベルのセキュリティ」を参照してください。

このセクションでは、、メジャーリーグベースボールの catdesc 値を持つ cat テーブル内の行のみを表示する salesengineer アクセス許可をユーザーに付与する RLS ポリシーを作成します。次に、salesengineer ユーザーとしてデータベースにクエリを実行します。

1. salesengineer ユーザーとしてデータベースに接続します。
2. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

3. dbadmin ユーザーとしてデータベースに接続します。
4. cat テーブル内の catdesc 列の RLS ポリシーを作成するには、次の例を使用します。

```
CREATE RLS POLICY policy_mlb_engineer
WITH (catdesc VARCHAR(50))
USING (catdesc = 'Major League Baseball');
```

5. RLS ポリシーを sales_rw ロールにアタッチするには、次の例を使用します。

```
ATTACH RLS POLICY policy_mlb_engineer ON sales.cat TO ROLE sales_rw;
```

6. RLS を有効にするようにテーブルを変更するには、次の例を使用します。

```
ALTER TABLE sales.cat ROW LEVEL SECURITY ON;
```

7. salesengineer ユーザーとしてデータベースに接続します。
8. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。catdesc 列が Major League Baseball のときのみエントリが表示されることに注意してください。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+-----------------------+
| catid | catgroup | catname |        catdesc        |
```

```
+-------+----------+---------+-----------------------+
|     1 | Sports   | MLB     | Major League Baseball |
+-------+----------+---------+-----------------------+
```

9. salesanalyst ユーザーとしてデータベースに接続します。
10. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。デフォルトのすべて拒否ポリシーが適用されているため、エントリは表示されないことに注意してください。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+-----------------------+
| catid | catgroup | catname |        catdesc        |
+-------+----------+---------+-----------------------+
```

11. dbadmin ユーザーとしてデータベースに接続します。
12. IGNORE RLS アクセス許可を sales_ro ロールに付与するには、次の例を使用します。これにより、salesanalyst ユーザーは sales_ro ロールのメンバーであるため、RLS ポリシーを無視するアクセス許可が付与されます。

```
GRANT IGNORE RLS TO ROLE sales_ro;
```

13. salesanalyst ユーザーとしてデータベースに接続します。
14. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
|     1 | Sports   | MLB     | Major League Baseball           |
|     2 | Sports   | NHL     | National Hockey League          |
|     3 | Sports   | NFL     | National Football League        |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer             |
+-------+----------+---------+---------------------------------+
```

15. dbadmin ユーザーとしてデータベースに接続します。
16. sales_ro ロールから IGNORE RLS アクセス許可を取り消すには、次の例を使用してください。

```
REVOKE IGNORE RLS FROM ROLE sales_ro;
```

17. salesanalyst ユーザーとしてデータベースに接続します。
18. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。デフォルトのすべて拒否ポリシーが適用されているため、エントリは表示されないことに注意してください。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+---------------------+
| catid | catgroup | catname |       catdesc       |
+-------+----------+---------+---------------------+
```

19. dbadmin ユーザーとしてデータベースに接続します。
20. RLS ポリシーを cat テーブルから切り離すには、次の例を使用します。

```
DETACH RLS POLICY policy_mlb_engineer ON cat FROM ROLE sales_rw;
```

21. salesanalyst ユーザーとしてデータベースに接続します。
22. cat テーブルの最初の 5 エントリを表示するには、次の例を使用します。デフォルトのすべて拒否ポリシーが適用されているため、エントリは表示されないことに注意してください。

```
SELECT *
FROM sales.cat
ORDER BY catid ASC
LIMIT 5;

+-------+----------+---------+----------------------------------+
| catid | catgroup | catname |             catdesc              |
+-------+----------+---------+----------------------------------+
|     1 | Sports   | MLB     | Major League Baseball            |
|     2 | Sports   | NHL     | National Hockey League           |
|     3 | Sports   | NFL     | National Football League         |
|     4 | Sports   | NBA     | National Basketball Association |
|     5 | Sports   | MLS     | Major League Soccer              |
```

```
+--------+----------+---------+--------------------------------+
```

23.dbadmin ユーザーとしてデータベースに接続します。

24.RLS ポリシーをドロップするには、次の例を使用します。

```
DROP RLS POLICY policy_mlb_engineer;
```

25.RLS を削除するには、次の例を使用します。

```
ALTER TABLE cat ROW LEVEL SECURITY OFF;
```

関連トピック

RBAC の詳細については、次のドキュメントを参照してください。

- ロール階層
- ロールの割り当て
- データベースオブジェクトへのアクセス許可
- RBAC での ALTER DEFAULT PRIVILEGES

行レベルのセキュリティ

Amazon Redshift で行レベルセキュリティ (RLS) を使用すると、機密データに対するきめ細かなアクセス制御を行うことができます。データベースオブジェクトレベルで定義されたセキュリティポリシーに基づいて、スキーマまたはテーブル内の特定のデータレコードにアクセスできるユーザーまたはロールを選択できます。列のサブセットに対する許可をユーザーに付与できる列レベルのセキュリティに加えて、RLS ポリシーを使用して表示されている列の特定の行に対するアクセスをさらに制限します。行レベルのセキュリティの詳細については、「列レベルのアクセスコントロールの使用上の注意」を参照してください。

テーブルに RLS ポリシーを実行するとき、ユーザーがクエリを実行したときに返される結果セットを制限できます。

RLS ポリシーを作成するとき、Amazon Redshift がクエリのテーブル内にある既存の行を返すかどうか決定する式を指定できます。アクセスを制限する RLS ポリシーを作成することで、クエリにさらなる条件を追加または外在化する必要がなくなります。

RLS ポリシーを作成するとき、単純なポリシーを作成して、ポリシーの複雑なステートメントを避けることをお勧めします。RLS ポリシーを定義するとき、ポリシーに基づくポリシー定義で過度なテーブル結合を使用しないでください。

ポリシーがルックアップテーブルを参照するとき、ポリシーが存在するテーブルに加え、Amazon Redshift は追加のテーブルをスキャンします。RLS ポリシーがアタッチされているユーザーと、ポリシーがアタッチされていないユーザーでは、同じクエリのパフォーマンスに違いが発生します。

SQL ステートメントで RLS ポリシーの使用

SQL ステートメントで RLS ポリシーを使用するとき、Amazon Redshift は次のルールを適用します。

- Amazon Redshift は、デフォルトで SELECT、UPDATE、DELETE ステートメントに RLS ポリシーを適用します。
- SELECT と UNLOAD の場合、Amazon Redshift は定義されたポリシーに従って行をフィルタリングします。
- UPDATE の場合、Amazon Redshift は表示されている行のみを更新します。ポリシーがテーブルの行のサブセットを制限している場合、更新することはできません。
- DELETE の場合、表示されている行のみを削除できます。ポリシーがテーブル内の行のサブセットを制限している場合、削除することはできません。TRUNCATE の場合でも、テーブルを切り捨てることができます。
- CREATE TABLE LIKE の場合、LIKE オプションで作成されたテーブルはソーステーブルから許可設定を継承しません。同様に、ターゲットテーブルはソーステーブルから RLS ポリシーを継承しません。

ユーザーごとに複数ポリシーの組み合わせ

Amazon Redshift の RLS は、ユーザーとオブジェクトごとに複数ポリシーのアタッチをサポートしています。1 人のユーザーに対して複数ポリシーが定義されていると、Amazon Redshift は、テーブル上の RLS CONJUNCTION TYPE に応じて、AND 構文または OR 構文ですべてのポリシーを適用します。設定タイプの詳細については、「ALTER TABLE」を参照してください。

テーブルの複数ポリシーをお客様に関連付けることができます。複数のポリシーが直接お客様にアタッチされているか、複数のロールに属していて、ロールに異なるポリシーがアタッチされています。

複数のポリシーで特定の関係の行アクセスを制限する必要がある場合、関係の RLS CONJUNCTION TYPE を AND に設定できます。次の例を考えます。Alice は、指定されたポリシーで「キャットネーム」が「NBA」になっているスポーツイベントしか見ることができません。

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Create an RLS policy that only lets the user see NBA.
CREATE RLS POLICY policy_nba
WITH (catname VARCHAR(10))
USING (catname = 'NBA');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;
ATTACH RLS POLICY policy_nba ON category TO ROLE analyst;

-- Activate RLS on the category table with AND CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;

-- Select all from the category table.
SELECT catgroup, catname
FROM category;

 catgroup | catname
---------+---------
 Sports   | NBA
(1 row)
```

複数のポリシーによって、ユーザーが特定の関係で、より多くの行を表示できるようにする必要がある場合、ユーザーはそのリレーションの RLS CONJUNCTION TYPE を OR に設定できます。次の例を考えます。Alice は、指定されたポリシーに従って「コンサート」と「スポーツ」しか表示できません。

```
-- Create an analyst role and grant it to a user named Alice.
CREATE ROLE analyst;
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
GRANT ROLE analyst TO alice;

-- Create an RLS policy that only lets the user see concerts.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');

-- Create an RLS policy that only lets the user see sports.
CREATE RLS POLICY policy_sports
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Sports');

-- Attach both to the analyst role.
ATTACH RLS POLICY policy_concerts ON category TO ROLE analyst;
ATTACH RLS POLICY policy_sports ON category TO ROLE analyst;

-- Activate RLS on the category table with OR CONJUNCTION TYPE.
ALTER TABLE category ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;

-- Change session to Alice.
SET SESSION AUTHORIZATION alice;

-- Select all from the category table.
SELECT catgroup, count(*)
FROM category
GROUP BY catgroup ORDER BY catgroup;

 catgroup | count
---------+-------
 Concerts |  3
 Sports   |  5
(2 rows)
```

RLS ポリシーの所有権と管理

スーパーユーザー、セキュリティ管理者、sys:secadmin ロールを持つユーザーのいずれかとして、テーブルのすべての RLS ポリシーを作成、修正、管理ができます。オブジェクトレベルでは、テーブルのスキーマ定義を修正せずに行レベルのセキュリティをオンまたはオフに切り替えることができます。

行レベルのセキュリティを開始するには、次の SQL ステートメントを使用できます。

- ALTER TABLE ステートメントを使用して、テーブルの RLS をオンまたはオフに切り替えます。詳細については、「ALTER TABLE」を参照してください。
- CREATE RLS POLICY ステートメントを使用し、1 つ以上のテーブルのセキュリティポリシーを作成して、ポリシーに 1 つ以上のユーザーまたはロールを指定します。

  詳細については、「CREATE RLS POLICY」を参照してください。
- ALTER RLS POLICY ステートメントを使用して、ポリシー定義の変更など、ポリシーを変更します。同じポリシーを複数のテーブルまたはビューに使用できます。

  詳細については、「RLS ポリシーの変更」を参照してください。
- ATTACH RLS POLICY ステートメントを使用して、1 つ以上の関係、1 つ以上のユーザー、ロールのいずれかにポリシーをアタッチします。

  詳細については、「ATTACH RLS POLICY 」を参照してください。
- DETACH RLS POLICY ステートメントを使用して、1 つ以上の関係、1 つ以上のユーザー、ロールのいずれかからポリシーをデタッチします。

  詳細については、「DETACH RLS POLICY 」を参照してください。
- DROP RLS POLICY ステートメントを使用して、ポリシーを削除します。

  詳細については、「DROP RLS POLICY 」を参照してください。
- GRANT と REVOKE ステートメントを使用して、ルックアップテーブルを参照する RLS ポリシーの SELECT 許可を明示的に付与と取り消しを行います。詳細については、GRANTおよびREVOKEを参照してください。

作成されたポリシーをモニタリングするため、sys:secadmin は SVV_RLS_POLICY と SVV_RLS_ATTACHED_POLICY を確認できます。

RLS で保護された関係を一覧表示するため、sys:secadmin は SVV_RLS_RELATION を確認できます。

RLS で保護された関係を参照するクエリの RLS ポリシーの適用を追跡するため、スーパーユーザー、sys:operator、ACCESS SYSTEM TABLE のシステム許可を持つ任意のユーザーのいずれかは SVV_RLS_APPLIED_POLICY を確認できます。sys:secadmin はデフォルトでこれらの許可が付与されていないことに注意してください。

RLS ポリシーがアタッチされているテーブルを照合しても、表示されないようにするには、任意のユーザーに IGNORE RLS 許可を付与できます。スーパーユーザーまたは sys:secadmin であるユーザーには、IGNORE RLS 許可が自動的に付与されます。詳細については、「GRANT」を参照してください。

RLS 関連のクエリをトラブルシューティングするために EXPLAIN プランのクエリの RLS ポリシーフィルタを説明するには、EXPLAIN RLS 許可を任意のユーザーに付与できます。詳細については、GRANTおよびEXPLAINを参照してください。

ポリシーに依存するオブジェクトと原則

アプリケーションにセキュリティを提供し、ポリシーオブジェクトが古くなったり、無効になったりすることを防止するため、Amazon Redshift は RLS ポリシーによって参照されるオブジェクトの削除や変更を許可していません。

以下のものは、Amazon Redshift が RLS ポリシー用にトラッキングするスキーマオブジェクトの従属関係を一覧表示します。

- ターゲットテーブルのスキーマオブジェクトの従属関係をトラッキングするとき、Amazon Redshift は次のルールに従います。
  - ターゲットテーブルを削除するとき、Amazon Redshift は関係、ユーザー、ロール、パブリックからポリシーをデタッチします。
  - ターゲットテーブル名の名前を変更しても、アタッチされたポリシーには影響しません。
  - 最初にポリシーを削除またはデタッチしない限り、ポリシー定義内で参照対象のターゲットテーブルの列を削除することはできません。これは、CASCADE オプションが指定されている場合にも当てはまります。ターゲットテーブルの他の列を削除できます。
  - ターゲットテーブルの参照列の名前を変更することはできません。参照列の名前を変更するには、まずポリシーをデタッチします。これは、CASCADE オプションが指定されている場合にも当てはまります。
  - CASCADE オプションを指定しても、参照列のタイプを変更できません。
- ルックアップテーブルのスキーマオブジェクトの依存をトラッキングするとき、Amazon Redshift は次のルールに従います。
  - ルックアップテーブルは削除できません。ルックアップテーブルを削除するには、まずルックアップテーブルが参照対象のポリシーを削除します。
  - ルックアップテーブルの名前を変更することはできません。ルックアップテーブルの名前を変更するには、まずルックアップテーブルが参照対象のポリシーを削除します。これは、CASCADE オプションが指定されている場合にも当てはまります。

  - ポリシー定義で使用されているルックアップテーブルの列は削除できません。ポリシー定義で使用されているルックアップテーブルの列を削除するには、まずルックアップテーブルが参照対象のポリシーを削除します。これは、ALTER TABLE DROP COLUMN ステートメントで CASCADE オプションが指定されているときにも適用されます。ルックアップテーブルの他の列を削除できます。
  - ルックアップテーブルの参照列の名前を変更することはできません。参照列の名前を変更するには、まずルックアップテーブルの参照対象となるポリシーを削除します。これは、CASCADE オプションが指定されている場合にも当てはまります。
  - 参照列のタイプを変更することはできません。
- ユーザーまたはロールを削除すると、Amazon Redshift はそのユーザーまたはロールにアタッチされているすべてのポリシーを自動的にデタッチします。
- DROP SCHEMA ステートメントで CASCADE オプションを使用するとき、Amazon Redshift はスキーマの関係も削除します。削除されたスキーマの関係に依存する他のスキーマの関係もすべて削除します。ポリシーのルックアップテーブルである関係の場合、Amazon Redshift は DROP SCHEMA DDL に失敗します。DROP SCHEMA ステートメントによって削除された関係の場合、Amazon Redshift はそれらの関係にアタッチされているすべてのポリシーをデタッチします。
- ルックアップ関数 (ポリシー定義内で参照される関数) は、ポリシーも削除する場合に限り、削除できます。これは、CASCADE オプションが指定されている場合にも当てはまります。
- ポリシーがテーブルにアタッチされると、Amazon Redshift はこのテーブルが別のポリシーのルックアップテーブルであるかどうか確認します。この場合、Amazon Redshift はこのテーブルにポリシーのアタッチを許可しません。
- RLS ポリシーの作成時に、Amazon Redshift はこのテーブルが他の RLS ポリシーのターゲットテーブルであるかどうか確認します。この場合、Amazon Redshift はこのテーブルにポリシーの作成を許可しません。

例

次の例では、スキーマの依存がどのようにトラッキングされるかについて示しています。

```
-- The CREATE and ATTACH policy statements for `policy_events` references some
-- target and lookup tables.
-- Target tables are tickit_event_redshift and target_schema.target_event_table.
-- Lookup table is tickit_sales_redshift.
-- Policy `policy_events` has following dependencies:
--   table tickit_sales_redshift column eventid, qtysold
--   table tickit_event_redshift column eventid
```

```
--   table target_event_table column eventid
--   schema public and target_schema
CREATE RLS POLICY policy_events
WITH (eventid INTEGER)
USING (
    eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;

ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;
```

RLS ポリシーを使用する際の考慮事項と制限事項

考慮事項

RLS ポリシーを操作する際の考慮事項は次のとおりです。

- Amazon Redshift は、RLS ポリシーを SELECT、UPDATE、DELETE ステートメントに適用します。
- Amazon Redshift は、RLS ポリシーを INSERT、COPY、ALTER TABLE APPEND ステートメントに適用しません。
- 行レベルのセキュリティは、列レベルのセキュリティと連携してデータを保護します。
- Amazon Redshift クラスターが、RLS をサポートする最新の一般公開バージョンに含まれていたが、以前のバージョンにダウングレードされている場合、RLS ポリシーがアタッチされたベーステーブルでクエリを実行すると、Amazon Redshift はエラーを返します。sys:secadmin は、制限されたポリシーを付与されたユーザーからのアクセスの取り消し、テーブルの RLS をオフに切り替え、ポリシーの削除ができます。
- ソース関係で RLS がオンになっているとき、Amazon Redshift は、スーパーユーザー、IGNORE RLS のシステムアクセス許可を明示的に付与されたユーザー、sys:secadmin ロールに対して ALTER TABLE APPEND ステートメントをサポートします。この場合、ALTER TABLE APPEND ステートメントを実行して、既存のソーステーブルからデータを移動することにより、ターゲットテーブルに行を追加できます。Amazon Redshift は、すべてのタプルをソース関係からターゲット関係に移動します。ターゲット関係の RLS ステータスは、ALTER TABLE APPEND ステートメントには影響しません。
- 他のデータウェアハウスシステムからの移行を容易にするには、変数名と値を指定することにより、接続用にカスタマイズされたセッションコンテキスト変数を設定と取得できます。

次の例では、行レベルセキュリティ (RLS) ポリシーのセッションコンテキスト変数を設定します。

```
-- Set a customized context variable.
SELECT set_config('app.category', 'Concerts', FALSE);

-- Create a RLS policy using current_setting() to get the value of a customized
 context variable.
CREATE RLS POLICY policy_categories
WITH (catgroup VARCHAR(10))
USING (catgroup = current_setting('app.category', FALSE));

-- Set correct roles and attach the policy on the target table to one or more roles.
ATTACH RLS POLICY policy_categories ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

カスタマイズされたセッションコンテキスト変数の設定と取得の詳細については、「SET」、「SET_CONFIG」、「SHOW」、「CURRENT_SETTING」、「RESET」を参照してください。サーバー設定変更全般の詳細については、「サーバー設定の変更」を参照してください。

> **Important**
>
> RLS ポリシー内でセッションコンテキスト変数を使用する場合、セキュリティポリシーはポリシーを呼び出すユーザーまたはロールに依存します。RLS ポリシーでセッションコンテキスト変数を使用する場合は、セキュリティの脆弱性を避けるように注意してください。

- DECLARE と FETCH の間、または後続の FETCH ステートメントの間に SET SESSION AUTHORIZATION を使用してセッションユーザーを変更しても、DECLARE 時のユーザーポリシーに基づいて既に準備されているプランは更新されません。RLS で保護されたテーブルでカーソルを使用する場合は、セッションユーザーを変更しないでください。
- ビューオブジェクト内のベースオブジェクトが RLS で保護されている場合、クエリを実行しているユーザーにアタッチされたポリシーがそれぞれのベースオブジェクトに適用されます。これは、ビュー所有者の権限がビューベースオブジェクトと照合されるオブジェクトレベルのアクセス許可チェックとは異なります。クエリの RLS で保護されたリレーションは、EXPLAIN プランの出力で確認できます。
- ユーザー定義関数 (UDF) がユーザーに関連付けられたリレーションの RLS ポリシーで参照される場合、リレーションをクエリするには UDF に対する EXECUTE 権限が必要です。

- 行レベルのセキュリティは、クエリの最適化を制限する可能性があります。RLS で保護されたビューを大きなデータセットにデプロイする前に、クエリのパフォーマンスを慎重に評価することをお勧めします。
- 遅延バインディングビューに適用される行レベルのセキュリティポリシーは、フェデレーションテーブルにプッシュされる可能性があります。これらの RLS ポリシーは、外部の処理エンジンログに表示される場合があります。

制約事項

RLS ポリシーを操作する場合の制限事項は次のとおりです。

- Amazon Redshift は、複雑な結合を持つルックアップを備える特定の RLS ポリシーの SELECT ステートメントをサポートしていますが、UPDATE または DELETE ステートメントはサポートしていません。UPDATE または DELETE ステートメントの場合、Amazon Redshift は次のエラーを返します。

```
ERROR: One of the RLS policies on target relation is not supported in UPDATE/DELETE.
```

- ユーザー定義関数 (UDF) がユーザーに関連付けられたリレーションの RLS ポリシーで参照される場合は常に、リレーションをクエリするには UDF に対する EXECUTE 権限が必要です。
- 相関サブクエリはサポートされていません。Amazon Redshift は次のエラーを返します。

```
ERROR: RLS policy could not be rewritten.
```

- RLS ポリシーは、外部テーブルにはアタッチできません。
- Amazon Redshift は RLS とのデータの共有をサポートしていません。リレーションでデータ共有の RLS がオフになっていない場合、クエリはコンシューマークラスターで失敗し、次のエラーが表示されます。

```
RLS-protected relation "rls_protected_table" cannot be accessed via datasharing
 query.
```

  パラメータ ROW LEVEL SECURITY OFF FOR DATASHARES を指定して ALTER TABLE コマンドを使用すると、データ共有の RLS をオフにできます。ALTER TABLE を使用して RLS を有効または無効にする方法の詳細については、「ALTER TABLE」を参照してください。
- クロスデータベースクエリでは、Amazon Redshift は RLS で保護されたリレーションへの読み取りをブロックします。IGNORE RLS 権限を持つユーザーは、クロスデータベースクエリを使用

して保護されたリレーションにアクセスできます。IGNORE RLS 権限のないユーザーが、クロスデータベースクエリを通じて RLS で保護されたリレーションにアクセスすると、次のエラーが表示されます。

```
RLS-protected relation "rls_protected_table" cannot be accessed via cross-database
 query.
```

- ALTER RLS POLICY では、USING (using_predicate_exp) 句を使用する、RLS ポリシーの変更のみがサポートされています。ALTER RLS POLICY を実行しているときに WITH 句を使用して RLS ポリシーを変更することはできません。
- 以下の設定オプションのいずれかの値がセッションのデフォルト値と一致しない場合、行レベルのセキュリティが有効になっているリレーションをクエリすることはできません。
  - `enable_case_sensitive_super_attribute`
  - `enable_case_sensitive_identifier`
  - `downcase_delimited_identifier`

  行レベルのセキュリティをオンにしてリレーションをクエリしようとして、「RLS で保護されたリレーションは、大文字と小文字の区別がデフォルト値と異なるため、セッションレベルの設定をサポートしていません」というメッセージが表示される場合は、セッションの設定オプションをリセットすることを検討してください。
- プロビジョニングされたクラスターまたはサーバーレス名前空間に行レベルのセキュリティポリシーが適用されている場合、以下のコマンドは標準ユーザーにはブロックされます。

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

  RLS ポリシーを作成するときは、ポリシー作成時のセッションの設定オプションの設定と一致するように、標準ユーザーのデフォルトの設定オプション設定を変更することをお勧めします。スーパーユーザーと ALTER USER 権限を持つユーザーは、パラメータグループ設定または ALTER USER コマンドを使用して、この操作を行うことができます。パラメータグループの詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift パラメータグループ」を参照してください。ALTER USER コマンドの詳細については、「ALTER USER」を参照してください。
- 行レベルのセキュリティポリシーが設定されたビューや遅延バインディングビューは、通常のユーザーが CREATE VIEW コマンドを使用して置き換えることはできません。ビューまたは LBV を RLS ポリシーに置き換えるには、まず、それらにアタッチされている RLS ポリシーをすべてデタッチし、ビューまたは LBV を置き換えてから、ポリシーを再アタッチします。スーパーユー

ザーと、`sys:secadmin permission` を持っているユーザーは、ポリシーをデタッチしなくても、RLS ポリシーが設定されたビューまたは LBV で CREATE VIEW を使用できます。

- 行レベルのセキュリティポリシーが設定されているビューは、システムテーブルやシステムビューを参照できません。
- 通常のビューが参照する遅延バインディングビューは RLS 保護ができません。
- RLS 保護がなされた関係と、データレイクからのネストされたデータには、同じクエリではアクセスできません。

RLS パフォーマンスのベストプラクティス

RLS で保護されているテーブルで Amazon Redshift のパフォーマンスを向上させるためのベストプラクティスは次のとおりです。

演算子と関数の安全性

RLS で保護されたテーブルを照合するとき、特定の演算子または関数を使用すると、パフォーマンスが低下する可能性があります。Amazon Redshift は、RLS で保護されたテーブルを照合するうえで、演算子と関数を安全または安全でないものとして分類します。関数または演算子は、入力内容に応じて観察可能な副作用がない場合、RLS に安全なものとして分類されます。特に、RLS に安全な関数または演算子は、次のいずれに該当することはできません。

- エラーメッセージの有無にかかわらず、入力値または入力値に依存する値を出力。
- 入力値に依存する失敗またはエラーを返答。

RLS に対して安全ではない演算子には、次のものが含まれます。

- 算術演算子 - +、-、/、*、%。
- テキスト演算子 - LIKE と SIMILAR TO。
- 演算子の投入。
- UDF。

次の SELECT ステートメントを使用して、演算子と関数の安全性を確認します。

```
SELECT proname, proc_is_rls_safe(oid) FROM pg_proc;
```

Amazon Redshift は、RLS で保護されたテーブルにクエリを計画する際に、RLS に安全ではない演算子と関数を含むユーザー述語の評価順序に制限を課します。RLS に安全ではない演算子または関数を参照するクエリは、RLS で保護されたテーブルを照合する際にパフォーマンス低下に影響する可能性があります。Amazon Redshift が RLS に安全ではない述語をベーステーブルスキャンにプッシュダウンしてソートキーを活用できない場合、パフォーマンスが大幅に低下する可能性があります。パフォーマンスを向上させるには、ソートキーを活用する RLS に安全ではない述語を使用するクエリは避けてください。Amazon Redshift が演算子と関数をプッシュダウンできることを確認するには、EXPLAIN ステートメントをシステムアクセス許可の EXPLAIN RLS と組み合わせて使用できます。

結果のキャッシュ

クエリの実行時間を短縮してシステムパフォーマンスを向上させるため、Amazon Redshift では、リーダーノード上のメモリで特定タイプのクエリ結果がキャッシュされます。

保護されていないテーブルのすべての条件が満たされて、かつ次の条件がすべて満たされた場合、Amazon Redshift は RLS で保護されたテーブルをスキャンする新しいクエリに対してキャッシュ結果を適用します。

- ポリシーのテーブルまたはビューは修正されていません。
- ポリシーは、実行するたびに評価する必要がある関数 (GETDATE または CURRENT_USER など) を使用しません。

パフォーマンスを向上させるには、上記の条件を満たさないポリシー述語の使用を避けてください。

Amazon Redshift での結果キャッシュの詳細については、「結果のキャッシュ」を参照してください。

複雑なポリシー

パフォーマンスを向上させるには、複数のテーブルを結合するサブクエリを持つ複雑なポリシーの使用を避けてください。

行レベルのセキュリティのエンドツーエンドの例

以下の内容は、スーパーユーザーが一部のユーザーとロールを作成する方法を示すエンドツーエンドの例です。次に、secadmin ロールを持つユーザーが RLS ポリシーを作成、アタッチ、デタッチ、削除します。この例では、チケットサンプルデータベースを使用します。詳細については、

「Amazon Redshift 入門ガイド」の「[Amazon S3 から Amazon Redshift にデータのロード]」を参照してください。

```
-- Create users and roles referenced in the policy statements.
CREATE ROLE analyst;
CREATE ROLE consumer;
CREATE ROLE dbadmin;
CREATE ROLE auditor;
CREATE USER bob WITH PASSWORD 'Name_is_bob_1';
CREATE USER alice WITH PASSWORD 'Name_is_alice_1';
CREATE USER joe WITH PASSWORD 'Name_is_joe_1';
CREATE USER molly WITH PASSWORD 'Name_is_molly_1';
CREATE USER bruce WITH PASSWORD 'Name_is_bruce_1';
GRANT ROLE sys:secadmin TO bob;
GRANT ROLE analyst TO alice;
GRANT ROLE consumer TO joe;
GRANT ROLE dbadmin TO molly;
GRANT ROLE auditor TO bruce;
GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_sales_redshift TO PUBLIC;
GRANT ALL ON TABLE tickit_event_redshift TO PUBLIC;

-- Create table and schema referenced in the policy statements.
CREATE SCHEMA target_schema;
GRANT ALL ON SCHEMA target_schema TO PUBLIC;
CREATE TABLE target_schema.target_event_table (LIKE tickit_event_redshift);
GRANT ALL ON TABLE target_schema.target_event_table TO PUBLIC;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check the tuples visible to analyst alice.
-- Should contain all 3 categories.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');
```

```
SELECT poldb, polname, polalias, polatts, polqual, polenabled, polmodifiedby FROM
 svv_rls_policy WHERE poldb = CURRENT_DATABASE();

ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;

SELECT * FROM svv_rls_attached_policy;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that tuples with only `Concert` category will be visible to analyst alice.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to consumer joe.
SET SESSION AUTHORIZATION joe;

-- Although the policy is attached to a different role, no tuples will be
-- visible to consumer joe because the default deny all policy is applied.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that tuples with only `Concert` category will be visible to dbadmin molly.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Check that EXPLAIN output contains RLS SecureScan to prevent disclosure of
-- sensitive information such as RLS filters.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;
```

```
-- Grant IGNORE RLS permission so that RLS policies do not get applicable to role
 dbadmin.
GRANT IGNORE RLS TO ROLE dbadmin;

-- Grant EXPLAIN RLS permission so that anyone in role auditor can view complete
 EXPLAIN output.
GRANT EXPLAIN RLS TO ROLE auditor;

-- Change session to dbadmin molly.
SET SESSION AUTHORIZATION molly;

-- Check that all tuples are visible to dbadmin molly because `IGNORE RLS` is granted
 to role dbadmin.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to auditor bruce.
SET SESSION AUTHORIZATION bruce;

-- Check explain plan is visible to auditor bruce because `EXPLAIN RLS` is granted to
 role auditor.
EXPLAIN SELECT catgroup, count(*) FROM tickit_category_redshift GROUP BY catgroup ORDER
 BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;

-- Change session to analyst alice.
SET SESSION AUTHORIZATION alice;

-- Check that no tuples are visible to analyst alice.
-- Although the policy is detached, no tuples will be visible to analyst alice
-- because of default deny all policy is applied if the table has RLS on.
SELECT catgroup, count(*)
FROM tickit_category_redshift
GROUP BY catgroup ORDER BY catgroup;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;
```

```
CREATE RLS POLICY policy_events
WITH (eventid INTEGER) AS ev
USING (
    ev.eventid IN (SELECT eventid FROM tickit_sales_redshift WHERE qtysold <3)
);

ATTACH RLS POLICY policy_events ON tickit_event_redshift TO ROLE analyst;
ATTACH RLS POLICY policy_events ON target_schema.target_event_table TO ROLE consumer;

RESET SESSION AUTHORIZATION;

-- Can not cannot alter type of dependent column.
ALTER TABLE target_schema.target_event_table ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_event_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN eventid TYPE float;
ALTER TABLE tickit_sales_redshift ALTER COLUMN qtysold TYPE float;

-- Can not cannot rename dependent column.
ALTER TABLE target_schema.target_event_table RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_event_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN eventid TO renamed_eventid;
ALTER TABLE tickit_sales_redshift RENAME COLUMN qtysold TO renamed_qtysold;

-- Can not drop dependent column.
ALTER TABLE target_schema.target_event_table DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_event_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN eventid CASCADE;
ALTER TABLE tickit_sales_redshift DROP COLUMN qtysold CASCADE;

-- Can not drop lookup table.
DROP TABLE tickit_sales_redshift CASCADE;

-- Change session to security administrator bob.
SET SESSION AUTHORIZATION bob;

DROP RLS POLICY policy_concerts;
DROP RLS POLICY IF EXISTS policy_events;

ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;

RESET SESSION AUTHORIZATION;

-- Drop users and roles.
DROP USER bob;
```

```
DROP USER alice;
DROP USER joe;
DROP USER molly;
DROP USER bruce;
DROP ROLE analyst;
DROP ROLE consumer;
DROP ROLE auditor FORCE;
DROP ROLE dbadmin FORCE;
```

メタデータセキュリティ

Amazon Redshift の行レベルのセキュリティと同様に、メタデータセキュリティではメタデータをよりきめ細かく制御できます。プロビジョニングされたクラスターまたはサーバーレスワークグループでメタデータセキュリティが有効になっている場合、ユーザーは、表示アクセス権のあるオブジェクトのメタデータを表示できます。メタデータセキュリティにより、必要に応じて可視性を分離できます。例えば、1 つのデータウェアハウスを使用してすべてのデータストレージを一元化できます。ただし、複数のセクターのデータを保存すると、セキュリティ管理が難しくなる可能性があります。メタデータセキュリティを有効にすると、可視性を設定できます。あるセクターのユーザーにはオブジェクトの可視性を高め、別のセクターのユーザーには表示アクセスを制限することができます。メタデータセキュリティは、スキーマ、テーブル、ビュー、マテリアライズドビュー、ストアドプロシージャ、ユーザー定義機能、および機械学習モデルなどのすべてのオブジェクトタイプをサポートしています。

ユーザーは、次の状況でオブジェクトのメタデータを表示できます。

- オブジェクトへのアクセスがユーザーに許可されている場合。
- ユーザーが所属するグループまたはロールにオブジェクトアクセスが付与されている場合。
- オブジェクトが公開されている場合。
- ユーザーがデータベースオブジェクトの所有者である場合。

メタデータセキュリティを有効にするには、ALTER SYSTEM コマンドを使用します。ALTER SYSTEM コマンドをメタデータセキュリティで使用する方法の構文は次のとおりです。

```
ALTER SYSTEM SET metadata_security=[true|t|on|false|f|off];
```

メタデータセキュリティを有効にすると、必要なアクセス許可を持つすべてのユーザーが、アクセス権を持つオブジェクトの関連メタデータを表示できます。特定のユーザーだけがメタデータセキュリ

ティを表示できるようにするには、ACCESS CATALOG アクセス許可をロールに付与し、そのロールをそのユーザーに割り当てます。ロールを使用してセキュリティコントロールを改善する方法の詳細については、「ロールベースのアクセスコントロール」を参照してください。

次の例は、ロールに ACCESS CATALOG アクセス許可を付与し、そのロールをユーザーに割り当てる方法を示しています。アクセス許可の付与の詳細については、GRANT コマンドを参照してください。

```
CREATE ROLE sample_metadata_viewer;

GRANT ACCESS CATALOG TO ROLE sample_metadata_viewer;

GRANT ROLE sample_metadata_viewer to salesadmin;
```

すでに定義されているロールを使用する場合は、システム定義ロール operator、secadmin、dba、および superuser のすべてに、オブジェクトメタデータを表示するために必要なアクセス許可が付与されます。デフォルトでは、スーパーユーザーはカタログ全体を表示できます。

```
GRANT ROLE operator to sample_user;
```

ロールを使用してメタデータセキュリティをコントロールしている場合、ロールベースのアクセスコントロールに付属するすべてのシステムビューと機能にアクセスできます。例えば、SVV_ROLES ビューをクエリして、すべてのロールを表示できます。ユーザーがロールまたはグループのメンバーかどうかを確認するには、USER_IS_MEMBER_OF 関数を使用します。SVV ビューの完全なリストについては、「SVV メタデータビュー」を参照してください。システム情報関数の一覧については、「システム情報関数」を参照してください。

動的データマスキング

Amazon Redshift の動的データマスキング (DDM) を使用すると、データウエアハウス内の機密データを保護できます。Amazon Redshift が機密データをクエリ時にユーザーに表示する方法を、データベースで変換せずに操作できます。特定のユーザーまたはロールにカスタムの難読化ルールを適用するマスキングポリシーを通じて、データへのアクセスを制御します。これにより、基になるデータを変更したり、SQL クエリを編集したりすることなく、プライバシー要件の変更に対応できます。

動的データマスキングポリシーは、特定の形式に一致するデータを隠したり、難読化したり、仮名化したりします。テーブルにアタッチされると、その 1 つ以上の列にマスキング式が適用されます。さらにマスキングポリシーを変更して、特定のユーザーのみに適用したり、ロールベースのアクセス

コントロール (RBAC) で作成できるユーザー定義のロールにのみ適用したりできます。さらに、マスキングポリシーを作成するときに条件列を使用してセルレベルで DDM を適用できます。条件付きマスキングの詳細については、「条件付き動的データマスキング」を参照してください。

難読化レベルの異なる複数のマスキングポリシーをテーブル内の同じ列に適用し、それらを異なるロールに割り当てることができます。1 つの列に異なるポリシーが異なるロールに適用されている場合に競合が発生しないように、アプリケーションごとに優先順位を設定できます。これにより、特定のユーザーまたはロールがどのデータにアクセスできるかを制御できます。DDM ポリシーでは、SQL や Python で記述されたユーザー定義関数、または AWS Lambda を使用して、データを部分的または完全に編集したり、データをハッシュしたりできます。ハッシュを使用してデータをマスキングすることで、機密情報にアクセスせずに、このデータに結合を適用できます。

動的データマスキングポリシーを管理するための SQL コマンド

次のアクションを実行して、動的データマスキングポリシーの作成、アタッチ、デタッチ、削除を行うことができます。

- DDM ポリシーを作成するには、CREATE MASKING POLICY コマンドを使用します。

  以下は、SHA-2 ハッシュ関数を使用してマスキングポリシーを作成する例です。

  ```
  CREATE MASKING POLICY hash_credit
  WITH (credit_card varchar(256))
  USING (sha2(credit_card + 'testSalt', 256));
  ```

- 既存の DDM ポリシーを変更するには、ALTER MASKING POLICY コマンドを使用します。

  次に示すのは、既存のマスキングポリシーを変更する例です。

  ```
  ALTER MASKING POLICY hash_credit
  USING (sha2(credit_card + 'otherTestSalt', 256));
  ```

- DDM ポリシーをテーブルで 1 つ以上のユーザーまたはロールにアタッチするには、ATTACH MASKING POLICY コマンドを実行します。

  次に示すのは、マスキングポリシーを列とロールのペアにアタッチする例です。

  ```
   ATTACH MASKING POLICY hash_credit
  ON credit_cards (credit_card)
  TO ROLE science_role
  PRIORITY 30;
  ```

PRIORITY 句は、同じ列に複数のポリシーがアタッチされている場合に、どのマスキングポリシーをユーザーセッションに適用するかを決定します。例えば、前の例のユーザーが、同じクレジットカード列に優先度が 20 の別のマスキングポリシーをアタッチしている場合、science_role のポリシーが適用されます。優先度が 30 と高いためです。

- DDM ポリシーをテーブルで 1 つ以上のユーザーまたはロールからデタッチするには、DETACH MASKING POLICY コマンドを実行します。

  以下は、列/ロールペアからマスキングポリシーをデタッチする例です。

```
DETACH MASKING POLICY hash_credit
ON credit_cards(credit_card)
FROM ROLE science_role;
```

- DDM ポリシーをすべてのデータベースから削除するには、DROP MASKING POLICY コマンドを使用します。

  以下は、すべてのデータベースからマスキングポリシーを削除する例です。

```
DROP MASKING POLICY hash_credit;
```

動的データマスキングポリシー階層

複数のマスキングポリシーをアタッチする場合は、以下を考慮します。

- 複数のマスキングポリシーを 1 つの列にアタッチできます。
- 複数のマスキングポリシーがクエリに適用される場合、それぞれの列にアタッチされている最も優先度の高いポリシーが適用されます。次の例を考えます。

```
ATTACH MASKING POLICY partial_hash
ON credit_cards(address, credit_card)
TO ROLE analytics_role
PRIORITY 20;

ATTACH MASKING POLICY full_hash
ON credit_cards(credit_card, ssn)
TO ROLE auditor_role
PRIORITY 30;

SELECT address, credit_card, ssn
```

```
FROM credit_cards;
```

SELECT ステートメントを実行すると、分析と監査の両方のロールを持つユーザーには、partial_hash マスキングポリシーが適用されたアドレス列が表示されます。クレジットカード列では full_hash ポリシーの優先度が高いため、full_hash マスキングポリシーが適用されたクレジットカード列と SSN 列が表示されます。

- マスキングポリシーをアタッチするときに優先度を指定しない場合、デフォルトの優先度は 0 です。
- 同じ列に 2 つのポリシーを同じ優先度でアタッチすることはできません。
- ユーザーと列、またはロールと列の同じ組み合わせに 2 つのポリシーをアタッチすることはできません。
- 同じユーザーまたはロールにアタッチされているときに、同じ SUPER パスに複数のマスキングポリシーが適用される場合、最も優先順位の高いアタッチメントのみが有効になります。次に挙げるサンプルを参考にしてください。

  最初の例は、同じパスに 2 つのマスキングポリシーがアタッチされ、優先順位の高いポリシーが有効になっていることを示しています。

```
ATTACH MASKING POLICY hide_name
ON employees(col_person.name)
TO PUBLIC
PRIORITY 20;

ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 30;

--Only the hide_last_name policy takes effect.
SELECT employees.col_person.name FROM employees;
```

  2 つ目の例は、同じ SUPER オブジェクト内の異なるパスに 2 つのマスキングポリシーがアタッチされているため、ポリシー間で競合が発生していないことを示しています。両方のアタッチメントが同時に適用されます。

```
ATTACH MASKING POLICY hide_first_name
ON employees(col_person.name.first)
TO PUBLIC
PRIORITY 20;
```

```
ATTACH MASKING POLICY hide_last_name
ON employees(col_person.name.last)
TO PUBLIC
PRIORITY 20;

--Both col_person.name.first and col_person.name.last are masked.
SELECT employees.col_person.name FROM employees;
```

特定のユーザーと列またはロールと列の組み合わせにどのマスキングポリシーが適用されるかを確認するには、`sys:secadmin` ロールを持つユーザーが、SVV_ATTACHED_MASKING_POLICY システムビューで列/ロールまたは列/ユーザーのペアを検索できます。詳細については、「動的データマスキングのシステムビュー」を参照してください。

SUPER データタイプパスでの動的データマスキングの使用

Amazon Redshift は、SUPER タイプ列のパスへの動的データマスキングポリシーのアタッチをサポートしています。SUPER データ型の詳細については、Amazon Redshift の半構造化データを参照してください。

SUPER タイプ列のパスにマスキングポリシーをアタッチするときは、次の点を考慮します。

- マスキングポリシーを列のパスにアタッチする場合、その列を SUPER データタイプとして定義する必要があります。マスキングポリシーは SUPER パスのスカラー値にのみ適用できます。複雑な構造や配列にはマスキングポリシーを適用できません。
- SUPER パスが競合しない限り、1 つの SUPER 列の複数のスカラー値に異なるマスキングポリシーを適用できます。例えば、スーパーパス `a.b` と `a.b.c` が競合するのは、それらが同じパス上にあるときに、`a.b` が `a.b.c` の親になっているためです。スーパーパス `a.b.c` と `a.b.d` は競合しません。
- Amazon Redshift で、マスキングポリシーがアタッチするパスがデータに存在して、想定されるタイプであることを確認するには、ユーザーのクエリランタイム時にポリシーが適用される必要があります。例えば、INT 値を含む SUPER パスに TEXT 値をマスキングするマスキングポリシーをアタッチすると、Amazon Redshift はパスで値のタイプをキャストしようとします。

  このような状況では、ランタイム時の Amazon Redshift の動作は、SUPER オブジェクトをクエリするための設定によって異なってきます。デフォルトでは、Amazon Redshift は lax モードになっており、指定した SUPER パスに関して欠落したパスや無効なキャストを NULL として解決します。SUPER 関連の設定の詳細については、「SUPER 設定」を参照してください。

- SUPER はスキーマレスタイプです。つまり、Amazon Redshift は特定の SUPER パスに値が存在することを確認できません。存在しない SUPER パスにマスキングポリシーをアタッチし、Amazon Redshift が lax モードの場合、Amazon Redshift はパスを NULL 値に解決します。SUPER 列のパスにマスキングポリシーをアタッチするときは、SUPER オブジェクトの予想される形式と、それらに予期しない属性が含まれる可能性を考慮することをお勧めします。SUPER 列に予期しないスキーマがあると思われる場合は、マスキングポリシーを SUPER 列に直接アタッチすることを検討します。SUPER タイプ情報関数を使用して属性とタイプをチェックしたり、OBJECT_TRANSFORM を使用して値をマスクしたりできます。SUPER タイプ情報関数の詳細については、「SUPER 型の情報関数」を参照してください。

例

SUPER パスにマスキングポリシーをアタッチする

次の例では、1 列の複数の SUPER タイプパスに複数のマスキングポリシーをアタッチします。

```
CREATE TABLE employees (
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc."
            }
        ')
    ),
    (
        json_parse('
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
```

```
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org."
            }
        ')
    )
;
GRANT ALL ON ALL TABLES IN SCHEMA "public" TO PUBLIC;

-- Create the masking policies.

-- This policy converts the given name to all uppercase letters.
CREATE MASKING POLICY mask_first_name
WITH(first_name TEXT)
USING ( UPPER(first_name) );

-- This policy replaces the given name with the fixed string 'XXXX'.
CREATE MASKING POLICY mask_last_name
WITH(last_name TEXT)
USING ( 'XXXX'::TEXT );

-- This policy rounds down the given age to the nearest 10.
CREATE MASKING POLICY mask_age
WITH(age INT)
USING ( (FLOOR(age::FLOAT / 10) * 10)::INT );

-- This policy converts the first five digits of the given SSN to 'XXX-XX'.
CREATE MASKING POLICY mask_ssn
WITH(ssn TEXT)
USING ( 'XXX-XX-'::TEXT || SUBSTRING(ssn::TEXT FROM 8 FOR 4) );

-- Attach the masking policies to the employees table.
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.name.first)
TO PUBLIC;

ATTACH MASKING POLICY mask_last_name
ON employees(col_person.name.last)
TO PUBLIC;

ATTACH MASKING POLICY mask_age
ON employees(col_person.age)
TO PUBLIC;
```

```
ATTACH MASKING POLICY mask_ssn
ON employees(col_person.ssn)
TO PUBLIC;

-- Verify that your masking policies are attached.
SELECT
    policy_name,
    TABLE_NAME,
    priority,
    input_columns,
    output_columns
FROM
    svv_attached_masking_policy;

   policy_name   | table_name | priority |           input_columns           |
 output_columns
-----------------+------------+----------+-----------------------------------
+-----------------------------------
 mask_age        | employees  |        0 | ["col_person.\"age\""]            |
 ["col_person.\"age\""]
 mask_first_name | employees  |        0 | ["col_person.\"name\".\"first\""] |
 ["col_person.\"name\".\"first\""]
 mask_last_name  | employees  |        0 | ["col_person.\"name\".\"last\""]  |
 ["col_person.\"name\".\"last\""]
 mask_ssn        | employees  |        0 | ["col_person.\"ssn\""]            |
 ["col_person.\"ssn\""]
(4 rows)

-- Observe the masking policies taking effect.
SELECT col_person FROM employees ORDER BY col_person.age;

-- This result is formatted for ease of reading.
         col_person
--------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc."
}
```

```
{
    "name": {
        "first": "JANE",
        "last": "XXXX"
    },
    "age": 30,
    "ssn": "XXX-XX-7777",
    "company": "Organization Org."
}
```

SUPER パスに無効なマスキングポリシーをアタッチする例を以下に示します。

```
-- This attachment fails because there is already a policy
-- with equal priority attached to employees.name.last, which is
-- on the same SUPER path as employees.name.
ATTACH MASKING POLICY mask_ssn
ON employees(col_person.name)
TO PUBLIC;
ERROR:  DDM policy "mask_last_name" is already attached on relation "employees" column
 "col_person."name"."last"" with same priority

-- Create a masking policy that masks DATETIME objects.
CREATE MASKING POLICY mask_date
WITH(INPUT DATETIME)
USING ( INPUT );

-- This attachment fails because SUPER type columns can't contain DATETIME objects.
ATTACH MASKING POLICY mask_date
ON employees(col_person.company)
TO PUBLIC;
ERROR:  cannot attach masking policy for output of type "timestamp without time zone"
 to column "col_person."company"" of type "super
```

次に示すのは、存在しないスーパーパスにマスキングポリシーをアタッチする例です。デフォルトでは、Amazon Redshift は NULL へのパスを解決します。

```
ATTACH MASKING POLICY mask_first_name
ON employees(col_person.not_exists)
TO PUBLIC;

SELECT col_person FROM employees LIMIT 1;

-- This result is formatted for ease of reading.
```

```
         col_person
-----------------------------------
{
    "name": {
        "first": "JOHN",
        "last": "XXXX"
    },
    "age": 20,
    "ssn": "XXX-XX-3333",
    "company": "Company Inc.",
    "not_exists": null
}
```

条件付き動的データマスキング

マスキング式に条件式を含むマスキングポリシーを作成することで、セルレベルでデータをマスクできます。例えば、その行の別の列の値に応じて、値に異なるマスクを適用するマスキングポリシーを作成できます。

以下は、条件付きデータマスキングを使用して、詐欺に巻き込まれたクレジットカード番号を部分的に編集し、他のすべてのクレジットカード番号を完全に隠すマスキングポリシーを作成してアタッチする例です。この例を実行するには、スーパーユーザーであるか、`sys:secadmin` ロールを持っている必要があります。

```
--Create an analyst role.
CREATE ROLE analyst;

--Create a credit card table. The table contains an is_fraud boolean column,
--which is TRUE if the credit card number in that row was involved in a fraudulent
 transaction.
CREATE TABLE credit_cards (id INT, is_fraud BOOLEAN, credit_card_number VARCHAR(16));

--Create a function that partially redacts credit card numbers.
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card VARCHAR(16))
RETURNS VARCHAR(16) IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
```

```
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--Create a masking policy that partially redacts credit card numbers if the is_fraud
 value for that row is TRUE,
--and otherwise blanks out the credit card number completely.
CREATE MASKING POLICY card_number_conditional_mask
    WITH (fraudulent BOOLEAN, pan varchar(16))
    USING (CASE WHEN fraudulent THEN REDACT_CREDIT_CARD(pan)
                ELSE Null
           END);

--Attach the masking policy to the credit_cards/analyst table/role pair.
ATTACH MASKING POLICY card_number_conditional_mask ON credit_cards (credit_card_number)
 USING (is_fraud, credit_card_number)
 TO ROLE analyst PRIORITY 100;
```

動的データマスキングのシステムビュー

スーパーユーザー、sys:operator ロールを持つユーザー、ACCESS SYSTEM TABLE アクセス許可を持つユーザーは、次の DDM 関連のシステムビューにアクセスできます。

- SVV_MASKING_POLICY

  SVV_MASKING_POLICY では、クラスターまたはワークグループで作成されたすべてのマスキングポリシーを表示できます。
- SVV_ATTACHED_MASKING_POLICY

  SVV_ATTACHED_MASKING_POLICY では、現在接続されているデータベースにポリシーがアタッチされているすべてのリレーションとユーザーまたはロールを表示できます。
- SYS_APPLIED_MASKING_POLICY_LOG

  SYS_APPLIED_MASKING_POLICY_LOG では、DDM で保護されたリレーションを参照するクエリに対するマスキングポリシーの適用をトレースできます。

以下に、システムビューを使用して確認できる情報の例を示します。

```
--Select all policies associated with specific users, as opposed to roles
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee_type = 'user';

--Select all policies attached to a specific user
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE grantee = 'target_grantee_name'

--Select all policies attached to a given table
SELECT policy_name,
       schema_name,
       table_name,
       grantee
FROM svv_attached_masking_policy
WHERE table_name = 'target_table_name'
      AND schema_name = 'target_schema_name';

--Select the highest priority policy attachment for a given role
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_role_on_column(
        'target_schema_name',
        'target_table_name',
        'target_column_name',
        'target_role_name')
ORDER BY samp.priority desc
LIMIT 1;
```

```
--See which policy a specific user will see on a specific column in a given relation
SELECT samp.policy_name,
       samp.priority,
       samp.grantee,
       smp.policy_expression
FROM svv_masking_policy AS smp
JOIN svv_attached_masking_policy AS samp
    ON samp.policy_name = smp.policy_name
WHERE
    samp.grantee_type = 'role' AND
    samp.policy_name = mask_get_policy_for_user_on_column(
        'target_schema_name',
        'target_table_name',
        'target_column_name',
        'target_user_name')
ORDER BY samp.priority desc;

 --Select all policies attached to a given relation.
SELECT policy_name,
schema_name,
relation_name,
database_name
FROM sys_applied_masking_policy_log
WHERE relation_name = 'relation_name'
AND schema_name = 'schema_name';
```

動的なデータマスキングを使用する際の考慮事項

動的なデータマスキングを使用する場合は、以下を考慮してください。

- ビューなど、テーブルから作成されたオブジェクトをクエリすると、オブジェクトを作成したユーザーのポリシーではなく、クエリを実行するユーザーのマスキングポリシーに基づいて結果が表示されます。例えば、アナリストロールを持つユーザーが secadmin によって作成されたビューをクエリすると、アナリストロールにアタッチされたマスキングポリシーを使用して結果が表示されます。
- EXPLAIN コマンドによって機密性の高いマスキングポリシーフィルターが公開されるのを防ぐために、SYS_EXPLAIN _DDM のアクセス許可を持つユーザーのみが EXPLAIN 出力に適用されたマスキングポリシーを確認できます。デフォルトでは、ユーザーには SYS_EXPLAIN_DDM のアクセス許可がありません。

  ロールにアクセス許可を付与するための構文は、次のとおりです。

```
GRANT EXPLAIN MASKING TO ROLE rolename
```

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

- ロールが異なるユーザーには、使用したフィルター条件または結合条件に基づいて異なる結果が表示されます。例えば、特定の列値を使用するテーブルで SELECT コマンドを実行する際、コマンドを実行するユーザーにその列を難読化するマスキングポリシーが適用されている場合、失敗します。
- DDM ポリシーは、前提となる操作や述語よりも先に適用する必要があります。マスキングポリシーには以下が含まれる場合があります。
  - 値を NULL に変換するなどの低コストの定数演算
  - HMAC ハッシュなどの中コストのオペレーション
  - 外部の Lambda ユーザー定義関数への呼び出しなどの高コストの操作

  そのため、可能な限り、単純なマスキング式を使用することをお勧めします。
- 行レベルのセキュリティポリシーを持つロールには DDM ポリシーを使用できますが、RLS ポリシーは DDM の前に適用されることに注意してください。動的データマスキング式は、RLS で保護された行を読み取ることができません。RLS の詳細については、「行レベルのセキュリティ」を参照してください。
- COPY コマンドを使用してパーケットから保護されているターゲットテーブルにコピーするときには、COPY ステートメントで列を明示的に指定する必要があります。COPY による列のマッピングの詳細については、「列のマッピングオプション」を参照してください。
- DDM ポリシーは、次の関係にアタッチできません。
  - システムテーブルとカタログ
  - 外部テーブル
  - データ共有テーブル
  - マテリアライズドビュー
  - データベース間の関係
  - 一時テーブル
  - 相関クエリ
- DDM ポリシーにはルックアップテーブルを含めることができます。USING 句にはルックアップテーブルを含めることができます。以下の関係タイプはルックアップテーブルとして使用できません。

- システムテーブルとカタログ
- 外部テーブル
- データ共有テーブル
- ビュー、マテリアライズドビュー、遅延バインディングビュー
- データベース間の関係
- 一時テーブル
- 相関クエリ

次に示すのは、マスキングポリシーをルックアップテーブルにアタッチする例です。

```
--Create a masking policy referencing a lookup table
CREATE MASKING POLICY lookup_mask_credit_card WITH (credit_card TEXT) USING (
  CASE
    WHEN
      credit_card IN (SELECT credit_card_lookup FROM credit_cards_lookup)
    THEN '000000XXXX0000'
    ELSE REDACT_CREDIT_CARD(credit_card)
    END
  );

--Provides access to the lookup table via a policy attached to a role
GRANT SELECT ON TABLE credit_cards_lookup TO MASKING POLICY lookup_mask_credit_card;
```

- ターゲット列のタイプとサイズと互換性のない出力を生成するようなマスキングポリシーをアタッチすることはできません。例えば、12 文字の文字列を VARCHAR(10) 列に出力するマスキングポリシーをアタッチすることはできません。Amazon Redshift では、次の例外がサポートされます。
  - 入力タイプ INTN のマスキングポリシーは、M < N であればサイズ INTM のポリシーにアタッチできます。例えば、BIGINT (INT8) 入力ポリシーを smallint (INT4) 列にアタッチできます。
  - 入力タイプが NUMERIC または DECIMAL のマスキングポリシーは、常に FLOAT 列にアタッチできます。
- DDM ポリシーは、データ共有では使用できません。データ共有のデータプロデューサーがデータ共有内のテーブルに DDM ポリシーをアタッチすると、テーブルをクエリしようとしているデータコンシューマーのユーザーはテーブルにアクセスできなくなります。DDM ポリシーがアタッチされたテーブルは、データ共有に追加できません。
- 以下の設定オプションのいずれかの値がセッションのデフォルト値と一致しない場合、DDM ポリシーがアタッチされているリレーションをクエリすることはできません。
  - `enable_case_sensitive_super_attribute`

- `enable_case_sensitive_identifier`
- `downcase_delimited_identifier`

DDM ポリシーがアタッチされているリレーションをクエリしようとして、「DDM で保護されたリレーションは、大文字と小文字の区別がデフォルト値と異なるため、セッションレベルの設定をサポートしていません」というメッセージが表示される場合は、セッションの設定オプションをリセットすることを検討してください。

- プロビジョニングされたクラスターまたはサーバーレス名前空間にデータマスキングポリシーが適用されている場合、以下のコマンドは標準ユーザーにはブロックされます。

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

DDM ポリシーを作成するときは、ポリシー作成時のセッションの設定オプション設定と一致するように、標準ユーザーのデフォルト設定オプション設定を変更することをお勧めします。スーパーユーザーと ALTER USER 権限を持つユーザーは、パラメータグループ設定または ALTER USER コマンドを使用して、この操作を行うことができます。パラメータグループの詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift パラメータグループ」を参照してください。ALTER USER コマンドの詳細については、「ALTER USER」を参照してください。

- DDM ポリシーがアタッチされたビューや遅延バインディングビューは、通常のユーザーが CREATE VIEW コマンドを使用して置き換えることはできません。ビューまたは LBV を DDM ポリシーに置き換えるには、まず、それらにアタッチされている DDM ポリシーをすべてデタッチし、ビューまたは LBV を置き換えてから、ポリシーを再アタッチします。スーパーユーザーと `sys:secadmin` アクセス許可を持っているユーザーは、ポリシーをデタッチすることなく、DDM ポリシーが設定されたビューまたは LBV で CREATE VIEW を使用できます。
- DDM ポリシーがアタッチされたビューは、システムテーブルとビューを参照できません。遅延バインディングビューは、システムテーブルとビューを参照できます。
- DDM ポリシーがアタッチされた遅延バインディングビューは、JSON ドキュメントなどのデータレイク内のネストされたデータを参照できません。
- 遅延バインディングビューがいずれかのビューから参照されている場合、その遅延バインディングビューには DDM ポリシーをアタッチできません。
- 遅延バインディングビューにアタッチされた DDM ポリシーは、列名でアタッチされます。クエリ時に、Amazon Redshift は、遅延バインディングビューにアタッチされたすべてのマスキングポリシーが正常に適用されていること、および遅延バインディングビューの出力列タイプがアタッチ

されたマスキングポリシーのタイプと一致することを検証します。検証が失敗した場合、Amazon Redshift はクエリのエラーを返します。

- DDM ポリシーの作成する際に、カスタマイズされたセッションコンテキスト変数を使用できます。次の例では、セッションコンテキスト変数を設定します。

```
-- Set a customized context variable.
SELECT set_config('app.city', 'XXXX', FALSE);

-- Create a MASKING policy using current_setting() to get the value of a customized
 context variable.
CREATE MASKING POLICY city_mask
WITH (city VARCHAR(30))
USING (current_setting('app.city')::VARCHAR(30));

-- Attach the policy on the target table to one or more roles.
ATTACH MASKING POLICY city_mask
ON tickit_users_redshift(city)
TO ROLE analyst, ROLE dbadmin;
```

カスタマイズされたセッションコンテキスト変数の設定と取得の詳細については、「SET」、「SET_CONFIG」、「SHOW」、「CURRENT_SETTING」、「RESET」を参照してください。サーバー設定変更全般の詳細については、「サーバー設定の変更」を参照してください。

> **Important**
>
> DDM ポリシー内でセッションコンテキスト変数を使用する場合、セキュリティポリシーは、ポリシーを呼び出すユーザーまたはロールに依存します。DDM ポリシーでセッションコンテキスト変数を使用する場合は、セキュリティの脆弱性を避けるように注意してください。

動的データマスキングのエンドツーエンドの例

次の内容は、マスキングポリシーを作成して列にアタッチする方法を示すエンドツーエンドの例です。これらのポリシーにより、ユーザーは各自のロールに割り当てられているポリシーの難読化の度合いに応じて、列にアクセスしてさまざまな値を確認できます。この例を実行するには、スーパーユーザーであるか、`sys:secadmin` ロールを持っている必要があります。

マスキングポリシーの作成

まず、テーブルを作成し、クレジットカードの値を入力します。

```
--create the table
CREATE TABLE credit_cards (
  customer_id INT,
  credit_card TEXT
);

--populate the table with sample values
INSERT INTO credit_cards
VALUES
  (100, '4532993817514842'),
  (100, '4716002041425888'),
  (102, '5243112427642649'),
  (102, '6011720771834675'),
  (102, '6011378662059710'),
  (103, '373611968625635')
;

--run GRANT to grant permission to use the SELECT statement on the table
GRANT SELECT ON credit_cards TO PUBLIC;

--create two users
CREATE USER regular_user WITH PASSWORD '1234Test!';

CREATE USER analytics_user WITH PASSWORD '1234Test!';

--create the analytics_role role and grant it to analytics_user
--regular_user does not have a role
CREATE ROLE analytics_role;

GRANT ROLE analytics_role TO analytics_user;
```

次に、分析ロールに適用するマスキングポリシーを作成します。

```
--create a masking policy that fully masks the credit card number
CREATE MASKING POLICY mask_credit_card_full
WITH (credit_card VARCHAR(256))
USING ('000000XXXX0000'::TEXT);

--create a user-defined function that partially obfuscates credit card data
```

```
CREATE FUNCTION REDACT_CREDIT_CARD (credit_card TEXT)
RETURNS TEXT IMMUTABLE
AS $$
    import re
    regexp = re.compile("^([0-9]{6})[0-9]{5,6}([0-9]{4})")

    match = regexp.search(credit_card)
    if match != None:
        first = match.group(1)
        last = match.group(2)
    else:
        first = "000000"
        last = "0000"

    return "{}XXXXX{}".format(first, last)
$$ LANGUAGE plpythonu;

--create a masking policy that applies the REDACT_CREDIT_CARD function
CREATE MASKING POLICY mask_credit_card_partial
WITH (credit_card VARCHAR(256))
USING (REDACT_CREDIT_CARD(credit_card));

--confirm the masking policies using the associated system views
SELECT * FROM svv_masking_policy;

SELECT * FROM svv_attached_masking_policy;
```

マスキングポリシーをアタッチする

マスキングポリシーをクレジットカードテーブルにアタッチします。

```
--attach mask_credit_card_full to the credit card table as the default policy
--all users will see this masking policy unless a higher priority masking policy is
 attached to them or their role
ATTACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
TO PUBLIC;

--attach mask_credit_card_partial to the analytics role
--users with the analytics role can see partial credit card information
ATTACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
TO ROLE analytics_role
```

```
PRIORITY 10;

--confirm the masking policies are applied to the table and role in the associated
 system view
SELECT * FROM svv_attached_masking_policy;

--confirm the full masking policy is in place for normal users by selecting from the
 credit card table as regular_user
SET SESSION AUTHORIZATION regular_user;

SELECT * FROM credit_cards;

--confirm the partial masking policy is in place for users with the analytics role by
 selecting from the credit card table as analytics_user
SET SESSION AUTHORIZATION analytics_user;

SELECT * FROM credit_cards;
```

マスキングポリシーの変更

次のセクションでは、動的データマスキングポリシーを変更する方法について説明します。

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--alter the mask_credit_card_full policy
ALTER MASKING POLICY mask_credit_card_full
USING ('00000000000000'::TEXT);

--confirm the full masking policy is in place after altering the policy, and that
 results are altered from '000000XXXX0000' to '00000000000000'
SELECT * FROM credit_cards;
```

マスキングポリシーのデタッチとドロップ

次のセクションでは、テーブルからすべての動的データマスキングポリシーを削除して、マスキングポリシーをデタッチおよびドロップする方法を示します。

```
--reset session authorization to the default
RESET SESSION AUTHORIZATION;

--detach both masking policies from the credit_cards table
```

```
DETACH MASKING POLICY mask_credit_card_full
ON credit_cards(credit_card)
FROM PUBLIC;

DETACH MASKING POLICY mask_credit_card_partial
ON credit_cards(credit_card)
FROM ROLE analytics_role;

--drop both masking policies
DROP MASKING POLICY mask_credit_card_full;

DROP MASKING POLICY mask_credit_card_partial;
```

スコープ設定アクセス許可

スコープ設定アクセス許可を使用すると、データベースまたはスキーマ内の特定タイプのすべてのオブジェクトに対するアクセス許可をユーザーまたはロールに付与できます。スコープ設定アクセス許可を持つユーザーやロールは、データベースまたはスキーマ内の現在および将来のすべてのオブジェクトに対して指定されたアクセス許可を持ちます。

データベースレベルのスコープ付きアクセス許可の範囲は、SVV_DATABASE_PRIVILEGES で確認できます。スキーマレベルのスコープ付きアクセス許可の範囲は、SVV_SCHEMA_PRIVILEGES で確認できます。

スコープ設定アクセス許可の適用について詳しくは、「GRANT」と「REVOKE」を参照してください。

スコープ設定アクセス許可の使用に関する考慮事項

スコープ設定アクセス許可を使用する場合は、次の事項を考慮します。

- スコープ設定アクセス許可を使用して、データベーススコープまたはスキーマスコープへのアクセス許可を、指定したユーザーまたはロールに対して付与したり、取り消したりすることができます。
- スコープ設定アクセス許可をユーザーグループに付与することはできません。
- スコープ設定アクセス許可の付与または取り消しによって、スコープ内の現在および今後のすべてのオブジェクトに対するアクセス許可が変更されます。
- スコープ設定アクセス許可とオブジェクトレベルのアクセス許可は互いに無関係に動作します。例えば、次の両方の場合、ユーザーはテーブルに対するアクセス許可を保持します。

  - テーブル schema1.table1 に対する SELECT と schema1 に対する SELECT スコープ設定アクセス許可をユーザーに付与します。次に、スキーマ schema1 内のすべてのテーブルに対する SELECT を取り消します。ユーザーは schema1.table1 に対する SELECT を保持します。
  - テーブル schema1.table1 に対する SELECT と schema1 に対する SELECT スコープ設定アクセス許可をユーザーに付与します。次に、schema1.table1 に対する SELECT を取り消します。ユーザーは schema1.table1 に対する SELECT を保持します。
- スコープ設定アクセス許可の付与または取り消すのためには、次の条件のうち 1 つを満たす必要があります。
  - スーパーユーザー。
  - そのアクセス許可の GRANT OPTION を持つユーザー。GRANT OPTION の詳細については、GRANT の WITH GRANT OPTION パラメータを参照してください。
- スコープ設定アクセス許可は、接続されているデータベースのオブジェクト、またはデータ共有からインポートされたデータベースに対してのみ付与または取り消しができます。
- スコープ設定アクセス許可を使用して、データ共有から作成した共有データベースに対するデフォルトのアクセス許可を設定できます。共有データベースに対するスコープ設定アクセス許可を付与されたコンシューマー側のデータ共有ユーザーは、プロデューサー側のデータ共有に新しく追加されたオブジェクトに対してそのアクセス許可を自動的に取得します。
- プロデューサーは、スキーマ内のオブジェクトに対するスコープ設定アクセス許可をデータ共有に付与できます。(プレビュー)

SQL リファレンス

Amazon Redshift を使用すると、SQL を活用して、データウェアハウスに保存されている大量のデータを効率的にクエリおよび分析できます。SQL リファレンスでは、SQL コマンド、データ型、関数、演算子などの構文と用途について説明しており、インサイトを抽出してデータドリブンな意思決定を行うことができます。このリファレンスを参照して、最適化されたクエリを記述し、データベースオブジェクトを作成および管理し、複雑なデータ変換を実行できます。以下のリファレンスでは、Amazon Redshift 環境内で SQL を使用してデータ分析要件を満たすための包括的な詳細を説明しています。

トピック

Amazon Redshift SQL

トピック

Amazon Redshift は、業界標準の SQL をベースに構築され、大規模データセットを管理する機能やハイパフォーマンス分析および分析結果のレポートをサポートする機能が追加されています。

> **Note**
>
> 単一 Amazon Redshift SQL ステートメントの最大サイズは 16 MB です。

リーダーノードでサポートされる SQL 関数

一部のAmazon Redshift クエリはコンピューティングノードに配布され実行されます。ほかのクエリはリーダーノードのみで実行されます。

ユーザーによって作成されたテーブルまたはシステムテーブル (STL または STV プレフィックスが付いたテーブル、SVL または SVV プレフィックスが付いたシステムビュー) をクエリが参照するたびに、リーダーノードは SQL をコンピューティングノードに配布します。リーダーノード上の PG プレフィックス付きカタログ テーブル (PG_TABLE_DEF など) のみを参照するクエリや、いずれのテーブルも参照しないクエリは、リーダーノード上で排他的に実行されます。

Amazon Redshift SQL 関数の中にはリーダーノードのみでサポートされ、コンピューティングノードではサポートされないものがあります。リーダーノード専用関数を使用するクエリは、コンピューティングノードではなくリーダーノードのみで実行される必要があり、そうでなければエラーが返されます。

リーダーノードで排他的に実行されるべき関数がユーザー定義テーブルまたは Amazon Redshift システムテーブルを参照した場合、ドキュメントにメモとして説明されているとおり、エラーが返されます。リーダーノードで排他的に実行される関数については、「リーダーノード専用関数」を参照してください。

例

次の例では、TICKIT のサンプルデータベースを使用します。サンプルデータベースの詳細については、「サンプルデータベース」を参照してください。

CURRENT_SCHEMA

CURRENT_SCHEMA 関数は、リーダーノード専用の関数です。この例では、クエリはテーブルを参照しないのでリーダーノードで排他的に実行されます。

```
select current_schema();

current_schema
---------------
public
```

次の例に示すクエリは、システムカタログテーブルを参照するので、リーダーノードで排他的に実行されます。

```
select * from pg_table_def
where schemaname = current_schema() limit 1;

 schemaname | tablename | column |   type   | encoding | distkey | sortkey | notnull
------------+-----------+--------+----------+----------+---------+---------+---------
 public     | category  | catid  | smallint | none     | t       |       1 | t
```

次の例に示すクエリは、コンピューティングノード上の Amazon Redshift システムテーブルを参照するため、エラーを返します。

```
select current_schema(), userid from users;

INFO:  Function "current_schema()" not supported.
ERROR:  Specified types or functions (one per INFO message) not supported on Amazon
 Redshift tables.
```

SUBSTR

SUBSTR もリーダーノード専用の関数です。次の例では、クエリはテーブルを参照しないのでリーダーノードで排他的に実行されます。

```
SELECT SUBSTR('amazon', 5);

+--------+
| substr |
+--------+
| on     |
+--------+
```

次の例に示すクエリは、コンピューティングノード上のテーブルを参照します。その結果、エラーが発生します。

```
SELECT SUBSTR(catdesc, 1) FROM category LIMIT 1;

ERROR: SUBSTR() function is not supported (Hint: use SUBSTRING instead)
```

前のクエリを正常に実行するには、SUBSTRING を使用してください。

```
SELECT SUBSTRING(catdesc, 1) FROM category LIMIT 1;

+-------------------------------+
|            substring          |
+-------------------------------+
| National Basketball Association |
+-------------------------------+
```

FACTORIAL()

FACTORIAL() はリーダーノード専用の関数です。次の例では、クエリはテーブルを参照しないのでリーダーノードで排他的に実行されます。

```
SELECT FACTORIAL(5);

 factorial
-------------
 120
```

次の例に示すクエリは、コンピューティングノード上のテーブルを参照します。これにより、クエリエディタ v2 を使用して実行すると、エラーが発生します。

```
create table t(a int);
insert into t values (5);
select factorial(a) from t;

ERROR: Specified types or functions (one per INFO message) not supported on Redshift
 tables.
Info: Function "factorial(bigint)" not supported.
```

Amazon Redshift および PostgreSQL

トピック

- Amazon Redshift、PostgreSQL JDBC および ODBC
- 実装方法が異なる機能
- サポートされていない PostgreSQL 機能
- サポートされていない PostgreSQL データ型
- サポートされていない PostgreSQL 関数

Amazon Redshift は PostgreSQL に基づいています。Amazon Redshift と PostgreSQL の間には非常に重要な相違点がいくつかあり、データウェアハウスアプリケーションを設計して開発するときはそれを考慮する必要があります。

Amazon Redshift は、具体的には、大規模データセットに対して複雑なクエリを行う必要があるオンライン分析処理 (OLAP) アプリケーションおよびビジネスインテリジェンス (BI) アプリケーション向けに設計されています。Amazon Redshift は多種多様な要件に対処するため、Amazon Redshift で使用する専用のデータストレージスキーマおよびクエリ実行エンジンは PostgreSQL の実装とは完全に異なります。例えば、オンライントランザクション処理 (OLTP) アプリケーションが一般的に

データを行に保存する場合、Amazon Redshift は、最適なメモリ使用量とディスク I/O のために特殊なデータ圧縮エンコードを使用してデータを列に保存します。セカンダリインデックスおよび効率的な単一行データオペレーションなど、小規模な OLTP 処理に適した一部の PostgreSQL 機能はパフォーマンスを向上させるために省略されています。

Amazon Redshift データウェアハウスシステムのアーキテクチャの詳細については、Amazon Redshift アーキテクチャ を参照してください。

PostgreSQL 9.x には、Amazon Redshift によってサポートされていない機能が一部含まれています。さらに、Amazon Redshift SQL と PostgreSQL との間には、認識しておく必要がある重要な違いがあります。このセクションでは、Amazon Redshift と PostgreSQL との違いに焦点を当てるとともに、SQL 実装を十分に活用したデータウェアハウスを開発するためのガイダンスを提供します。

Amazon Redshift、PostgreSQL JDBC および ODBC

Amazon Redshift は PostgreSQL に基づいているため、以前は JDBC4 Postgresql のドライバーバージョン 8.4.703 および psqlODBC バージョン 9.x ドライバーを使用することをお勧めしました。これらのドライバーを現在使用している場合は、新しい Amazon Redshift 特定のドライバーに移行することをお勧めします。ドライバーと接続の設定の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift 用の JDBC および ODBC ドライバー」を参照してください。

JDBC を使用して大きなデータセットを取得する際にユーザー側でメモリ不足エラーが発生することを避けるため、JDBC フェッチサイズパラメータを指定して、クライアントがデータをバッチ単位でフェッチするように設定できます。詳細については、「JDBC フェッチサイズパラメータの設定」を参照してください。

Amazon Redshift は JDBC maxRows パラメータを認識しません。その代わり、結果セットを制限するには LIMIT 句を指定します。また、OFFSET 句を使用して結果セット内の特定の開始点にスキップすることもできます。

実装方法が異なる機能

Amazon Redshift SQL の多くの言語要素は、対応する PostgreSQL 実装とはパフォーマンス特性が異なり、使用する構文およびセマンティクスもまったく異なるものとなっています。

> **Important**
>
> Amazon Redshift と PostgreSQL に含まれる共通要素のセマンティクスは同じであるとみなさないでください。判断しかねる差異については、Amazon Redshift デベロッパーガイドの SQL コマンド を参照して確認してください。

具体的な例として VACUUM コマンドが挙げられます。これはテーブルのクリーンアップおよび再編成に使用されます。VACUUM は PostgreSQL バージョンの場合とは機能が異なり、異なるパラメータセットを使用します。Amazon Redshift での VACUUM の使用についての詳細は、テーブルのバキューム処理 を参照してください。

しばしば、データベース管理機能およびツールも異なることがあります。例えば、Amazon Redshift では、システムの稼働状況に関する情報を、一連のシステムテーブルおよびビューで確認できる仕組みになっています。詳細については、「SYS モニタリングビュー」を参照してください。

Amazon Redshift 独自の実装を有する SQL 機能の例を以下に示します。

- CREATE TABLE

  Amazon Redshift では、テーブルスペース、テーブル分割、継承、および特定の制約をサポートしていません。Amazon Redshift の CREATE TABLE では、テーブルに対してソートおよびディストリビューションのアルゴリズムを定義することで、並列処理を最適化することができます。

  Amazon Redshift Spectrum では、CREATE EXTERNAL TABLE コマンドを使用したテーブルのパーティショニングをサポートしています。
- ALTER TABLE

  ALTER COLUMN アクションのサブセットのみがサポートされています。

  ADD COLUMN の場合、各 ALTER TABLE ステートメントで 1 つの列しか追加できません。
- COPY

  Amazon Redshift の COPY コマンドは、Amazon S3 バケットおよび Amazon DynamoDB テーブルからのデータのロードを可能にし、自動圧縮を容易にすることに特化されています。詳細については、「Amazon Redshift でのデータのロード」セクションおよび COPY コマンドリファレンスを参照してください。
- VACUUM

  VACUUM のパラメータは完全に異なります。例えば、PostgreSQL のデフォルトの VACUUM オペレーションは、単純に領域を再利用し、再び使用できるようにするだけです。一方で、Amazon Redshift のデフォルトの VACUUM オペレーションは VACUUM FULL です。これは、ディスク領域を再利用し、すべての行を再ソートします。
- VARCHAR 値の末尾のスペースは、文字列値の比較時に無視されます。詳細については、「末尾の空白の重要性」を参照してください。

サポートされていない PostgreSQL 機能

Amazon Redshift でサポートされていないこれらの PostgreSQL 機能を示します。

> **Important**
>
> Amazon Redshift と PostgreSQL に含まれる共通要素のセマンティクスは同じであるとみなさないでください。判断しかねる差異については、Amazon Redshift デベロッパーガイドの SQL コマンド を参照して確認してください。

- クエリツール psql はサポートされていません。Amazon Redshift RSQL クライアントはサポートされています。
- テーブル分割 (範囲およびリストの分割)
- テーブルスペース
- 制約
  - Unique
  - 外部キー
  - 主キー
  - 検査制約
  - 排他制約

  一意制約、主キー制約、および外部キー制約は許可されますが、情報提供専用です。これらの制約はシステムでは実施されませんが、クエリプランナーによって使用されます。
- データベースロール
- 継承
- PostgreSQL システム列

  Amazon Redshift SQL ではシステム列を暗黙的に定義しません。ただし、PostgreSQL システム列の名前 `oid`、`tableoid`、`xmin`、`cmin`、`xmax`、`cmax`、および `ctid` を、ユーザー定義の列の名前として使用することはできません。詳細については、https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html を参照してください。
- インデックス
- ウィンドウ関数の NULLS 句
- 照合

Amazon Redshift では、ロケール固有の照合順序またはユーザー定義の照合順序をサポートしていません。「照合順序」を参照してください。

- 値式
  - 添字付き式
  - 配列コンストラクタ
  - 行コンストラクタ
- トリガー
- 外部データの管理 (SQL/MED)
- テーブル関数
- 定数テーブルとして使用される VALUES リスト
- シーケンス
- フルテキスト検索
- RULE および TRIGGER アクセス許可。

  Amazon Redshift は、GRANT ALL または REVOKE ALL を実行するときにこれらのアクセス許可を付与または取り消しますが、RULE と TRIGGER アクセス許可の有無は、いかなる場合も被付与者のアクセス許可に影響しません。

サポートされていない PostgreSQL データ型

一般にクエリは、サポートされていないデータ型 (明示的または暗黙的なキャストなど) の使用を試みると、エラーを返します。ただし、サポートされていないデータ型を使用する一部のクエリはリーダーノードで実行されますが、コンピューティングノードでは実行されません。「リーダーノードでサポートされる SQL 関数」を参照してください。

サポートされているデータ型のリストについては、「データ型」を参照してください。

Amazon Redshift でサポートされていないこれらの PostgreSQL データタイプを示します。

- 配列
- BIT、BIT VARYING
- BYTEA
- コンポジット型
- 日付/時刻型

- 列挙型
- 幾何型
- HSTORE
- JSON
- ネットワークアドレス型
- 数値型
  - SERIAL、BIGSERIAL、SMALLSERIAL
  - MONEY
- オブジェクト識別子型
- 疑似型
- 範囲型
- 特殊な文字型
  - "char" – シングルバイトの内部型 (char というデータ型は引用符で囲まれている)。
  - name – オブジェクト名の内部型。

  このような型の詳細については、PostgreSQL ドキュメントの「[特殊な文字型](#)」を参照してください。
- テキスト検索型
- TXID_SNAPSHOT
- UUID
- XML

サポートされていない PostgreSQL 関数

サポートされている関数もその多くは、PostgreSQL とセマンティクスや用途が異なります。例えば、サポートされている関数の中にはリーダーノードでしか実行されないものがあります。また、サポートされていない関数の中には、リーダーノードで実行された場合、エラーを返さないものがあります。いくつかのケースでこれらの関数がエラーを返さないからといって、関数が Amazon Redshift によってサポートされていると受け取るべきではありません。

> **Important**
>
> Amazon Redshift と PostgreSQL に含まれる共通要素のセマンティクスは同じであるとみなさないでください。判断しかねる差異については、Amazon Redshift データベースデベロッパーガイドの SQL コマンド を参照して確認してください。

詳細については、「リーダーノードでサポートされる SQL 関数」を参照してください。

Amazon Redshift でサポートされていないこれらの PostgreSQL 機能を示します。

- アクセス特権照会関数
- アドバイザリロック関数
- 集計関数
  - STRING_AGG()
  - ARRAY_AGG()
  - EVERY()
  - XML_AGG()
  - CORR()
  - COVAR_POP()
  - COVAR_SAMP()
  - REGR_AVGX()、REGR_AVGY()
  - REGR_COUNT()
  - REGR_INTERCEPT()
  - REGR_R2()
  - REGR_SLOPE()
  - REGR_SXX()、REGR_SXY()、REGR_SYY()
- 配列関数と演算子
- バックアップ管理関数
- コメント情報関数
- データベースオブジェクト位置関数
- データベースオブジェクトサイズ関数
- 日付/時刻関数と演算子

  - CLOCK_TIMESTAMP()
  - JUSTIFY_DAYS()、JUSTIFY_HOURS()、JUSTIFY_INTERVAL()
  - PG_SLEEP()
  - TRANSACTION_TIMESTAMP()
- ENUM サポート関数
- 幾何関数と演算子
- 汎用ファイルアクセス関数
- IS DISTINCT FROM
- ネットワークアドレス関数と演算子
- 数学関数
  - DIV()
  - SETSEED()
  - WIDTH_BUCKET()
- セットを返す関数
  - GENERATE_SERIES()
  - GENERATE_SUBSCRIPTS()
- 範囲関数と演算子
- リカバリ制御関数
- リカバリ情報関数
- ROLLBACK_TO_SAVEPOINT 関数
- スキーマ可視性照会関数
- サーバーシグナリング関数
- スナップショット同期関数
- シーケンス操作関数
- 文字列関数
  - BIT_LENGTH()
  - OVERLAY()
  - CONVERT()、CONVERT_FROM()、CONVERT_TO()
  - ENCODE()
  - FORMAT()

  - QUOTE_NULLABLE()
  - REGEXP_MATCHES()
  - REGEXP_SPLIT_TO_ARRAY()
  - REGEXP_SPLIT_TO_TABLE()
- システムカタログ情報関数
- システム情報関数
  - CURRENT_CATALOG CURRENT_QUERY()
  - INET_CLIENT_ADDR()
  - INET_CLIENT_PORT()
  - INET_SERVER_ADDR() INET_SERVER_PORT()
  - PG_CONF_LOAD_TIME()
  - PG_IS_OTHER_TEMP_SCHEMA()
  - PG_LISTENING_CHANNELS()
  - PG_MY_TEMP_SCHEMA()
  - PG_POSTMASTER_START_TIME()
  - PG_TRIGGER_DEPTH()
  - SHOW VERSION()
- テキスト検索関数と演算子
- トランザクション ID およびスナップショット関数
- トリガー関数
- XML 関数

SQL の使用

トピック

- SQL リファレンスの規則
- 基本的要素
- 式
- 条件

SQL 言語は、データベースおよびデータベースオブジェクトを操作するために使用するコマンドおよび関数から構成されています。SQL 言語はまた、データ型、式、およびリテラルに関するルールを適用します。

SQL リファレンスの規則

このセクションでは、SQL リファレンスのセクションに記載されている SQL の式、コマンド、および関数の構文を記述する際に使用される規則について説明します。

文字	説明
CAPS	大文字の単語はキーワードです。
[ ]	角括弧はオプションの引数を示します。角括弧に複数の引数が含まれる場合は、任意の個数の引数を選択できることを示します。さらに、複数の行に角括弧で囲まれた引数がある場合、Amazon Redshift パーサーは、それらの引数が構文の順番どおりに出現するものと想定します。例については、「SELECT」を参照してください。
{ }	中括弧は、括弧内の引数の 1 つを選択する必要があることを示します。
|	縦線は、どちらかの引数を選択できることを示します。
イタリック	イタリック体の単語は、プレースホルダーを示します。イタリック体の単語の場所に適切な値を挿入する必要があります。
. . .	省略符号は、先行する要素の繰り返しが可能であることを示します。
'	一重引用符に囲まれた単語は、引用符の入力が必要であることを示します。

基本的要素

トピック

- 名前と識別子
- リテラル
- Null
- データ型

- 照合順序

このセクションでは、データベースオブジェクトの名前、リテラル、Null、およびデータ型を操作するためのルールについて説明します。

名前と識別子

名前は、データベースオブジェクト (テーブルや列など) と、ユーザーおよびパスワードを識別します。名前という用語と識別子という用語は、ほとんど同じ意味で使用できます。2 種類の識別子があります。標準的な識別子と、引用符で囲まれた (すなわち、区切り記号付き) 識別子です。識別子は UTF-8 印字可能文字のみで構成する必要があります。標準的な識別子と区切り記号付き識別子の ASCII 文字は、大文字と小文字の区別がありませんが、データベースでは小文字で表記されます。クエリの結果では、列名は、デフォルトで小文字で返されます。列名を大文字で返すには、describe_field_name_in_uppercase 設定パラメータを **true** に設定します。

標準的な識別子

標準的な SQL 識別子はルールセットに準拠するものであり、以下の条件を満たしている必要があります。

- ASCII のシングルバイトのアルファベット文字または下線文字、または UTF-8 のマルチバイト文字 (2 バイトから 4 バイト) で開始します。
- 後続の文字には、ASCII のシングルバイトの英数字、下線、またはドル記号、またはマルチバイトの UTF-8 文字 (2 バイトから 4 バイト) を使用できます。
- 長さが 1～127 バイトで、区切り記号として引用符を含まない。
- 引用符とスペースは含めない。
- 予約された SQL キーワードではない。

区切り記号付き識別子

区切り記号付き識別子 (引用符で囲まれた識別子とも言う) は、二重引用符 (") で囲まれます。区切り記号付き識別子を使用する場合は、そのオブジェクトへのすべての参照で二重引用符を使用する必要があります。この識別子には、二重引用符以外の標準の UTF-8 印字可能文字を含めることができます。したがって、任意の文字列 (スペースやパーセント記号など本来なら無効な文字も含む) で列またはテーブルの名前を作成できます。

区切り記号付き識別子の ASCII 文字は、大文字と小文字の区別がありませんが、小文字で表記されます。文字列内で二重引用符を使用するには、その二重引用符の前に別の二重引用符文字を付ける必要があります。

大文字と小文字を区別する識別子

大文字と小文字を区別する識別子 (大文字と小文字が混在する識別子とも呼ぶ) には、大文字と小文字の両方を使用することができます。大文字と小文字を区別する識別子を使用するには、構成 `enable_case_sensitive_identifier` を `true` に設定します。この構成は、クラスターまたはセッションに対して設定できます。詳細については、「Amazon Redshift 管理ガイド」の「デフォルトパラメータ値」および「enable_case_sensitive_identifier」を参照してください。

システム列名

ただし、次の PostgreSQL システム列の名前を、ユーザー定義の列の名前として使用することはできません。詳細については、https://www.postgresql.org/docs/8.0/static/ddl-system-columns.html を参照してください。

- `oid`
- `tableoid`
- `xmin`
- `cmin`
- `xmax`
- `cmax`
- `ctid`

例

次の表に、区切り記号付き識別子の例、結果として生じる出力、および説明を示します。

構文	結果	説明
"group"	グループ	GROUP は予約語であるため、これを識別子の中で使用するには二重引用符が必要です。
"""WHERE"""	"where"	WHERE も予約語です。文字列内に引用符を含めるには、二重引用符それぞれに二重引用符を付けてエスケープします。

構文	結果	説明
"This name"	this name	スペースを保持するために二重引用符が必要です。
"This ""IS IT"""	this "is it"	IS IT を囲んでいる各引用符を名前の一部とするには、その各引用符の前にそれぞれ追加の引用符を配置する必要があります。

this "is it" という名前の列を持つ group という名前のテーブルを作成するには、以下のように記述します。

```
create table "group" (
"This ""IS IT""" char(10));
```

次のクエリは同じ結果を返します。

```
select "This ""IS IT"""
from "group";

this "is it"
--------------
(0 rows)
```

```
select "this ""is it"""
from "group";

this "is it"
--------------
(0 rows)
```

次に示す完全に修飾された `table.column` 構文も同じ結果を返します。

```
select "group"."this ""is it"""
from "group";

this "is it"
--------------
(0 rows)
```

次の CREATE TABLE コマンドは、列名にスラッシュを含むテーブルを作成します。

```
create table if not exists city_slash_id(
                  "city/id" integer not null,
                  state char(2) not null);
```

リテラル

リテラルまたは定数は固定データ値であり、一連の文字または数値定数から構成されます。Amazon Redshift は、次のようないくつかのタイプのリテラルをサポートしています。

- 数値リテラル。整数、10 進数、および浮動小数点数に使用されます。詳細については、「整数リテラルと浮動小数点リテラル」を参照してください。
- 文字リテラル。文字列、キャラクタ文字列、または文字定数とも呼ばれます。
- 日時データ型で使用される、日時および間隔のリテラル。詳細については、「日付、時刻、およびタイムスタンプのリテラル」および「間隔のデータ型とリテラル」を参照してください。

Null

行内で列が見つからない場合、列が不明である場合、または列が適用できない場合、その列は Null 値であり、または Null を含んでいるといわれます。Null は、主キー制約または NOT NULL 制約によって制限されないデータ型のフィールドに表示される場合があります。Null は値ゼロまたは空の文字列と等価ではありません。

Null を含む演算式の結果はいずれも常に Null になります。Null の引数またはオペランドが指定された場合、連結を除く演算子はすべて Null を返します。

Null かどうかをテストするには、比較条件 IS NULL および IS NOT NULL を使用します。Null はデータが不足していることを表現するものなので、Null はいずれかの値または別の Null に等しいわけでも等しくないわけでもありません。

データ型

トピック

- マルチバイト文字
- 数値型
- 文字型
- 日時型

- ブール型
- HLLSKETCH タイプ
- SUPER タイプ
- VARBYTE 型
- 型の互換性と変換

Amazon Redshift によって保存または取得される各値は、データ型と一定の関連するプロパティセットを持ちます。データ型はテーブルの作成時に宣言されます。データ型は、列または引数に含めることができる値セットを制限します。

次の表に、Amazon Redshift テーブルで使用できるデータ型を一覧表示します。

データ型	エイリアス	説明
SMALLINT	INT2	符号付き 2 バイト整数
INTEGER	INT、INT4	符号付き 4 バイト整数
BIGINT	INT8	符号付き 8 バイト整数
DECIMAL	NUMERIC	精度の選択が可能な真数
REAL	FLOAT4	単精度浮動小数点数
DOUBLE PRECISION	FLOAT8、FLOAT	倍精度浮動小数点数
CHAR	CHARACTER、NCHAR、BPCHAR	固定長のキャラクタ文字列
VARCHAR	CHARACTER VARYING、NVARCHAR、TEXT	ユーザーによって定義された制限を持つ可変長キャラクタ文字列
DATE		カレンダー日付 (年、月、日)
TIME	TIME WITHOUT TIME ZONE	時刻
TIMETZ	TIME WITH TIME ZONE	時刻 (タイムゾーン付き)

データ型	エイリアス	説明
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	日付と時刻 (タイムゾーンなし)
TIMESTAMPTZ	TIMESTAMP WITH TIME ZONE	日付と時刻 (タイムゾーンあり)
INTERVAL YEAR TO MONTH		年から月への順での期間
INTERVAL DAY TO SECOND		日から秒への順での期間
BOOLEAN	BOOL	論理ブール演算型 (true/false)
HLLSKETCH		HyperLogLog スケッチで使用するタイプ。
SUPER		ARRAY や STRUCTS などの複合型が含まれる Amazon Redshift のすべてのスカラー型を包含するスーパーセットデータ型。
VARBYTE	VARBINARY、BINARY VARYING	可変長バイナリ値
GEOMETRY		空間データ
GEOGRAPHY		空間データ

Note

サポートされていないデータ型 (「char」は引用符で囲まれています) については、サポートされていない PostgreSQL データ型 を参照してください。

マルチバイト文字

VARCHAR データ型では、最大 4 バイトの UTF-8 マルチバイト文字をサポートします。5 バイト以上の文字はサポートされていません。マルチバイト文字を含む VARCHAR 列のサイズを計算するには、文字数と 1 文字当たりのバイト数を掛けます。例えば、文字列に漢字が 4 文字含まれ、各文字のサイズが 3 バイトである場合は、文字列を格納するのに VARCHAR(12) 列が必要です。

VARCHAR データ型は、次に示す無効な UTF-8 コードポイントをサポートしていません:

`0xD800 – 0xDFFF`(バイトシーケンス:`ED A0 80` – `ED BF BF`)

CHAR データ型は、マルチバイト文字をサポートしていません。

数値型

トピック

- 整数型
- DECIMAL 型または NUMERIC 型
- 128 ビットの DECIMAL または NUMERIC の列の使用に関する注意事項
- 浮動小数点型
- 数値に関する計算
- 整数リテラルと浮動小数点リテラル
- 数値型を使用する例

数値データ型には、整数型、10 進数型、および浮動小数点数型などがあります。

整数型

SMALLINT、INTEGER、および BIGINT の各データ型を使用して、各種範囲の数値全体を格納します。各データ型の許容範囲の外にある値を格納することはできません。

名前	ストレージ	範囲
SMALLINT または INT2	2 バイト	-32768 ~ +32767
INTEGER、INT、または INT4	4 バイト	-2147483648 ~ +2147483647

名前	ストレージ	範囲
BIGINT または INT8	8 バイト	-9223372036854775808 ~ 9223372036854775807

DECIMAL 型または NUMERIC 型

DECIMAL データ型または NUMERIC データ型を使用し、ユーザー定義の精度で値を格納します。DECIMAL キーワードと NUMERIC キーワードは、ほぼ同じ意味で使用されます。このドキュメントでは、このデータ型を表す用語として decimal を優先的に使用します。numeric という用語は一般的に整数、10 進数、および浮動小数点のデータ型を称する場合に使用します。

ストレージ	範囲
可変。非圧縮の DECIMAL 型の場合は最大 128 ビット。	最大で 38 桁の精度を持つ、128 ビットの符号付き整数。

テーブル内に DECIMAL 列を定義するには、precision と scale を次のように指定します。

```
decimal(precision, scale)
```

precision

値全体での有効な桁の合計。小数点の両側の桁数。例えば、数値 `48.2891` の場合は精度が 6、スケールが 4 となります。指定がない場合、デフォルトの精度は 18 です。最大精度は 38 です。

入力値で小数点の左側の桁数が、列の精度から列のスケールを引いて得られた桁数を超えている場合、入力値を列にコピー (または挿入も更新も) することはできません。このルールは、列の定義を外れるすべての値に適用されます。例えば、`numeric(5,2)` 列の値の許容範囲は、`-999.99`～`999.99` です。

scale

小数点の右側に位置する、値の小数部における小数の桁数です。整数のスケールはゼロです。列の仕様では、スケール値は精度値以下である必要があります。指定がなければ、デフォルトのスケールは 0 です。最大スケールは 37 です。

テーブルにロードされた入力値のスケールが列のスケールより大きい場合、値は指定されたスケールに丸められます。SALES テーブルの PRICEPAID 列が DECIMAL(8,2) 列である場合を例にとります。DECIMAL(8,4) の値を PRICEPAID 列に挿入すると、値のスケールは 2 に丸められます。

```
insert into sales
values (0, 8, 1, 1, 2000, 14, 5, 4323.8951, 11.00, null);

select pricepaid, salesid from sales where salesid=0;

pricepaid | salesid
-----------+---------
4323.90 |       0
(1 row)
```

ただし、テーブルから選択された値の明示的なキャストの結果は丸められません。

> **Note**
>
> DECIMAL(19,0) 列に挿入できる最大の正の値は、9223372036854775807 (2^{63} -1) です。最大の負の値は -9223372036854775808 です。例えば、値 9999999999999999999 (19 桁の 9 の並び) の挿入を試みると、オーバーフローエラーが発生します。小数点の位置に関係なく、Amazon Redshift が DECIMAL 数として表現できる最大の文字列は 9223372036854775807 です。例えば、DECIMAL(19,18) 列にロードできる最大値は 9.223372036854775807 です。
> これらの規則は、有効桁数が 19 桁以下の DECIMAL 値は内部で 8 バイトの整数として格納され、有効桁数が 20～38 桁の DECIMAL 値は 16 バイトの整数として格納されるためです。

128 ビットの DECIMAL または NUMERIC の列の使用に関する注意事項

アプリケーションがそのような精度を必要とすることが明確でない限り、最大精度を DECIMAL 列に任意に割り当てないでください。128 ビット値は、64 ビット値の 2 倍のディスク容量を使用するので、クエリの実行時間が長くなる可能性があります。

浮動小数点型

可変精度の数値を格納するには、REAL および DOUBLE PRECISION のデータ型を使用します。これらのデータ型は非正確型です。すなわち、一部の値が近似値として格納されるため、特定の値を格

納して返すと若干の相違が生じる場合があります。正確な格納および計算が必要な場合は (金額の場合など)、DECIMAL データ型を使用します。

REAL は、IEEE 標準 754 の 2 進浮動小数点演算に従って単精度浮動小数点形式を表します。精度は約 6 桁で、範囲は約 1E-37 から 1E+37 です。このデータ型を FLOAT4 として指定することもできます。

DOUBLE PRECISION は、IEEE 標準 754 の 2 進浮動小数点演算に従って倍精度浮動小数点形式を表します。精度は約 15 桁で、範囲は約 1E-307 から 1E+308 です。このデータ型を FLOAT または FLOAT8 として指定することもできます。

通常の数値に加えて、浮動小数点型にはいくつかの特別な値があります。SQL でこれらの値を使用するときは、一重引用符で囲んでください。

- `NaN` - Not a Number (非数)
- `Infinity` — Infinity (無限大)
- `-Infinity` — Negative Infinity (負の無限大)

例えば、`customer_activity` テーブルの `day_charge` 列に非数を挿入するには、次の SQL を実行します。

```
insert into customer_activity(day_charge) values('NaN');
```

数値に関する計算

ここで、計算とは加算、減算、乗算、および除算といった 2 進算術演算を示しています。このセクションでは、このような演算の予期される戻り型について、さらに DECIMAL データ型が必要な場合に精度とスケールを決定するのに適用される特定の計算式について説明します。

クエリ処理中に数値の計算が行われる場合には、計算が不可能であるためにクエリが数値オーバーフローエラーを返すといった状況が発生することがあります。さらに、算出される値のスケールが、変化したり予期せぬものであったりする状況が発生することもあります。一部の演算については、明示的なキャスト (型の上位変換) または Amazon Redshift 設定パラメータを使用してこれらの問題を解決できます。

SQL 関数を使用した同様の計算の結果の詳細については、「集計関数」を参照してください。

計算の戻り型

Amazon Redshift で一連の数値データ型がサポートされていると仮定して、次の表に加算、減算、乗算、および除算の予期される戻り型を示します。表の左側の最初の列は計算の 1 番目のオペランドを示し、一番上の行は 2 番目のオペランドを示します。

オペランド 1	オペランド 2	戻り型
INT2	INT2	INT2
INT2	INT4	INT4
INT2	INT8	INT8
INT2	DECIMAL	DECIMAL
INT2	FLOAT4	FLOAT8
INT2	FLOAT8	FLOAT8
INT4	INT4	INT4
INT4	INT8	INT8
INT4	DECIMAL	DECIMAL
INT4	FLOAT4	FLOAT8
INT4	FLOAT8	FLOAT8
INT8	INT8	INT8
INT8	DECIMAL	DECIMAL
INT8	FLOAT4	FLOAT8
INT8	FLOAT8	FLOAT8
DECIMAL	DECIMAL	DECIMAL
DECIMAL	FLOAT4	FLOAT8

オペランド 1	オペランド 2	戻り型
DECIMAL	FLOAT8	FLOAT8
FLOAT4	FLOAT8	FLOAT8
FLOAT8	FLOAT8	FLOAT8

DECIMAL の計算結果の精度とスケール

次の表に、算術演算で DECIMAL 型の結果が返されるときに算出結果の精度とスケールに適用されるルールの概要を示します。この表で、`p1` と `s1` は計算における 1 番目のオペランドの精度とスケールを示し、`p2` と `s2` は 2 番目のオペランドの精度とスケールを示します (これらの計算に関係なく、結果の最大精度は 38、結果の最大スケールは 38 です)。

オペレーション	結果の精度とスケール
+ または -	スケール = `max(s1,s2)` 精度 = `max(p1-s1,p2-s2)+1+scale`
*	スケール = `s1+s2` 精度 = `p1+p2+1`
/	スケール = `max(4,s1+p2-s2+1)` 精度 = `p1-s1+ s2+scale`

例えば、SALES テーブルの PRICEPAID 列と COMMISSION 列は両方とも DECIMAL(8,2) 列です。PRICEPAID を COMMISSION で除算する場合 (または COMMISSION を PRICEPAID で除算する場合)、計算式は次のように適用されます。

```
Precision = 8-2 + 2 + max(4,2+8-2+1)
= 6 + 2 + 9 = 17

Scale = max(4,2+8-2+1) = 9
```

```
Result = DECIMAL(17,9)
```

次の計算は、UNION、INTERSECT、EXCEPT などの集合演算子および COALESCE や DECODE などの関数を使用して DECIMAL 値に対して演算を実行した場合に、結果として生じる精度とスケールを計算するための汎用的なルールです。

```
Scale = max(s1,s2)
Precision = min(max(p1-s1,p2-s2)+scale,19)
```

例えば、DECIMAL(7,2) 列が 1 つ含まれる DEC1 テーブルと、DECIMAL(15,3) 列が 1 つ含まれる DEC2 テーブルを結合して DEC3 テーブルを作成します。DEC3 のスキーマを確認すれば、列が NUMERIC(15,3) 列になることが分かります。

```
create table dec3 as select * from dec1 union select * from dec2;
```

結果

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'dec3';

column |     type      | encoding | distkey | sortkey
-------+---------------+----------+---------+---------
c1     | numeric(15,3) | none     | f       | 0
```

上記の例では、計算式は次のように適用されます。

```
Precision = min(max(7-2,15-3) + max(2,3), 19)
= 12 + 3 = 15

Scale = max(2,3) = 3

Result = DECIMAL(15,3)
```

除算演算に関する注意事項

除算演算の場合、ゼロ除算を行うとエラーが返されます。

精度とスケールが計算されると、スケール制限 100 が適用されます。算出された結果スケールが 100 より大きい場合、除算結果は次のようにスケーリングされます。

- 精度 = `precision - (scale - max_scale)`
- スケール = `max_scale`

算出された精度が最大精度 (38) より高い場合、精度は 38 に引き下げられ、スケールは `max((38 + scale - precision), min(4, 100))` で計算された結果となります。

オーバーフロー条件

すべての数値計算についてオーバーフローが調べられます。精度が 19 以下の DECIMAL データは 64 ビットの整数として格納されます。精度が 19 を上回る DECIMAL データは 128 ビットの整数として格納されます。すべての DECIMAL 値の最大精度は 38 であり、最大スケールは 37 です。値がこれらの制限を超えるとオーバーフローエラーが発生します。このルールは中間結果セットと最終結果セットの両方に適用されます。

- 特定のデータ値がキャスト関数で指定された所定の精度またはスケールに適合していない場合、明示的なキャストでは実行時オーバーフローエラーが生じます。例えば、SALES テーブルの PRICEPAID 列 (DECIMAL(8,2) 列) からの値をすべてキャストできるとは限らないので、結果として DECIMAL(7,3) を返すことはできません。

  ```
  select pricepaid::decimal(7,3) from sales;
  ERROR:  Numeric data overflow (result precision)
  ```

  このエラーは、PRICEPAID 列に含まれる大きな値のいくつかをキャストできないために発生します。

- 乗算演算によって得られる結果では、結果スケールが各オペランドのスケールを足し算した値となります。例えば、両方のオペランドとも 4 桁のスケールを持っている場合、結果としてスケールは 8 桁となり、小数点の左側には 10 桁のみが残ります。したがって、有効スケールを持つ 2 つの大きな数値を乗算する場合は、比較的簡単にオーバーフロー状態に陥ります。

  次の例はオーバーフローエラーになります。

  ```
  SELECT CAST(1 AS DECIMAL(38, 20)) * CAST(10 AS DECIMAL(38, 20));
  ERROR: 128 bit numeric data overflow (multiplication)
  ```

  乗算の代わりに除算を使用することでオーバーフローエラーを回避できます。次の例を使用して、1 を元の除数で割って除算します。

  ```
  SELECT CAST(1 AS DECIMAL(38, 20)) / (1 / CAST(10 AS DECIMAL(38, 20)));
  ```

```
+----------+
| ?column? |
+----------+
| 10       |
+----------+
```

INTEGER 型および DECIMAL 型での数値計算

計算式のオペランドの一方が INTEGER データ型を持っており、他方のオペランドが DECIMAL データ型を持っている場合、INTEGER オペランドは DECIMAL として暗黙的にキャストされます。

- INT2 (SMALLINT) は、DECIMAL(5,0) としてキャストされます。
- INT4 (INTEGER) は、DECIMAL(10,0) としてキャストされます。
- INT8 (BIGINT) は、DECIMAL(19,0) としてキャストされます。

例えば、DECIMAL(8,2) 列の SALES.COMMISSION と、SMALLINT 列の SALES.QTYSOLD を乗算する場合、この計算は次のようにキャストされます。

```
DECIMAL(8,2) * DECIMAL(5,0)
```

整数リテラルと浮動小数点リテラル

数値を表現するリテラルまたは定数は、整数または浮動小数点数になり得ます。

整数リテラル

整数定数は一連の 0～9 の数字であり、この一連の数字の先頭にはオプションで正 (+) 符号または負 (-) 符号が付けられます。

構文

```
[ + | - ] digit ...
```

例

以下は有効な整数です。

```
23
-555
```

```
+17
```

浮動小数点リテラル

浮動小数点リテラル (10 進リテラル、数値リテラル、分数リテラルとも呼ばれる) は、小数点を含むことができるほか、必要に応じて指数マーカー (e) も含むことができる一連の数字です。

構文

```
[ + | - ] digit ... [ . ] [ digit ...]
[ e | E [ + | - ] digit ... ]
```

引数

e | E

e または E は、数値が科学的表記で指定されていることを示します。

例

以下は有効な浮動小数点リテラルです。

```
3.14159
-37.
2.0e19
-2E-19
```

数値型を使用する例

CREATE TABLE ステートメント

次の CREATE TABLE ステートメントでは、さまざまな数値データ型を実際に宣言しています。

```
create table film (
film_id integer,
language_id smallint,
original_language_id smallint,
rental_duration smallint default 3,
rental_rate numeric(4,2) default 4.99,
length smallint,
```

```
replacement_cost real default 25.00);
```

範囲外の整数を挿入する試み

次の例では、値 33000 を SMALLINT 列に挿入しようとしています。

```
insert into film(language_id) values(33000);
```

SMALLINT の範囲は、-32768～+32767 であるため、Amazon Redshift はエラーを返します。

```
An error occurred when executing the SQL command:
insert into film(language_id) values(33000)

ERROR: smallint out of range [SQL State=22003]
```

整数列への 10 進値の挿入

次の例では、10 進値を INT 列に挿入します。

```
insert into film(language_id) values(1.5);
```

この値は挿入されますが、整数値 2 に切り上げられます。

10 進値のスケールが丸められるため挿入に成功する場合

次の例では、列よりも上位の精度を持つ 10 進値を挿入します。

```
insert into film(rental_rate) values(35.512);
```

この例では、値 `35.51` が列に挿入されます。

範囲外の 10 進値を挿入する試み

この場合、値 `350.10` は範囲外です。DECIMAL 列内の値の桁数は、列の精度からそのスケールを引いた結果となります (RENTAL_RATE 列の場合は 4 から 2 を引いた結果)。すなわち、`DECIMAL(4,2)` 列の許容範囲は、`-99.99`～`99.99` です。

```
insert into film(rental_rate) values (350.10);
ERROR:  numeric field overflow
DETAIL:  The absolute value is greater than or equal to 10^2 for field with precision
 4, scale 2.
```

REAL 列への可変精度値の挿入

次の例では可変精度値を REAL 列に挿入します。

```
insert into film(replacement_cost) values(1999999.99);

insert into film(replacement_cost) values(1999.99);

select replacement_cost from film;

+-------------------+
| replacement_cost |
+-------------------+
| 2000000          |
| 1999.99          |
+-------------------+
```

値 1999999.99 は、REAL 列の精度要件を満たすために 2000000 に変換されます。値 1999.99 はそのままロードされます。

文字型

トピック

- ストレージと範囲
- CHAR または CHARACTER
- VARCHAR または CHARACTER VARYING
- NCHAR 型および NVARCHAR 型
- TEXT 型および BPCHAR 型
- 末尾の空白の重要性
- 文字型を使用する例

文字データ型には、CHAR (文字) や VARCHAR (可変文字) などがあります。

ストレージと範囲

CHAR および VARCHAR のデータ型は、文字単位でなくバイト単位で定義されます。CHAR 列にはシングルバイト文字のみを含めることができます。したがって、CHAR(10) 列には、最大 10 バイト長の文字列を含めることができます。VARCHAR にはマルチバイト文字 (1 文字あたり最大で 4 バ

イトまで) を含めることができます。例えば、VARCHAR(12) 列には、シングルバイト文字なら 12 個、2 バイト文字なら 6 個、3 バイト文字なら 4 個、4 バイト文字なら 3 個含めることができます。

名前	ストレージ	範囲 (列の幅)
CHAR、CHARACTER、NCHAR	文字列の長さ。末尾の空白を含む (存在する場合)。	4096 バイト
VARCHAR、CHARACTER VARYING、NVARCHAR	4 バイト + 文字の合計バイト数 (ここで、各文字は 1～4 バイト)。	65535 バイト (64K -1)
BPCHAR	固定長の CHAR(256) に変換されます。	256 バイト
TEXT	VARCHAR(256) に変換されます。	260 バイト

Note

CREATE TABLE 構文では、文字データ型の MAX キーワードをサポートします。次に例を示します。

```
create table test(col1 varchar(max));
```

MAX 設定は列幅を定義します。CHAR の場合は 4096 バイトであり、VARCHAR の場合は 65535 となります。

CHAR または CHARACTER

固定長の文字列を格納するには、CHAR または CHARACTER を使用します。これらの文字列は空白で埋められるので、CHAR(10) 列は常に 10 バイトのストレージを占有します。

```
char(10)
```

長さの指定がない場合 CHAR 列は、CHAR(1) 列になります。

VARCHAR または CHARACTER VARYING

一定の制限を持つ可変長の文字列を格納するには、VARCHAR 列または CHARACTER VARYING 列を使用します。これらの文字列は空白で埋められないので、VARCHAR(120) 列は、最大で 120 個のシングルバイト文字、60 個の 2 バイト文字、40 個の 3 バイト文字、または 30 個の 4 バイト文字で構成されます。

```
varchar(120)
```

CREATE TABLE ステートメントで長さの指定子なしで VARCHAR データ型を使用すると、デフォルト長は 256 になります。式で使用する場合、出力のサイズは入力式 (最大 65535) を使用して決定されます。

NCHAR 型および NVARCHAR 型

NCHAR 型および NVARCHAR 型 (NATIONAL CHARACTER 型および NATIONAL CHARACTER VARYING 型と呼ぶこともある) で、列を作成できます。これらの型はそれぞれ CHAR 型と VARCHAR 型に変換され、指定されたバイト数で格納されます。

長さを指定しない場合、NCHAR 列は CHAR(1) 列に変換されます。

長さを指定しない場合、NVARCHAR 列は VARCHAR(256) 列に変換されます。

TEXT 型および BPCHAR 型

TEXT 列を使用して Amazon Redshift テーブルを作成できますが、この列は最大 256 文字の可変長値を受け入れる VARCHAR(256) 列に変換されます。

BPCHAR (空白で埋められる文字) 型を使用して Amazon Redshift 列を作成できますが、この列によって固定長の CHAR(256) 列に変換されます。

末尾の空白の重要性

CHAR と VARCHAR のデータ型は両方とも、最大 n バイト長の文字列を格納できます。それよりも長い文字列をこれらの型の列に格納しようとすると、エラーが発生します。ただし、余分な文字がすべてスペース (空白) ならば例外で、文字列は最大長に切り捨てられます。文字列の長さが最大長よりも短い場合、CHAR 値は空白で埋められますが、VARCHAR 値では空白なしで文字列を格納します。

CHAR 値の末尾の空白はいつも意味的に重要であるとは限りません。末尾の空白は、LENGTH 計算を含めないで 2 つの CHAR 値を比較するときは無視され、CHAR 値を別の文字列型に変換するときは削除されます。

VARCHAR および CHAR の値の末尾の空白は、値が比較されるとき、意味的に重要でないものとして扱われます。

長さの計算によって返される VARCHAR キャラクタ文字列の長さには末尾の空白が含められます。固定長のキャラクタ文字列の長さを計算する場合、末尾の空白はカウントされません。

文字型を使用する例

CREATE TABLE ステートメント

次の CREATE TABLE ステートメントでは、VARCHAR および CHAR データタイプの使用を示しています。

```
create table address(
address_id integer,
address1 varchar(100),
address2 varchar(50),
district varchar(20),
city_name char(20),
state char(2),
postal_code char(5)
);
```

次の例ではこの表を使用しています。

可変キャラクタ文字列における末尾の空白

ADDRESS1 は VARCHAR 列であるため、2 番目の挿入アドレスの末尾の空白は意味的に重要ではありません。すなわち、これらの 2 つの挿入アドレスは一致します。

```
insert into address(address1) values('9516 Magnolia Boulevard');

insert into address(address1) values('9516 Magnolia Boulevard  ');
```

```
select count(*) from address
where address1='9516 Magnolia Boulevard';

count
-------
2
(1 row)
```

もし ADDRESS1 列が CHAR 列であり、同じ値が挿入されたと仮定すると、COUNT(*) クエリはキャラクタ文字列を同じものとして認識し、2 を返します。

LENGTH 関数の結果

LENGTH 関数は、VARCHAR 列内の末尾の空白を認識します。

```
select length(address1) from address;

length
--------
23
25
(2 rows)
```

CHAR 列である CITY_NAME 列の `Augusta` の値は、入力文字列内の末尾の空白に関係なく常に 7 文字になります。

列の長さを超える値

文字列は、宣言した列幅に適合するように切り捨てられることはありません。

```
insert into address(city_name) values('City of South San Francisco');
ERROR: value too long for type character(20)
```

この問題の解決策は、値を列のサイズにキャストすることです。

```
insert into address(city_name)
values('City of South San Francisco'::char(20));
```

この場合は、文字列の最初の 20 文字 (`City of South San Fr`) が列にロードされます。

日時型

トピック

- ストレージと範囲
- DATE
- TIME
- TIMETZ
- TIMESTAMP
- TIMESTAMPTZ
- 日時型を使用する例
- 日付、時刻、およびタイムスタンプのリテラル
- 間隔のデータ型とリテラル

日時データ型には DATE、TIME、TIMETZ、TIMESTAMP、TIMESTAMPTZ があります。

ストレージと範囲

名前	ストレージ	範囲	解像度
DATE	4 バイト	4713 BC ~ 294276 AD	1 日
TIME	8 バイト	00:00:00 ~ 24:00:00	1 マイクロ秒
TIMETZ	8 バイト	00:00:00 + 1459 ~ 00:00:00 + 1459	1 マイクロ秒
TIMESTAMP	8 バイト	4713 BC ~ 294276 AD	1 マイクロ秒
TIMESTAMPTZ	8 バイト	4713 BC ~ 294276 AD	1 マイクロ秒

DATE

タイムスタンプなしで単純にカレンダー日付だけを保存するには DATE データ型を使用します。

TIME

TIME は、TIME WITHOUT TIME ZONE のエイリアスです。

時刻を保存するには、TIME データ型を使用します。

TIME 列に値を保存する場合、小数秒の精度については最大で 6 桁まで保存されます。

デフォルトでは、ユーザーテーブルと Amazon Redshift システムテーブルでは、TIME 値は協定世界時 (UTC) になります。

TIMETZ

TIMETZ は、TIME WITH TIME ZONE のエイリアスです。

TIMETZ データ型を使用して、時刻とタイムゾーンを保存します。

TIMETZ 列に値を保存する場合、小数秒の精度については最大で 6 桁まで保存されます。

デフォルトでは、ユーザーテーブルと Amazon Redshift システムテーブルの両方で、TIMETZ 値は UTC です。

TIMESTAMP

TIMESTAMP は、TIMESTAMP WITHOUT TIME ZONE のエイリアスです。

日付と時刻を含む完全なタイムスタンプ値を保存するには TIMESTAMP データ型を使用します。

TIMESTAMP 列に値を保存する場合、小数秒の精度については最大で 6 桁まで保存されます。

TIMESTAMP 列に日付または部分的なタイムスタンプ値を持つ日付を挿入すると、値は暗黙的に完全なタイムスタンプ値に変換されます。この完全なタイムスタンプ値には、時間、分、および秒が抜けている場合のデフォルト値 (00) があります。入力文字列のタイムゾーン値は無視されます。

デフォルトでは、ユーザーテーブルと Amazon Redshift システムテーブルの両方で、TIMESTAMP 値は UTC です。

TIMESTAMPTZ

TIMESTAMPTZ は、TIMESTAMP WITH TIME ZONE のエイリアスです。

日付、時刻、タイムゾーンを含む完全なタイムスタンプ値を入力するには TIMESTAMPTZ データ型を使用します。入力値にタイムゾーンが含まれる場合、Amazon Redshift はタイムゾーンを使用して値を UTC に変換し、UTC 値を保存します。

サポートされるタイムゾーン名のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_names();
```

サポートされるタイムゾーン省略形のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_abbrevs();
```

タイムゾーンの最新情報については、「IANA Time Zone Database」も参照してください。

次の表に、タイムゾーン形式の例を示します。

形式	例
dd mon hh:mi:ss yyyy tz	17 Dec 07:37:16 1997 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 PST
mm/dd/yyyy hh:mi:ss.ss tz	12/17/1997 07:37:16.00 US/Pacific
yyyy-mm-dd hh:mi:ss+/-tz	1997-12-17 07:37:16-08
dd.mm.yyyy hh:mi:ss tz	17.12.1997 07:37:16.00 PST

TIMESTAMPTZ 列に値を保存する場合、小数秒の精度については最大で 6 桁まで保存されます。

TIMESTAMPTZ 列に日付または部分的なタイムスタンプを持つ日付を挿入すると、値は暗黙的に完全なタイムスタンプ値に変換されます。この完全なタイムスタンプ値には、時間、分、および秒が抜けている場合のデフォルト値 (00) があります。

TIMESTAMPTZ 値は、ユーザーテーブルでは UTC です。

日時型を使用する例

以下に、Amazon Redshift でサポートされている日時タイプを操作する例を示します。

日付の例

次の例では、形式が異なる複数の日付を挿入して、出力を表示します。

```
create table datetable (start_date date, end_date date);
```

```
insert into datetable values ('2008-06-01','2008-12-31');

insert into datetable values ('Jun 1,2008','20081231');
```

```
select * from datetable order by 1;

start_date |  end_date
-----------------------
2008-06-01 | 2008-12-31
2008-06-01 | 2008-12-31
```

DATE 列にタイムスタンプ値を挿入した場合、時刻部分が無視され、日付のみロードされます。

時間の例

次の例では、形式の異なる複数の TIME および TIMETZ 値を挿入して、出力を表示します。

```
create table timetable (start_time time, end_time timetz);
```

```
insert into timetable values ('19:11:19','20:41:19 UTC');
insert into timetable values ('191119', '204119 UTC');
```

```
select * from timetable order by 1;
start_time |  end_time
------------------------
 19:11:19  | 20:41:19+00
 19:11:19  | 20:41:19+00
```

タイムスタンプの例

日付を TIMESTAMP 列または TIMESTAMPTZ 列に挿入すると、時刻はデフォルトでは午前 0 時になります。例えば、リテラル `20081231` を挿入すると、格納される値は `2008-12-31 00:00:00` です。

現在のセッションのタイムゾーンを変更するには、SET コマンドを使用して timezone 設定パラメータを設定します。

次の例では、形式の異なる複数のタイムスタンプを挿入して、結果のテーブルを表示します。

```
create table tstamp(timeofday timestamp, timeofdaytz timestamptz);

insert into tstamp values('Jun 1,2008  09:59:59', 'Jun 1,2008 09:59:59 EST' );
insert into tstamp values('Dec 31,2008 18:20','Dec 31,2008 18:20');
insert into tstamp values('Jun 1,2008  09:59:59 EST', 'Jun 1,2008 09:59:59');

SELECT * FROM tstamp;

+---------------------+------------------------+
|      timeofday      |      timeofdaytz       |
+---------------------+------------------------+
| 2008-06-01 09:59:59 | 2008-06-01 14:59:59+00 |
| 2008-12-31 18:20:00 | 2008-12-31 18:20:00+00 |
| 2008-06-01 09:59:59 | 2008-06-01 09:59:59+00 |
+---------------------+------------------------+
```

日付、時刻、およびタイムスタンプのリテラル

Amazon Redshift によってサポートされている日付、時刻、およびタイムスタンプリテラルを使用する際に従うべきルールは次のとおりです。

日付

次に示す入力日付はすべて、Amazon Redshift テーブルにロードできる DATE データ型の有効な日付リテラル値の例です。デフォルトの `MDY DateStyle` モードが有効であると想定されます。このモードでは、`1999-01-08` や `01/02/00` などの文字列で月の値が日の値より前にあることを意味します。

> **Note**
>
> 日付またはタイムスタンプのリテラルをテーブルにロードするには、それらのリテラルを引用符で囲む必要があります。

入力日付	完全な日付
January 8, 1999	January 8, 1999
1999-01-08	January 8, 1999

入力日付	完全な日付
1/8/1999	January 8, 1999
01/02/00	2000 年 1 月 2 日
2000-Jan-31	2000 年 1 月 31 日
Jan-31-2000	2000 年 1 月 31 日
31-Jan-2000	2000 年 1 月 31 日
20080215	2008 年 2 月 15 日
080215	2008 年 2 月 15 日
2008.366	2008 年 12 月 31 日 (日付の 3 桁部分は 001～366 である必要があります)

Times

次に示す入力時間はすべて、Amazon Redshift テーブルにロードできる TIME データ型および TIMETZ データ型の有効な時刻リテラル値の例です。

入力時間	説明（時刻部分）
04:05:06.789	午前 4 時 5 分 6.789 秒
04:05:06	午前 4 時 5 分 6 秒
04:05	午前 4 時 5 分ちょうど
040506	午前 4 時 5 分 6 秒
午前 4 時 5 分	午前 4 時 5 分ちょうど、午前はオプション
午後 4 時 5 分	午後 4 時 5 分ちょうど。時値は 12 未満である必要があります。
16:05	午後 4 時 5 分ちょうど

タイムスタンプ

次に示す入力タイムスタンプはすべて、Amazon Redshift テーブルにロードできる TIMESTAMP データ型および TIMESTAMPTZ データ型の有効な時刻リテラル値の例です。有効な日付リテラルはすべて、以下の時刻リテラルと結合することができます。

入力タイムスタンプ（日付と時刻が結合されている）	説明（時刻部分）
20080215 04:05:06.789	午前 4 時 5 分 6.789 秒
20080215 04:05:06	午前 4 時 5 分 6 秒
20080215 04:05	午前 4 時 5 分ちょうど
20080215 040506	午前 4 時 5 分 6 秒
20080215 04:05 AM	午前 4 時 5 分ちょうど、午前はオプション
20080215 04:05 PM	午後 4 時 5 分ちょうど。時値は 12 未満である必要があります。
20080215 16:05	午後 4 時 5 分ちょうど
20080215	午前 0 時 (デフォルト)

特殊な日時値

次に示す特殊な値は、日時リテラルとして、および日付関数に渡す引数として使用できます。このような特殊な値は、一重引用符を必要とし、クエリの処理時に正規のタイムスタンプ値に変換されます。

特殊な値	説明
now	現在のトランザクションの開始時間に等しく、マイクロ秒の精度を持つタイムスタンプを返します。

特殊な値	説明
today	該当する日付に等しく、時刻部分がゼロのタイムスタンプを返します。
tomorrow	該当する日付に等しく、時刻部分がゼロのタイムスタンプを返します。
yesterday	該当する日付に等しく、時刻部分がゼロのタイムスタンプを返します。

次の例では、now および today が DATEADD 関数でどのように動作するかを示しています。

```
select dateadd(day,1,'today');

date_add
---------------------
2009-11-17 00:00:00
(1 row)

select dateadd(day,1,'now');

date_add
----------------------------
2009-11-17 10:45:32.021394
(1 row)
```

間隔のデータ型とリテラル

間隔のデータ型を使用して、seconds、minutes、hours、days、months、years などの単位で期間を保存できます。間隔のデータ型とリテラルは、日付とタイムスタンプへの間隔の追加、間隔の合計、日付またはタイムスタンプからの間隔の減算など、日時の計算に使用できます。間隔リテラルは、テーブル内の間隔データ型列への入力値として使用できます。

間隔データ型の構文

期間を年数と月数で保存する間隔データ型を指定するには:

```
INTERVAL year_to_month_qualifier
```

期間を日数、時間数、分数、および秒数で保存する間隔データ型を指定するには:

```
INTERVAL day_to_second_qualifier [ (fractional_precision) ]
```

間隔リテラルの構文

間隔リテラルを指定して、期間を年数と月数で定義するには:

```
INTERVAL quoted-string year_to_month_qualifier
```

間隔リテラルを指定して、期間を日数、時間数、分数、および秒数で定義するには:

```
INTERVAL quoted-string day_to_second_qualifier [ (fractional_precision) ]
```

引数

quoted-string

数量と日時単位を入力文字列として指定する正または負の数値を指定します。quoted-string に数値のみが含まれている場合、Amazon Redshift は year_to_month_qualifier または day_to_second_qualifier から単位を決定します。例えば、`'23' MONTH` は `1 year 11 months`、`'-2' DAY` は `-2 days 0 hours 0 minutes 0.0 seconds`、`'1-2' MONTH` は `1 year 2 months`、`'13 day 1 hour 1 minute 1.123 seconds' SECOND` は `13 days 1 hour 1 minute 1.123 seconds` を表します。間隔の出力形式の詳細については、「間隔スタイル」を参照してください。

year_to_month_qualifier

間隔の範囲を指定します。修飾子を使用して、修飾子よりも小さい時間単位で間隔を作成すると、Amazon Redshift ではそれより小さい間隔の単位を切り捨てて破棄します。year_to_month_qualifier の有効な値は次のとおりです。

- `YEAR`
- `MONTH`
- `YEAR TO MONTH`

day_to_second_qualifier

間隔の範囲を指定します。修飾子を使用して、修飾子よりも小さい時間単位で間隔を作成すると、Amazon Redshift ではそれより小さい間隔の単位を切り捨てて破棄します。day_to_second_qualifier の有効な値は次のとおりです。

- DAY
- HOUR
- MINUTE
- SECOND
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE TO SECOND

INTERVAL リテラルの出力は、指定された最小の INTERVAL コンポーネントで切り捨てられます。例えば、MINUTE 修飾子を使用すると、Amazon Redshift は MINUTE より小さい時間単位を破棄します。

```
select INTERVAL '1 day 1 hour 1 minute 1.123 seconds' MINUTE
```

結果の値は `'1 day 01:01:00'` で切り捨てられます。

Fractional_precision

間隔で使用できる小数点以下の桁数を指定するオプションのパラメータ。fractional_precision 引数は、間隔に SECOND が含まれている場合にのみ指定する必要があります。例えば、`SECOND(3)` は、1.234 秒など、小数点以下 3 桁のみを許可する間隔を作成します。小数点以下の最大桁数は 6 です。

セッション設定 `interval_forbid_composite_literals` は、YEAR TO MONTH と DAY TO SECOND の両方のパートを使用して間隔を指定している場合に、エラーを返すかどうかを決定します。詳細については、「interval_forbid_composite_literals」を参照してください。

区間演算

間隔の値を他の日時値と一緒に使用して算術演算を実行できます。次の表は、使用可能な演算と、各演算の結果であるデータ型を示しています。

> **Note**
>
> date と timestamp の両方の結果を生成できる演算は、式に関連する最小時間単位に基づいて行われます。例えば、interval を date に追加すると、YEAR TO MONTH 間隔の場合、結果は date になり、DAY TO SECOND 間隔の場合、結果はタイムスタンプになります。

最初のオペランドが interval の演算は、指定された 2 番目のオペランドに対して次の結果を生成します。

演算子	日付	タイムスタンプ	間隔	数値
-	該当なし	該当なし	間隔	該当なし
+	日付	日付/タイムスタンプ	間隔	該当なし
*	該当なし	該当なし	該当なし	間隔
/	該当なし	該当なし	該当なし	間隔

最初のオペランドが date の演算は、指定された 2 番目のオペランドに対して次の結果を生成します。

演算子	日付	タイムスタンプ	間隔	数値
-	数値	間隔	日付/タイムスタンプ	日付
+	該当なし	該当なし	該当なし	該当なし

最初のオペランドが timestamp の演算は、指定された 2 番目のオペランドに対して次の結果を生成します。

演算子	日付	タイムスタンプ	間隔	数値
-	数値	間隔	タイムスタンプ	タイムスタンプ
+	該当なし	該当なし	該当なし	該当なし

間隔スタイル

SQL the section called “SET” コマンドを使用すると、間隔値の出力表示形式を変更できます。SQL で間隔データ型を使用する場合は、テキストにキャストして、予想される間隔スタイル (YEAR TO MONTH::text など) を確認します。IntervalStyle 値を設定するために使用できる値は、次のとおりです。

- postgres — PostgreSQL スタイルに従います。これがデフォルトです。
- postgres_verbose — PostgreSQL の詳細スタイルに従います。
- sql_standard — SQL 標準間隔リテラルスタイルに従います。

次のコマンドは、間隔スタイルを sql_standard に設定します。

```
SET IntervalStyle to 'sql_standard';
```

postgres 出力形式

postgres 間隔スタイルの出力形式は、次のとおりです。各数値は負の値にすることができます。

```
'<numeric> <unit> [, <numeric> <unit> ...]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
---------------
1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
```

```
------------------
1 day 02:03:04.5678
```

postgres_verbose 出力形式

postgres_verbose 構文は postgres と似ていますが、postgres_verbose 出力には時間の単位も含まれています。

```
'[@] <numeric> <unit> [, <numeric> <unit> ...] [direction]'
```

```
select INTERVAL '1-2' YEAR TO MONTH::text

varchar
-----------------
@ 1 year 2 mons
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------------------
@ 1 day 2 hours 3 mins 4.56 secs
```

sql_standard 出力形式

間隔値 year to month は次のようにフォーマットされます。間隔の前にマイナス記号を指定すると、間隔が負の値になり、間隔全体に適用されます。

```
'[-]yy-mm'
```

間隔値 day to second は次のようにフォーマットされます。

```
'[-]dd hh:mm:ss.ffffff'
```

```
SELECT INTERVAL '1-2' YEAR TO MONTH::text

varchar
-------
1-2
```

```
select INTERVAL '1 2:3:4.5678' DAY TO SECOND::text

varchar
---------------
1 2:03:04.5678
```

間隔データ型の例

以下の例は、テーブルでの INTERVAL データ型の使用方法を示しています。

```
create table sample_intervals (y2m interval month, h2m interval hour to minute);
insert into sample_intervals values (interval '20' month, interval '2 days
 1:1:1.123456' day to second);
select y2m::text, h2m::text from sample_intervals;



      y2m      |       h2m
---------------+-----------------
 1 year 8 mons | 2 days 01:01:00
```

```
update sample_intervals set y2m = interval '2' year where y2m = interval '1-8' year to
 month;
select * from sample_intervals;

   y2m   |       h2m
---------+-----------------
 2 years | 2 days 01:01:00
```

```
delete from sample_intervals where h2m = interval '2 1:1:0' day to second;
select * from sample_intervals;

 y2m | h2m
-----+-----
```

間隔リテラルの例

以下の例は、間隔スタイルを `postgres` に設定して実行します。

次の例は、1 年の INTERVAL リテラルを作成する方法を示しています。

```
select INTERVAL '1' YEAR

intervaly2m
---------------
1 years 0 mons
```

修飾子を超える quoted-string を指定すると、残りの時間単位が間隔から切り捨てられます。次の例では、13 か月の間隔は 1 年と 1 か月になりますが、YEAR 修飾子により残りの 1 か月が除外されます。

```
select INTERVAL '13 months' YEAR

intervaly2m
---------------
1 years 0 mons
```

間隔文字列を下回る修飾子を使用すると、残った単位が含まれます。

```
select INTERVAL '13 months' MONTH

intervaly2m
---------------
1 years 1 mons
```

間隔で精度を指定すると、小数桁数が指定された精度まで切り捨てられます。

```
select INTERVAL '1.234567' SECOND (3)

intervald2s
--------------------------------
0 days 0 hours 0 mins 1.235 secs
```

精度を指定しない場合は、Amazon Redshift では最大精度 6 が使用されます。

```
select INTERVAL '1.23456789' SECOND

intervald2s
-----------------------------------
0 days 0 hours 0 mins 1.234567 secs
```

次の例では、範囲指定した間隔を作成する方法を示します。

```
select INTERVAL '2:2' MINUTE TO SECOND

intervald2s
------------------------------
0 days 0 hours 2 mins 2.0 secs
```

修飾子は、指定する単位を指定します。例えば、次の例では前の例と同じ quoted-string (2:2) を使用していますが、Amazon Redshift では、修飾子に基づいて異なる時間単位を使用していると認識します。

```
select INTERVAL '2:2' HOUR TO MINUTE

intervald2s
------------------------------
0 days 2 hours 2 mins 0.0 secs
```

各単位の略語と複数形もサポートされています。例えば、5s、5 second、5 seconds は同じ間隔です。サポートされている単位は、年、月、日、時間、分、秒です。

```
select INTERVAL '5s' SECOND

intervald2s
------------------------------
0 days 0 hours 0 mins 5.0 secs
```

```
select INTERVAL '5 HOURS' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

```
select INTERVAL '5 h' HOUR

intervald2s
------------------------------
0 days 5 hours 0 mins 0.0 secs
```

修飾子構文を使用しない間隔リテラルの例

> **Note**
>
> 以下の例は、YEAR TO MONTH または DAY TO SECOND 修飾子を使用しない間隔リテラルを示しています。修飾子を使用した間隔リテラルの推奨例については、「間隔のデータ型とリテラル」を参照してください。

12 hours または 6 months など、特定の期間を識別するには、間隔リテラルを使用します。これらの間隔リテラルは、日時式を必要とする条件および計算の中で使用できます。

間隔リテラルは、INTERVAL キーワードに数量およびサポートされている日付パートを組み合わせて表現します。例えば、INTERVAL '7 days' または INTERVAL '59 minutes' のようになります。複数の数量および単位をつなぎ合わせることで、より正確な間隔を作成できます (例えば、INTERVAL '7 days, 3 hours, 59 minutes')。各単位の省略形および複数形もサポートされています。例えば、5 s、5 second、および 5 seconds は等価な間隔です。

日付部分を指定しない場合、間隔値は秒数を示します。数量値は小数として指定できます (例えば、0.5 days)。

次の例に、さまざまな間隔値を使用した一連の計算を示します。

以下は、指定された日付に 1 秒を追加します。

```
select caldate + interval '1 second' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:00:01
(1 row)
```

以下は、指定された日付に 1 分を追加します。

```
select caldate + interval '1 minute' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 00:01:00
(1 row)
```

以下では、指定された日付に 3 時間と 35 分を追加します。

```
select caldate + interval '3 hours, 35 minutes' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 03:35:00
(1 row)
```

以下は、指定された日付に 52 週を追加します。

```
select caldate + interval '52 weeks' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-12-30 00:00:00
(1 row)
```

以下では、指定された日付に 1 週、1 時間、1 分、および 1 秒を追加します。

```
select caldate + interval '1w, 1h, 1m, 1s' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2009-01-07 01:01:01
(1 row)
```

以下は、指定された日付に 12 時間 (半日) を追加します。

```
select caldate + interval '0.5 days' as dateplus from date
where caldate='12-31-2008';
dateplus
---------------------
2008-12-31 12:00:00
(1 row)
```

以下では、2023 年 2 月 15 日から 4 か月を引いて、結果が 2022 年 10 月 15 日になります。

```
select date '2023-02-15' - interval '4 months';

?column?
```

```
---------------------
2022-10-15 00:00:00
```

以下では、2023 年 3 月 31 日から 4 か月を引いて、結果が 2022 年 11 月 30 日になります。計算では、1 か月の日数を考慮します。

```
select date '2023-03-31' - interval '4 months';

?column?
---------------------
2022-11-30 00:00:00
```

ブール型

シングルバイト列に true 値および false 値を格納するには、BOOLEAN データ型を使用します。次の表に、ブール値の取り得る 3 つの状態と、各状態をもたらすリテラル値について説明します。入力文字列に関係なく、ブール列では、true の場合は「t」を、false の場合は「f」を格納および出力します。

州	有効なリテラル値	ストレージ
True	`TRUE` `'t'` `'true'` `'y'` `'yes'` `'1'`	1 バイト
False	`FALSE` `'f'` `'false'` `'n'` `'no'` `'0'`	1 バイト
不明	`NULL`	1 バイト

IS 比較を使用すると、ブール値を WHERE 句の述語としてのみチェックできます。SELECT リストのブール値に対して IS 比較を使用することはできません。

例

BOOLEAN 列を使用すれば、CUSTOMER テーブル内の顧客ごとに「アクティブ/非アクティブ」状態を格納できます。

```
create table customer(
custid int,
active_flag boolean default true);
```

```
insert into customer values(100, default);
```

```
select * from customer;
custid | active_flag
-------+--------------
   100 | t
```

CREATE TABLE ステートメントにデフォルト値 (`true` または `false`) が指定されていない場合は、デフォルト値を挿入しても Null が挿入されます。

この例では、クエリによって、スポーツは好きだが映画館は好きでないユーザーが USERS テーブルから選択されます。

```
select firstname, lastname, likesports, liketheatre
from users
where likesports is true and liketheatre is false
order by userid limit 10;

firstname |  lastname  | likesports | liketheatre
----------+------------+------------+-------------
Lars      | Ratliff    | t          | f
Mufutau   | Watkins    | t          | f
Scarlett  | Mayer      | t          | f
Shafira   | Glenn      | t          | f
Winifred  | Cherry     | t          | f
Chase     | Lamb       | t          | f
Liberty   | Ellison    | t          | f
Aladdin   | Haney      | t          | f
Tashya    | Michael    | t          | f
Lucian    | Montgomery | t          | f
(10 rows)
```

次の例では、ロックミュージックを好むかどうか不明なユーザーが USERS テーブルから選択されます。

```
select firstname, lastname, likerock
```

```
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows)
```

次の例では、SELECT リストで IS 比較を使用しているため、エラーが返されます。

```
select firstname, lastname, likerock is true as "check"
from users
order by userid limit 10;

[Amazon](500310) Invalid operation: Not implemented
```

次の例は、IS 比較ではなく SELECT リストで等号比較 (=) を使用しているため成功します。

```
select firstname, lastname, likerock = true as "check"
from users
order by userid limit 10;

firstname | lastname  | check
----------+-----------+------
Rafael    | Taylor    |
Vladimir  | Humphrey  |
Lars      | Ratliff   | true
Barry     | Roy       |
Reagan    | Hodge     | true
Victor    | Hernandez | true
Tamekah   | Juarez    |
Colton    | Roy       | false
Mufutau   | Watkins   |
```

```
Naida      | Calderon   |
```

HLLSKETCH タイプ

HyperLogLog スケッチには HLLSKETCH データ型を使用します。Amazon Redshift は、スパースまたはデンスの HyperLogLog スケッチ表現をサポートしています。スケッチはスパースとして開始され、使用するメモリフットプリントを最小化するために、デンス形式がより効率的になるとデンスに切り替わります。

Amazon Redshift は、次の JSON 形式でスケッチをインポート、エクスポート、または印刷するときに、スパースの HyperLogLog スケッチを自動的に移行します。

```
{"logm":15,"sparse":{"indices":[4878,9559,14523],"values":[1,2,1]}}
```

Amazon Redshift は、Base64 形式の文字列表現を使用して、デンスの HyperLogLog スケッチを表します。

Amazon Redshift は、デンスの HyperLogLog スケッチを表現するために、Base64 形式の次の文字列表現を使用します。

```
"ABAABA..."
```

raw 圧縮で使用する場合、HLLSKETCH オブジェクトの最大サイズは 24,580 バイトです。

SUPER タイプ

SUPER データ型を使用して、半構造化データまたはドキュメントを値として保存します。

半構造化データは、SQL データベースで使用されるリレーショナルデータモデルの厳密な表形式の構造に準拠していません。これには、データ内の個別のエンティティを参照するタグが含まれます。配列やネストされた構造体、JSON などのシリアル化形式に関連付けられているその他の複雑な構造体などの複雑な値を含めることができます。SUPER データ型は、Amazon Redshift の他のスカラー型をすべて含む一連のスキーマレス配列と構造体の値です。

SUPER データ型は、SUPER オブジェクトごとに最大 16 MB のデータをサポートします。SUPER データ型の詳細 (テーブルへの実装例を含む) については、「Amazon Redshift の半構造化データ」を参照してください。

1 MB を超える SUPER オブジェクトは、以下のファイル形式からのみ取り込むことができます。

- Parquet
- JSON
- TEXT
- CSV

SUPER データ型には、以下のプロパティがあります。

- Amazon Redshift のスカラー値:
  - null
  - ブール型
  - smallint、integer、bigint、decimal、または浮動小数点 (float4 や float8) などの数値
  - varchar や char などの文字列値
- 複雑な値:
  - スカラーまたは複素数を含む値の配列
  - タプルまたはオブジェクトとも呼ばれる構造体。属性名と値 (スカラーまたは複合体) のマップです。

2 種類の複素数値のいずれにも、規則性の制限なしに、独自のスカラーまたは複素数値が含まれています。

SUPER データ型は、スキーマレス形式の半構造化データの永続性をサポートします。階層データモデルは変更できますが、データの古いバージョンは同じ SUPER 列に共存することができます。

Amazon Redshift は PartiQL を使用して配列と構造体へのナビゲーションを有効にします。また、Amazon Redshift は PartiQL 構文を使用しながら SUPER 配列を反復処理します。詳細については、「ナビゲーション」と「ネストされていないクエリ」を参照してください。

Amazon Redshift では、動的型付けを使用して、クエリで使用する前にデータ型を宣言することなく、スキーマレスの SUPER データを処理します。詳細については、「動的型付け」を参照してください。

SUPER 型の列のパス上の `scalar` 値に動的データマスキングポリシーを適用できます。動的データマスキングの詳細については、「動的データマスキング」を参照してください。動的データマスキングを SUPER データ型と使用する方法の詳細については、「SUPER データタイプパスでの動的データマスキングの使用」を参照してください。

VARBYTE 型

一定の制限を持つ可変長のバイナリ値を格納するには、VARBYTE 列、VARBINARY 列または BINARY VARYING 列を使用します。

```
varbyte [ (n) ]
```

最大バイト数 (n) の範囲は 1 ~ 16,777,216 です。デフォルト値は 64,000 です。

以下に、VARBYTE データ型を使用する場合の例をいくつか示します。

- VARBYTE 列でのテーブルの結合。
- VARBYTE 列を含むマテリアライズドビューの作成。VARBYTE 列を含むマテリアライズド･ビューでは、増分更新がサポートされます。ただし、VARBYTE 列の COUNT、MIN、MAX および GROUP BY 以外の集計関数は、増分更新をサポートしません。

すべてのバイトが表示可能な文字であることを保証するために、Amazon Redshift は 16 進数形式を使用して VARBYTE 値を出力します。例えば、次の SQL では 16 進数の文字列 6162 をバイナリ値に変換しています。戻り値がバイナリ値であっても、結果は 16 進数の 6162 で出力されます。

```
select from_hex('6162');

 from_hex
----------
 6162
```

Amazon Redshift では、VARBYTE と以下に示すデータ型間でのキャストがサポートされています。

- CHAR
- VARCHAR
- SMALLINT
- INTEGER
- BIGINT

CHAR と VARCHAR をキャストする場合は、UTF-8 形式が使用されます。UTF-8 形式の詳細については、「TO_VARBYTE」を参照してください。SMALLINT、INTEGER、BIGINT からキャストする

場合、元のデータ型のバイト数は維持されます。つまり、SMALLINT の場合は 2 バイト、INTEGER の場合は 4 バイト、BIGINT の場合は 8 バイトとなります。

次の SQL ステートメントは、VARCHAR 文字列を VARBYTE にキャストします。戻り値がバイナリ値であっても、結果は 16 進数の 616263 で出力されます。

```
select 'abc'::varbyte;

 varbyte
---------
 616263
```

次の SQL ステートメントは、列内の CHAR 値を VARBYTE にキャストします。次の使用例では、CHAR (10) 列 (c) を持つテーブルを作成し、長さが 10 より短い文字値を挿入します。出力されるキャストでは、スペース文字 (hex'20') を使用して、定義された列サイズに結果がパディングされます。戻り値がバイナリ値の場合でも、結果は 16 進数で表示されます。

```
create table t (c char(10));
insert into t values ('aa'), ('abc');
select c::varbyte from t;
          c
----------------------
 6161202020202020202020
 6162632020202020202020
```

次の SQL ステートメントは、SMALLINT 文字列を VARBYTE にキャストします。戻り値がバイナリ値の場合でも、この結果は 16 進数 0005 (2 バイトつまり 4 桁の 16 進数文字列) として表示されます。

```
select 5::smallint::varbyte;

 varbyte
---------
 0005
```

次の SQL ステートメントは、INTEGER を VARBYTE にキャストします。戻り値がバイナリ値の場合でも、この結果は 16 進数 00000005 (4 バイトつまり 8 桁の16進数文字列) として出力されます。

```
select 5::int::varbyte;

 varbyte
----------
 00000005
```

次の SQL ステートメントは、BIGINT を VARBYTE にキャストします。戻り値がバイナリ値の場合でも、この結果は 16 進数 `0000000000000005` (8 バイトつまり 16 桁の 16 進数文字列) として出力されます。

```
select 5::bigint::varbyte;

     varbyte
------------------
 0000000000000005
```

VARBYTE データ型をサポートする Amazon Redshift の各機能は以下のとおりです。

- VARBYTE 演算子
- CONCAT
- LEN
- LENGTH 関数
- OCTET_LENGTH
- SUBSTRING 関数
- FROM_HEX
- TO_HEX
- FROM_VARBYTE
- TO_VARBYTE
- GETBIT
- VARBYTE データ型の列のロード
- VARBYTE データ型の列のアップロード

Amazon Redshift で VARBYTE データ型を使用する際の制約事項

Amazon Redshift で VARBYTE データを使用する際の制約事項を以下に示します。

- Amazon Redshift Spectrum は、Parquet ファイルと ORC ファイルに対してのみ VARBYTE データ型をサポートします。
- Amazon Redshift のクエリエディタと Amazon Redshift クエリエディタ v2 は、現段階で VARBYTE データ型を完全にはサポートしていません。したがって、VARBYTE 表現を処理する際には、他の SQL クライアントを使用してください。

  クエリエディタを使用する際の回避策として、データの長さが 64 KB 未満で、かつコンテンツが有効な UTF-8 で構成されている場合は、VARBYTE 値を VARCHAR にキャストできます。次にこの例を示します。

  ```
  select to_varbyte('6162', 'hex')::varchar;
  ```
- Python または Lambda ユーザー定義関数 (UDF) では VARBYTE データ型を使用することはできません。
- VARBYTE 列から HLLSKETCH 列を作成したり、VARBYTE 列で APPROXIMATE COUNT DISTINCT を使用したりすることはできません。
- 1 MB を超える VARBYTE 値は、以下のファイル形式からのみ取り込むことができます。
  - Parquet
  - テキスト
  - カンマ区切り値 (CSV)

型の互換性と変換

Amazon Redshift における型変換ルールおよびデータ型の互換性についての説明を以下に示します。

互換性

データ型のマッチング、リテラル値および定数のデータ型とのマッチングは、以下のようなさまざまなデータベース操作で発生します。

- テーブルにおけるデータ操作言語 (DML) オペレーション
- UNION、INTERSECT、および EXCEPT のクエリ
- CASE 式
- LIKE や IN など、述語の評価
- データの比較または抽出を行う SQL 関数の評価
- 算術演算子との比較

これらの操作の結果は、型変換ルールおよびデータ型の互換性に左右されます。互換性は、特定の値と特定のデータ型との 1 対 1 のマッチングが必ずしも必要でないことを暗示しています。一部のデータ型は互換性があるため、暗黙変換、または強制が可能です (詳細については、「暗黙的な変換型」を参照)。データ型に互換性がない場合は、明示変換関数を使用することにより、値をあるデータ型から別のデータ型に変換することが可能な場合があります。

互換性と変換に関する全般的なルール

次に示す互換性と変換に関するルールに注意してください。

- 一般に、同じデータ型のカテゴリに属するデータ型 (各種の数値データ型) は互換性があり、暗黙的に変換することができます。

  例えば、暗黙的な変換では、10 進値を整数列に変換できます。10 進値は整数に四捨五入されます。または、`2008` のような数値を日付から抽出し、その値を整数列に挿入することができます。
- 数値データ型では、範囲外の値を挿入しようとしたときに発生するオーバーフロー条件を適用します。例えば、精度が 5 桁の 10 進値は、4 桁の精度で定義された 10 進列に適合しません。整数または 10 進値の整数部は決して切り捨てられません。しかし、10 進値の小数部は、適宜、切り上げまたは切り捨てることができます。ただし、テーブルから選択された値の明示的なキャストの結果は丸められません。
- 各種のキャラクタ文字列型には互換性があります。シングルバイトデータを含む VARCHAR 列の文字列と CHAR 列の文字列は互換性があり、暗黙的に変換することができます。マルチバイトデータを含む VARCHAR 文字列には互換性がありません。また、キャラクタ文字列については、文字列が適切なリテラル値であれば、日付、時間、タイムスタンプ、または数値に変換することができます。先頭のスペースまたは末尾のスペースはいずれも無視されます。逆に、日付、時間、タイムスタンプ、または数値は、固定長または可変長の文字列に変換することができます。

  > **Note**
  >
  > 数値型にキャストするキャラクタ文字列には、数字の文字表現が含まれている必要があります。例えば、文字列 `'1.0'` または `'5.9'` を 10 進値にキャストすることはできますが、文字列 `'ABC'` はいずれの数値型にもキャストできません。
- DECIMAL 値を文字列と比較すると、Amazon Redshift は文字列を DECIMAL 値に変換しようとします。他のすべての数値と文字列を比較する場合、数値は文字列に変換されます。反対方向の変換 (文字列を整数に変換する、DECIMAL 値を文字列に変換するなど) を実行するには、CAST などの明示的な関数を使用します。

- 64 ビットの DECIMAL または NUMERIC の値を上位の精度に変換するには、CAST や CONVERT などの明示的な変換関数を使用する必要があります。
- DATE または TIMESTAMP を TIMESTAMPTZ に変換する場合、または TIME を TIMETZ に変換する場合、タイムゾーンは現在のセッションのタイムゾーンに設定されます。セッションのタイムゾーンは、デフォルト値の UTC です。セッションのタイムゾーンを設定する方法の詳細については、「timezone」を参照してください。
- 同様に、TIMESTAMPTZ は、現在のセッションのタイムゾーンに基づいて DATE、TIME または TIMESTAMP に変換されます。セッションのタイムゾーンは、デフォルト値の UTC です。変換後、タイムゾーン情報は削除されます。
- 指定されたタイムゾーンを含むタイムスタンプを表す文字列は、現在のセッションタイムゾーン (デフォルトでは UTC) を使用して TIMESTAMPTZ に変換されます。同様に、タイムゾーンが指定されている時刻を表す文字列は、現在のセッションのタイムゾーン (デフォルトでは UTC) を使用して TIMETZ に変換されます。

暗黙的な変換型

暗黙的な変換には、2 つのタイプがあります。

- 割り当てにおける暗黙的な変換 (INSERT コマンドまたは UPDATE コマンドで値を設定するなど)
- 式における暗黙的な変換 (WHERE 句で比較を実行するなど)

次の表に、割り当てまたは式において暗黙的に変換できるデータ型を一覧表示します。これらの変換は、明示的な変換関数を使用して実行することもできます。

変換元の型	変換先の型
BIGINT (INT8)	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT、INT4)
	REAL (FLOAT4)

変換元の型	変換先の型
	SMALLINT (INT2)
	VARCHAR
CHAR	VARCHAR
DATE	CHAR
	VARCHAR
	TIMESTAMP
	TIMESTAMPTZ
DECIMAL (NUMERIC)	BIGINT (INT8)
	CHAR
	DOUBLE PRECISION (FLOAT8)
	INTEGER (INT、INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
DOUBLE PRECISION (FLOAT8)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT、INT4)
	REAL (FLOAT4)
	SMALLINT (INT2)

変換元の型	変換先の型
	VARCHAR
INTEGER (INT、INT4)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)
	REAL (FLOAT4)
	SMALLINT (INT2)
	VARCHAR
REAL (FLOAT4)	BIGINT (INT8)
	CHAR
	DECIMAL (NUMERIC)
	INTEGER (INT、INT4)
	SMALLINT (INT2)
	VARCHAR
SMALLINT (INT2)	BIGINT (INT8)
	BOOLEAN
	CHAR
	DECIMAL (NUMERIC)
	DOUBLE PRECISION (FLOAT8)

変換元の型	変換先の型
	INTEGER (INT、INT4)
	REAL (FLOAT4)
	VARCHAR
TIMESTAMP	CHAR
	DATE
	VARCHAR
	TIMESTAMPTZ
	TIME
TIMESTAMPTZ	CHAR
	DATE
	VARCHAR
	TIMESTAMP
	TIMETZ
TIME	VARCHAR
	TIMETZ
	INTERVAL DAY TO SECOND
TIMETZ	VARCHAR
	TIME
GEOMETRY	GEOGRAPHY
GEOGRAPHY	GEOMETRY

> **Note**
>
> TIMESTAMPTZ、TIMESTAMP、DATE、TIME、TIMETZ、またはキャラクタ文字列間の暗黙的な変換では、現在のセッションのタイムゾーンが使用されます。現在のタイムゾーンの設定については、「timezone」を参照してください。
> GEOMETRY および GEOGRAPHY データ型は、互いに変換する場合を除き、暗黙的に他のデータ型に変換することはできません。詳細については、「CAST 関数」を参照してください。
> VARBYTE データ型は、暗黙的に他のデータ型に変換できません。詳細については、「CAST 関数」を参照してください。

SUPER データ型での動的型付けの使用

Amazon Redshift では、動的型付けを使用して、クエリで使用する前にデータ型を宣言することなく、スキーマレスの SUPER データを処理します。動的型付けでは、明示的に Amazon Redshift 型にキャストすることなく、SUPER データ列に移動した結果が使用されます。SUPER データ型に対する動的型付けを使用する方法の詳細については、動的型付け を参照してください。

一部の例外を除いて、他のデータ型との間で SUPER 値をキャストすることができます。詳細については、「制約事項」を参照してください。

照合順序

Amazon Redshift では、ロケール固有の照合順序またはユーザー定義の照合順序をサポートしていません。一般に、コンテキストでの述語の結果は、データ値のソートおよび比較に関するロケール固有のルールがないことに影響を受ける可能性があります。例えば、ORDER BY 式と、MIN、MAX、および RANK などの関数は、ロケール固有の文字を考慮に入れないデータのバイナリ UTF8 順序付けに基づいて結果を返します。

式

トピック

- 単純式
- 複合式
- 式リスト
- スカラーサブクエリ
- 関数式

式は、1 つまたは複数の値、演算子、または関数 (評価して値を返す) の組み合わせです。式のデータ型は、一般にそのコンポーネントのデータ型と同じです。

単純式

単純式は以下のいずれかとなります。

- 定数またはリテラル値
- 列名または列参照
- スカラー関数
- 集計 (set) 関数
- ウィンドウ関数
- スカラーサブクエリ

単純式には次のようなものがあります。

```
5+12
dateid
sales.qtysold * 100
sqrt (4)
max (qtysold)
(select max (qtysold) from sales)
```

複合式

複合式は、一連の単純式が算術演算子によって結合されたものです。複合式内で使用される単純式は、数値を返す必要があります。

構文

```
expression
operator
expression | (compound_expression)
```

引数

expression

評価して値を返す単純式。

operator

複合演算式は、以下の演算子 (優先順位は列記順) を使用して作成することができます。

- ( ) : 評価の順番を制御する括弧
- + , - : 正および負の符号/演算子
- ^ , |/ , ||/ : 指数、平方根、立方根
- * , / , % : 乗算演算子、除算演算子、モジュロ演算子
- @ : 絶対値
- + , - : 加算および減算
- & , |, #, ~, <<, >> : AND、OR、NOT、左シフト、右シフトのビット演算子
- ||: 連結

(compound_expression)

複合式は括弧を使用して入れ子にすることができます。

例

複合式の例を以下に示します。

```
('SMITH' || 'JONES')
sum(x) / y
sqrt(256) * avg(column)
rank() over (order by qtysold) / 100
(select (pricepaid - commission) from sales where dateid = 1882) * (qtysold)
```

また、一部の関数は、他の関数内に入れ子にすることができます。例えば、任意のスカラー関数を別のスカラー関数内に入れ子にすることができます。次の例では、一連の数の絶対値の合計を返します。

```
sum(abs(qtysold))
```

ウィンドウ関数は、集計関数または他のウィンドウ関数の引数として使用できません。次の式は、エラーを返すことになります。

```
avg(rank() over (order by qtysold))
```

ウィンドウ関数には入れ子にした集計関数を含めることができます。次の式は、値セットの合計を出し、ランク付けします。

```
rank() over (order by sum(qtysold))
```

式リスト

式リストは、式の組み合わせであり、メンバーシップおよび比較条件 (WHERE 句) 内、および GROUP BY 句内に指定できます。

構文

```
expression , expression , ... | (expression, expression, ...)
```

引数

expression

評価して値を返す単純式。式リストには、1 つまたは複数のカンマ区切り式、または 1 つまたは複数のカンマ区切り式セットを含めることができます。複数の式セットがある場合、各セットはそれぞれ同じ個数の式を含み、括弧で区切る必要があります。各セット内の式の個数は、条件内の演算子の前にある式の個数と一致する必要があります。

例

条件内の式リストの例を次に示します。

```
(1, 5, 10)
('THESE', 'ARE', 'STRINGS')
(('one', 'two', 'three'), ('blue', 'yellow', 'green'))
```

各セット内の式の個数は、ステートメントの最初の部分にある式の個数と一致する必要があります。

```
select * from venue
where (venuecity, venuestate) in (('Miami', 'FL'), ('Tampa', 'FL'))
order by venueid;

venueid |        venuename        | venuecity | venuestate | venueseats
---------+-------------------------+-----------+------------+------------
```

```
28 | American Airlines Arena | Miami     | FL         |          0
54 | St. Pete Times Forum    | Tampa     | FL         |          0
91 | Raymond James Stadium   | Tampa     | FL         |      65647
(3 rows)
```

スカラーサブクエリ

スカラーサブクエリとは、ちょうど 1 つの値 (1 つの列を含む 1 つの行) を返す、括弧で囲まれた通常の SELECT クエリです。実行されたこのクエリが返す値は、外部のクエリで使用されます。サブクエリが 0 行ゼロを返した場合、サブクエリ式の値は Null になります。サブクエリが複数の行を返した場合、Amazon Redshift はエラーを返します。サブクエリは親クエリからの変数を参照することができます。この変数はサブクエリの起動時中には定数として働きます。

式を呼び出すほとんどのステートメントでスカラーサブクエリを使用できます。次のような場合、スカラーサブクエリは有効な式でなくなります。

- 式のデフォルト値として使用
- GROUP BY 句および HAVING 句内に使用

例

次のサブクエリは 2008 年の 1 年を通して販売 1 回あたりに支払われた平均料金を計算します。次に、外側のクエリが出力にその値を使用して、四半期ごとの販売 1 回あたりの平均料金と比較します。

```
select qtr, avg(pricepaid) as avg_saleprice_per_qtr,
(select avg(pricepaid)
from sales join date on sales.dateid=date.dateid
where year = 2008) as avg_saleprice_yearly
from sales join date on sales.dateid=date.dateid
where year = 2008
group by qtr
order by qtr;
qtr  | avg_saleprice_per_qtr | avg_saleprice_yearly
-------+-----------------------+----------------------
1     |                647.64 |              642.28
2     |                646.86 |              642.28
3     |                636.79 |              642.28
4     |                638.26 |              642.28
(4 rows)
```

関数式

構文

組み込み関数は式として使用できます。関数呼び出しの構文は、関数名の後に、その引数リストを括弧に囲んで配置する形式をとります。

```
function ( [expression [, expression...]] )
```

引数

function

組み込み関数。サンプルの関数については、「SQL 関数リファレンス」を参照してください。

expression

関数によって予期されるデータ型およびパラメータの個数と一致する式。

例

```
abs (variable)
select avg (qtysold + 3) from sales;
select dateadd (day,30,caldate) as plus30days from date;
```

条件

トピック

- 構文
- 比較条件
- 論理条件
- パターンマッチング条件
- BETWEEN 範囲条件
- Null 条件
- EXISTS 条件
- IN 条件

条件は、評価結果として true、false、または unknown を返す、1 つまたは複数の式および論理演算子から成るステートメントです。条件は述語と呼ばれることもあります。

> **Note**
> すべての文字列比較および LIKE パターンマッチングでは、大文字と小文字が区別されます。例えば、「A」と「a」は一致しません。ただし、ILIKE 述語を使用すれば、大文字小文字を区別しないパターンマッチングを行うことができます。

構文

```
comparison_condition
| logical_condition
| range_condition
| pattern_matching_condition
| null_condition
| EXISTS_condition
| IN_condition
```

比較条件

比較条件では、2 つの値の間の論理的な関係を指定します。比較条件はすべて、ブール型の戻り値を返す 2 項演算子です。Amazon Redshift では、次のテーブルで説明する比較演算子をサポートしています。

演算子	構文	説明
<	`a < b`	値 a は値 b より小さい。
>	`a > b`	値 a は値 b より大きい。
<=	`a <= b`	値 a は値 b 以下。
>=	`a >= b`	値 a は値 b 以上。
=	`a = b`	値 a は値 b と等しい。
<> または !=	`a <> b or a != b`	値 a は値 b と等しくない。

演算子	構文	説明
ANY | SOME	`a = ANY(subquery)`	値 a はサブクエリによって返されるいずれかの値と等しい。
ALL	`a <> ALL or != ALL (subquery))`	値 a はサブクエリによって返されるどの値とも等しくない。
IS TRUE | FALSE | UNKNOWN	`a IS TRUE`	値は a はブール数で TRUE。

使用に関する注意事項

= ANY | SOME

ANY と SOME のキーワードは IN 条件と同義であり、1 つまたは複数の値を返すサブクエリによって返された少なくとも 1 つの値に対して比較が true である場合に true を返します。Amazon Redshift は、ANY および SOME に対して = (等しい) 条件のみをサポートします。不等条件はサポートされていません。

> **Note**
>
> ALL 述語はサポートされていません。

<> ALL

ALL キーワードは NOT IN (IN 条件 条件を参照) と同義であり、サブクエリの結果に式が含まれていない場合に true を返します。Amazon Redshift は、ALL に対して <> または != (等しくない) 条件のみをサポートします。他の比較条件はサポートされていません。

IS TRUE/FALSE/UNKNOWN

ゼロ以外の値は TRUE と等しく、0 は FALSE に等しく、Null は UNKNOWN に等しくなります。「ブール型」データ型を参照してください。

例

ここで、比較条件の簡単な例をいくつか示します。

```
a = 5
a < b
min(x) >= 5
qtysold = any (select qtysold from sales where dateid = 1882
```

次のクエリは、VENUE テーブルから席数が 10000 席を超える会場を返します。

```
select venueid, venuename, venueseats from venue
where venueseats > 10000
order by venueseats desc;

venueid |           venuename            | venueseats
---------+--------------------------------+------------
83 | FedExField                     |      91704
 6 | New York Giants Stadium        |      80242
79 | Arrowhead Stadium              |      79451
78 | INVESCO Field                  |      76125
69 | Dolphin Stadium                |      74916
67 | Ralph Wilson Stadium           |      73967
76 | Jacksonville Municipal Stadium |      73800
89 | Bank of America Stadium        |      73298
72 | Cleveland Browns Stadium       |      73200
86 | Lambeau Field                  |      72922
...
(57 rows)
```

この例では、USERS テーブルからロックミュージックが好きなユーザー (USERID) を選択します。

```
select userid from users where likerock = 't' order by 1 limit 5;

userid
--------
3
5
6
13
16
(5 rows)
```

この例では、USERS テーブルから、ロックミュージックを好きかどうか不明であるユーザー (USERID) を選択します。

```
select firstname, lastname, likerock
from users
where likerock is unknown
order by userid limit 10;

firstname | lastname | likerock
----------+----------+----------
Rafael    | Taylor   |
Vladimir  | Humphrey |
Barry     | Roy      |
Tamekah   | Juarez   |
Mufutau   | Watkins  |
Naida     | Calderon |
Anika     | Huff     |
Bruce     | Beck     |
Mallory   | Farrell  |
Scarlett  | Mayer    |
(10 rows
```

TIME 列の例

次のテーブルの TIME_TEST の例には、3 つの値が挿入された列 TIME_VAL (タイプ TIME) があります。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

次の例では、各 timetz_val から時間を抽出します。

```
select time_val from time_test where time_val < '3:00';
   time_val
---------------
 00:00:00.5550
 00:58:00
```

次の例では、2 つの時刻リテラルを比較します。

```
select time '18:25:33.123456' = time '18:25:33.123456';
 ?column?
----------
 t
```

TIMETZ 列の例

次のテーブルの TIMETZ_TEST の例には、3 つの値が挿入された列 TIMETZ_VAL (タイプ TIMETZ) があります。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

次の例では、`3:00:00 UTC` 未満の TIMETZ 値のみを選択します。値を UTC に変換した後に比較が行われます。

```
select timetz_val from timetz_test where timetz_val < '3:00:00 UTC';

   timetz_val
---------------
 00:00:00.5550+00
```

次の例では、2 つの TIMETZ リテラルを比較します。タイムゾーンは、比較のために無視されます。

```
select time '18:25:33.123456 PST' < time '19:25:33.123456 EST';

 ?column?
----------
 t
```

論理条件

論理条件は、2 つの条件の結果を結合して 1 つの結果を作成します。論理条件はすべて、ブール型の戻り値を返す 2 項演算子です。

構文

```
expression
{ AND | OR }
expression
NOT expression
```

論理条件では 3 値ブール論理を使用します。この場合、Null 値は unknown 関係を表現します。次の表で論理条件の結果について説明します。ここで、E1 と E2 は式を表します。

E1	E2	E1 AND E2	E1 OR E2	NOT E2
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

NOT 演算子は AND 演算子より先に評価され、AND 演算子は OR 演算子より先に評価されます。括弧が使用されている場合、評価のデフォルトの順序より優先されます。

例

次の例では、USERS テーブルから、ラスベガスもスポーツも好きであるユーザーの USERID および USERNAME を返します。

```
select userid, username from users
where likevegas = 1 and likesports = 1
```

```
order by userid;

userid | username
--------+----------
1 | JSG99FHE
67 | TWU10MZT
87 | DUF19VXU
92 | HYP36WEQ
109 | FPL38HZK
120 | DMJ24GUZ
123 | QZR22XGQ
130 | ZQC82ALK
133 | LBN45WCH
144 | UCX04JKN
165 | TEY68OEB
169 | AYQ83HGO
184 | TVX65AZX
...
(2128 rows)
```

次の例では、USERS テーブルから、ラスベガスが好き、スポーツが好き、または両方が好きのいずれかに該当するユーザーの USERID と USERNAME を返します。このクエリでは、前の例の出力と、ラスベガスのみ好き、またはスポーツのみ好きなユーザーとをすべて返します。

```
select userid, username from users
where likevegas = 1 or likesports = 1
order by userid;

userid | username
--------+----------
1 | JSG99FHE
2 | PGL08LJI
3 | IFT66TXU
5 | AEB55QTM
6 | NDQ15VBM
9 | MSD36KVR
10 | WKW41AIW
13 | QTF33MCG
15 | OWU78MTR
16 | ZMG93CDD
22 | RHT62AGI
27 | KOY02CVE
29 | HUH27PKK
```

```
...
(18968 rows)
```

次のクエリでは、OR 条件を囲む括弧を使用して、マクベスが上演されたニューヨークあるいはカリフォルニアの劇場を探します。

```
select distinct venuename, venuecity
from venue join event on venue.venueid=event.venueid
where (venuestate = 'NY' or venuestate = 'CA') and eventname='Macbeth'
order by 2,1;

venuename                |   venuecity
----------------------------------------+---------------
Geffen Playhouse                        | Los Angeles
Greek Theatre                           | Los Angeles
Royce Hall                              | Los Angeles
American Airlines Theatre               | New York City
August Wilson Theatre                   | New York City
Belasco Theatre                         | New York City
Bernard B. Jacobs Theatre               | New York City
...
```

この例の括弧を削除すると、クエリの論理および結果が変更されます。

次の例では、NOT 演算子を使用します。

```
select * from category
where not catid=1
order by 1;

catid | catgroup |  catname  |                  catdesc
-------+----------+-----------+------------------------------------------
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
...
```

次の例では、NOT 条件の後に AND 条件を使用しています。

```
select * from category
where (not catid=1) and catgroup='Sports'
```

```
order by catid;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
2 | Sports   | NHL     | National Hockey League
3 | Sports   | NFL     | National Football League
4 | Sports   | NBA     | National Basketball Association
5 | Sports   | MLS     | Major League Soccer
(4 rows)
```

パターンマッチング条件

トピック

- LIKE
- SIMILAR TO
- POSIX 演算子

パターンマッチング演算子は、条件式で指定されたパターンが存在するかどうか文字列を調べ、該当するパターンが見つかったかどうかに応じて true または false を返します。Amazon Redshift は、パターン一致に 3 つの方法を使用します。

- LIKE 式

  LIKE 演算子は、列名などの文字列式を、ワイルドカード文字 % (パーセント) および _ (アンダースコア) を使用したパターンと比較します。LIKE パターンマッチングは常に文字列全体を網羅します。LIKE は大文字小文字を区別するマッチングを実行し、ILIKE は大文字小文字を区別しないマッチングを実行します。
- SIMILAR TO 正規表現

  SIMILAR TO 演算子は、文字列式を、SQL の標準的な正規表現パターンと突き合わせます。このパターンには、LIKE 演算子でサポートされている 2 つを含む一連のパターンマッチングメタ文字を含めることができます。SIMILAR TO は、文字列全体をマッチングし、大文字小文字を区別するマッチングを実行します。
- POSIX スタイルの正規表現

  POSIX 正規表現は、LIKE および SIMILAR TO の演算子の場合よりも強力なパターンマッチング手段を提供します。POSIX 正規表現パターンでは、文字列の任意の部分をマッチングすることができ、大文字小文字を区別するマッチングを実行します。

SIMILAR TO または POSIX の演算子を使用する正規表現マッチングは、計算コストが高くなります。非常に多くの行を処理する場合は特に、可能な限り、LIKE を使用することをお勧めします。例えば、以下に示す各クエリは機能的には同じですが、LIKE を使用したクエリは、正規表現を使用したクエリよりも数倍速く実行できます。

```
select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die%';
```

LIKE

LIKE 演算子は、列名などの文字列式を、ワイルドカード文字 % (パーセント) および _ (アンダースコア) を使用したパターンと比較します。LIKE パターンマッチングは常に文字列全体を網羅します。文字列内の任意の場所にあるシーケンスをマッチングするには、パターンがパーセント符号で始まりパーセント符号で終了する必要があります。

LIKE は大文字小文字を区別し、ILIKE は大文字小文字を区別しません。

構文

```
expression [ NOT ] LIKE | ILIKE pattern [ ESCAPE 'escape_char' ]
```

引数

expression

列名など、有効な UTF-8 文字式。

LIKE | ILIKE

LIKE は大文字小文字を区別するパターンマッチングを実行します。ILIKE は、シングルバイト UTF-8 (ASCII) 文字に対して大文字小文字を区別しないパターンマッチングを実行します。マルチバイト文字に対して大文字と小文字を区別しないパターンの一致を実行するには、LIKE 条件の pattern と pattern で LOWER 関数を使用します。

= や <> などの比較述語とは対照的に、述語 LIKE と ILIKE は、末尾の空白を暗黙的に無視しません。末尾のスペースを無視するには、RTRIM を使用するか、または CHAR 列を VARCHAR に明示的にキャストします。

~~ 演算子は LIKE に相当し、~~* は ILIKE に相当します。また、!~~ および !~~* 演算子は、NOT LIKE および NOT ILIKE に相当します。

pattern

マッチングするパターンが含まれる有効な UTF-8 文字式。

escape_char

パターン内でメタ文字をエスケープする文字式。デフォルトは 2 個のバックスラッシュ (「\\」) です。

pattern にメタ文字が含まれていない場合、pattern は文字列そのものを表現するにすぎません。その場合、LIKE は等号演算子と同じ働きをします。

どちらの文字式も CHAR または VARCHAR のデータ型になることができます。文字式の型が異なる場合、Amazon Redshift は pattern のデータ型を expression のデータ型に変換します。

LIKE では、次のパターンマッチングメタ文字をサポートしています。

演算子	説明
%	ゼロ個以上の任意の文字シーケンスをマッチングします。
_	任意の 1 文字をマッチングします。

例

次の例では、LIKE を使用したパターンマッチングの例を示します。

式	戻り値
`'abc' LIKE 'abc'`	True
`'abc' LIKE 'a%'`	True
`'abc' LIKE '_B_'`	False
`'abc' ILIKE '_B_'`	True
`'abc' LIKE 'c%'`	False

次の例では、名前が「E」で始まるすべての市を見つけます。

```
select distinct city from users
where city like 'E%' order by city;
city
---------------
East Hartford
East Lansing
East Rutherford
East St. Louis
Easthampton
Easton
Eatontown
Eau Claire
...
```

次の例では、姓に「ten」が含まれるユーザーを見つけます。

```
select distinct lastname from users
where lastname like '%ten%' order by lastname;
lastname
-------------
Christensen
Wooten
...
```

次の例は、複数のパターンを一致させる方法を示しています。

```
select distinct lastname from tickit.users
where lastname like 'Chris%' or lastname like '%Wooten' order by lastname;
lastname
-------------
Christensen
Christian
Wooten
...
```

次の例では、3 番目と 4 番目の文字が「ea」になっている市を見つけます。コマンドでは ILIKE を使用して、大文字小文字の区別なしを実演します。

```
select distinct city from users where city ilike '__EA%' order by city;
city
-------------
```

```
Brea
Clearwater
Great Falls
Ocean City
Olean
Wheaton
(6 rows)
```

次の例では、デフォルトのエスケープ文字列 (\\) を使用して「start_」を含む文字列を検索します (テキスト `start`、それに続いてアンダースコア _)。

```
select tablename, "column" from pg_table_def
where "column" like '%start\\_%'
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

次の例では、エスケープ文字として '^' を指定し、そのエスケープ文字を使用して「start_」を含む文字列を検索します (テキスト `start`、それに続いてアンダースコア _)。

```
select tablename, "column" from pg_table_def
where "column" like '%start^_%' escape '^'
limit 5;

     tablename     |    column
-------------------+---------------
 stl_s3client      | start_time
 stl_tr_conflict   | xact_start_ts
 stl_undone        | undo_start_ts
 stl_unload_log    | start_time
 stl_vacuum_detail | start_row
(5 rows)
```

次の例では、~~* 演算子を使用して「Ag」で始まる都市の大文字と小文字を区別しない (ILIKE) 検索を行います。

```
select distinct city from users where city ~~* 'Ag%' order by city;

city
------------
Agat
Agawam
Agoura Hills
Aguadilla
```

SIMILAR TO

SIMILAR TO 演算子は、列名などの文字列式を、SQL の標準的な正規表現パターンと突き合わせます。SQL の正規表現パターンには、LIKE 演算子によってサポートされる 2 つを含む、一連のパターンマッチングメタ文字を含めることができます。

SIMILAR TO 演算子の場合は、POSIX の正規表現の動作 (パターンは文字列の任意の部分と一致できる) とは異なり、パターンが文字全体と一致した場合にのみ true を返します。

SIMILAR TO は大文字小文字を区別するマッチングを実行します。

> **Note**
>
> SIMILAR TO を使用する正規表現マッチングは、計算コストが高くなります。非常に多くの行を処理する場合は特に、可能な限り、LIKE を使用することをお勧めします。例えば、以下に示す各クエリは機能的には同じですが、LIKE を使用したクエリは、正規表現を使用したクエリよりも数倍速く実行できます。
>
> ```
> select count(*) from event where eventname SIMILAR TO '%(Ring|Die)%';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

構文

```
expression [ NOT ] SIMILAR TO pattern [ ESCAPE 'escape_char' ]
```

引数

expression

列名など、有効な UTF-8 文字式。

SIMILAR TO

SIMILAR TO は、expression 内の文字列全体について、大文字小文字を区別するパターンマッチングを実行します。

pattern

SQL の標準的な正規表現パターンを表現する有効な UTF-8 文字式。

escape_char

パターン内でメタ文字をエスケープする文字式。デフォルトは 2 個のバックスラッシュ (「\\」) です。

pattern にメタ文字が含まれていない場合、pattern は文字列そのものを表現するにすぎません。

どちらの文字式も CHAR または VARCHAR のデータ型になることができます。文字式の型が異なる場合、Amazon Redshift は pattern のデータ型を expression のデータ型に変換します。

SIMILAR TO では、次のパターンマッチングメタ文字をサポートしています。

演算子	説明
%	ゼロ個以上の任意の文字シーケンスをマッチングします。
_	任意の 1 文字をマッチングします。
|	交替を示します (2 つの選択肢のいずれか)。
*	前の項目をゼロ回以上繰り返します。
+	前の項目を 1 回以上繰り返します。
?	前の項目をゼロ回または 1 回繰り返します。
{m}	前の項目をちょうど m 回だけ繰り返します。
{m,}	前の項目を m 回またはそれ以上の回数繰り返します。

演算子	説明
`{m,n}`	前の項目を少なくとも m 回、多くても n 回繰り返します。
`()`	グループ項目を括弧で囲んで、単一の論理項目にします。
`[...]`	角括弧式は、POSIX 正規表現の場合のように、文字クラスを指定します。

例

次の表に、SIMILAR TO を使用したパターンマッチングの例を示します。

式	戻り値
`'abc' SIMILAR TO 'abc'`	True
`'abc' SIMILAR TO '_b_'`	True
`'abc' SIMILAR TO '_A_'`	False
`'abc' SIMILAR TO '%(b|d)%'`	True
`'abc' SIMILAR TO '(b|c)%'`	False
`'AbcAbcdefgefg12efgefg12' SIMILAR TO '((Ab)?c)+d((efg)+(12))+'`	True
`'aaaaaab11111xy' SIMILAR TO 'a{6}_[0-9]{5}(x|y){2}'`	True
`'$0.87' SIMILAR TO '$[0-9]+(.[0-9][0-9])?'`	True

次の例では、名前に「E」または「H」が含まれる市を見つけます。

```
SELECT DISTINCT city FROM users
WHERE city SIMILAR TO '%E%|%H%' ORDER BY city LIMIT 5;
```

```
      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

次の例では、デフォルトのエスケープ文字列 ('\\') を使用して「_」を含む文字列を検索します。

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start\\_%'
ORDER BY tablename, "column" LIMIT 5;

        tablename          |       column
---------------------------+---------------------
 stcs_abort_idle           | idle_start_time
 stcs_abort_idle           | txn_start_time
 stcs_analyze_compression  | start_time
 stcs_auto_worker_levels   | start_level
 stcs_auto_worker_levels   | start_wlm_occupancy
```

次の例では、エスケープ文字列として '^' を指定し、エスケープ文字列を使用して「_」を含む文字列を検索します。

```
SELECT tablename, "column" FROM pg_table_def
WHERE "column" SIMILAR TO '%start^_%' ESCAPE '^'
ORDER BY tablename, "column" LIMIT 5;

        tablename          |       column
---------------------------+---------------------
 stcs_abort_idle           | idle_start_time
 stcs_abort_idle           | txn_start_time
 stcs_analyze_compression  | start_time
 stcs_auto_worker_levels   | start_level
 stcs_auto_worker_levels   | start_wlm_occupancy
```

POSIX 演算子

POSIX 正規表現は、マッチパターンを指定する一連の文字です。文字列が正規表現で記述された正規セットのメンバーであれば、その文字列は正規表現と一致します。

POSIX 正規表現は、LIKE および SIMILAR TO の演算子の場合よりも強力なパターンマッチング手段を提供します。POSIX 正規表現のパターンは、パターンが文字列全体と一致した場合にのみ true を返す SIMILAR TO 演算子の場合とは異なり、文字列の任意の部分と一致することができます。

> **Note**
>
> POSIX 演算子を使用する正規表現マッチングは、計算コストが高くなります。非常に多くの行を処理する場合は特に、可能な限り、LIKE を使用することをお勧めします。例えば、以下に示す各クエリは機能的には同じですが、LIKE を使用したクエリは、正規表現を使用したクエリよりも数倍速く実行できます。
>
> ```
> select count(*) from event where eventname ~ '.*(Ring|Die).*';
> select count(*) from event where eventname LIKE '%Ring%' OR eventname LIKE '%Die
> %';
> ```

構文

```
expression [ ! ] ~ pattern
```

引数

expression

列名など、有効な UTF-8 文字式。

!

拒否演算子。正規表現パターンに一致しません。

~

式の部分文字列に対して大文字/小文字を区別するマッチングを実行します。

> **Note**
>
> ~~ は LIKE の同義語です。

pattern

正規表現パターンを表す文字列リテラル。

pattern にワイルドカード文字が含まれていない場合、pattern は文字列そのものを表現するにすぎません。

'. * | ? ' などのメタ文字を含む文字列を検索するには、2 つのバックスラッシュ (' \\') を使用して文字をエスケープします。`SIMILAR TO` や `LIKE` と異なり、POSIX 正規表現の構文はユーザー定義のエスケープ文字をサポートしていません。

どちらの文字式も CHAR または VARCHAR のデータ型になることができます。文字式の型が異なる場合、Amazon Redshift は pattern のデータ型を expression のデータ型に変換します。

文字式はすべて CHAR または VARCHAR のデータ型にすることができます。文字式のデータ型が異なる場合、Amazon Redshift はそれらのデータ型を expression のデータ型に変換します。

POSIX パターンマッチングでは、以下のメタ文字がサポートされています。

POSIX	説明
.	任意の 1 文字をマッチングします。
*	直前文字の 0 回以上の出現にマッチングします。
+	直前文字の 1 回以上の出現にマッチングします。
?	直前文字の 0 回または 1 回の出現にマッチングします。
|	いずれかへのマッチングを指定します。たとえば `E | H` は `E` または `H` にマッチングします。
^	行頭の文字にマッチングします。
$	行末の文字にマッチングします。
$	文字列の末尾をマッチングします。
[ ]	角かっこでマッチングする文字のリストを指定します。リスト内のいずれかの文字にマッチングします。マッチングしない文字のリストにはその前にキャレット (^) を置きます。リスト内のいずれの文字以外の文字にマッチングします。
( )	グループ項目を括弧で囲んで、単一の論理項目にします。

POSIX	説明
{m}	前の項目をちょうど m 回だけ繰り返します。
{m,}	前の項目を m 回またはそれ以上の回数繰り返します。
{m,n}	前の項目を少なくとも m 回、多くても n 回繰り返します。
[: :]	POSIX 文字クラスのいずれかの文字とマッチングします。[:alnum:] 、[:alpha:] 、[:lower:] 、[:upper:] の文字クラスの場合 、Amazon Redshift では ASCII 文字のみがサポートされています。

Amazon Redshift では、以下の POSIX 文字クラスがサポートされています。

文字クラス	説明
[[:alnum:]]	すべての ASCII 英数字
[[:alpha:]]	すべての ASCII 英字
[[:blank:]]	すべての空白文字
[[:cntrl:]]	すべての制御文字 (印刷対象外)
[[:digit:]]	すべての数字
[[:lower:]]	すべての小文字の ASCII 英字
[[:punct:]]	すべての句読点
[[:space:]]	すべての空白文字 (印刷対象外)
[[:upper:]]	すべての大文字の ASCII 英字
[[:xdigit:]]	すべての有効な 16 進数文字

Amazon Redshift では、Perl の影響を受けた以下の演算子が正規表現でサポートされています。2 つのバックスラッシュ (「\\」) を使用して演算子をエスケープします。

演算子	説明	同等の文字のクラス式
`\\d`	数字	`[[:digit:]]`
`\\D`	数字以外の文字	`[^[:digit:]]`
`\\w`	単語	`[[:word:]]`
`\\W`	単語以外	`[^[:word:]]`
`\\s`	空白文字	`[[:space:]]`
`\\S`	空白文字以外	`[^[:space:]]`
`\\b`	単語の境界	

例

次の表に、POSIX 演算子を使用したパターンマッチングの例を示します。

式	戻り値
`'abc' ~ 'abc'`	True
`'abc' ~ 'a'`	True
`'abc' ~ 'A'`	False
`'abc' ~ '.*(b|d).*'`	True
`'abc' ~ '(b|c).*'`	True
`'AbcAbcdefgefg12efgefg12' ~ '((Ab)?c)+d((efg)+(12))+'`	True
`'aaaaaab11111xy' ~ 'a{6}.[1]{5}(x|y){2}'`	True
`'$0.87' ~ '\\$[0-9]+(\\.[0-9][0-9])?'`	True

式	戻り値
`'ab c' ~ '[[:space:]]'`	True
`'ab c' ~ '\\s'`	True
`' ' ~ '\\S'`	False

以下の例では、名前に E または H が含まれる市を見つけます。

```
SELECT DISTINCT city FROM users
WHERE city ~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Agoura Hills
 Auburn Hills
 Benton Harbor
 Beverly Hills
 Chicago Heights
```

以下の例では、名前に E または H が含まれない市を見つけます。

```
SELECT DISTINCT city FROM users WHERE city !~ '.*E.*|.*H.*' ORDER BY city LIMIT 5;

      city
-----------------
 Aberdeen
 Abilene
 Ada
 Agat
 Agawam
```

以下の例では、エスケープ文字列 ('\\') を使用してピリオドを含む文字列を検索します。

```
SELECT venuename FROM venue
WHERE venuename ~ '.*\\..*'
ORDER BY venueid;

          venuename
------------------------------
```

```
St. Pete Times Forum
Jobing.com Arena
Hubert H. Humphrey Metrodome
U.S. Cellular Field
Superpages.com Center
E.J. Nutter Center
Bernard B. Jacobs Theatre
St. James Theatre
```

BETWEEN 範囲条件

BETWEEN 条件では、キーワード BETWEEN および AND を使用して、値が範囲内に入っているかどうか式をテストします。

構文

```
expression [ NOT ] BETWEEN expression AND expression
```

式は、数値データ型、文字データ型、または日時データ型とすることができますが、互換性を持つ必要があります。範囲は両端を含みます。

例

最初の例では、2、3、または 4 のいずれかのチケットの販売を登録したトランザクション数をカウントします。

```
select count(*) from sales
where qtysold between 2 and 4;

count
--------
104021
(1 row)
```

範囲条件は開始値と終了値を含みます。

```
select min(dateid), max(dateid) from sales
where dateid between 1900 and 1910;

min  | max
-----+-----
```

```
1900 | 1910
```

範囲条件内の最初の式は最小値、2 番目の式は最大値である必要があります。次の例は、式の値のせいで、常にゼロ行を返します。

```
select count(*) from sales
where qtysold between 4 and 2;

count
-------
0
(1 row)
```

しかし、NOT 修飾子を加えると、論理が反転し、すべての行がカウントされます。

```
select count(*) from sales
where qtysold not between 4 and 2;

count
--------
172456
(1 row)
```

次のクエリは、20000～50000 席を備えた会場のリストを返します。

```
select venueid, venuename, venueseats from venue
where venueseats between 20000 and 50000
order by venueseats desc;

venueid |       venuename               | venueseats
---------+-------------------------------+------------
116 | Busch Stadium                 |      49660
106 | Rangers BallPark in Arlington |      49115
96 | Oriole Park at Camden Yards   |      48876
...
(22 rows)
```

次の例は、日付値に BETWEEN を使用する方法を示しています。

```
select salesid, qtysold, pricepaid, commission, saletime
from sales
where eventid between 1000 and 2000
```

```
   and saletime between '2008-01-01' and '2008-01-03'
order by saletime asc;

salesid | qtysold | pricepaid | commission |   saletime
--------+---------+-----------+------------+---------------
  65082 |       4 |       472 |       70.8 | 1/1/2008 06:06
 110917 |       1 |       337 |      50.55 | 1/1/2008 07:05
 112103 |       1 |       241 |      36.15 | 1/2/2008 03:15
 137882 |       3 |      1473 |     220.95 | 1/2/2008 05:18
  40331 |       2 |        58 |        8.7 | 1/2/2008 05:57
 110918 |       3 |      1011 |     151.65 | 1/2/2008 07:17
  96274 |       1 |       104 |       15.6 | 1/2/2008 07:18
 150499 |       3 |       135 |      20.25 | 1/2/2008 07:20
  68413 |       2 |       158 |       23.7 | 1/2/2008 08:12
```

BETWEEN の範囲は包括的ですが、日付はデフォルトで 00:00:00 の時刻値であることに注意してください。サンプルクエリで有効な 1 月 3 日の行は、販売時間が `1/3/2008 00:00:00` の行だけです。

Null 条件

Null 条件は、値が見つからないか、値が不明であるときに、Null かどうかテストします。

構文

```
expression IS [ NOT ] NULL
```

引数

expression

列のような任意の式。

IS NULL

式の値が Null の場合は true で、式が値を持つ場合は false です。

IS NOT NULL

式の値が Null の場合は false で、式が値を持つ場合は true です。

例

この例では、SALES テーブルの QTYSOLD フィールドで Null が何回検出されるかを示します。

```
select count(*) from sales
where qtysold is null;
count
-------
0
(1 row)
```

EXISTS 条件

EXISTS 条件は、サブクエリ内に行が存在するかどうかをテストし、サブクエリが少なくとも 1 つの行を返した場合に true を返します。NOT が指定されると、条件はサブクエリが行を返さなかった場合に true を返します。

構文

```
[ NOT ] EXISTS (table_subquery)
```

引数

EXISTS

table_subquery が少なくとも 1 つの行を返した場合に true となります。

NOT EXISTS

table_subquery が行を返さない場合に true になります。

table_subquery

評価結果として 1 つまたは複数の列と 1 つまたは複数の行を持つテーブルを返します。

例

この例では、任意の種類の販売があった日付ごとに、1 回ずつ、すべての日付識別子を返します。

```
select dateid from date
where exists (
select 1 from sales
where date.dateid = sales.dateid
)
order by dateid;

dateid
```

```
--------
1827
1828
1829
...
```

IN 条件

IN 条件は、一連の値の中に、またはサブクエリ内にあるメンバーシップの値をテストします。

構文

```
expression [ NOT ] IN (expr_list | table_subquery)
```

引数

expression

expr_list または table_subquery に対して評価される数値、文字、または日時であり、当該リストまたはサブクエリのデータ型との互換性が必要です。

expr_list

1 つまたは複数のカンマ区切り式、あるいは括弧で囲まれたカンマ区切り式の 1 つまたは複数のセット。

table_subquery

評価結果として 1 つまたは複数の行を持つテーブルを返すサブクエリですが、その選択リスト内の列数は 1 個に制限されています。

IN | NOT IN

IN は、式が式リストまたはクエリのメンバーである場合に true を返します。NOT IN は、式がメンバーでない場合に true を返します。expression の結果が Null である場合、または、一致する expr_list 値または table_subquery 値がなく、これらの比較行の 1 つ以上の結果が Null である場合、IN と NOT IN は NULL を返し、行は返されません。

例

次の条件は、リストされた値の場合にのみ true を返します。

```
qtysold in (2, 4, 5)
```

```
date.day in ('Mon', 'Tues')
date.month not in ('Oct', 'Nov', 'Dec')
```

大規模 IN リストの最適化

クエリのパフォーマンスを最適化するために、10 個を超える値が含まれる IN リストは内部的にスカラー配列として評価されます。10 個未満の値が含まれる IN リストは一連の OR 述語として評価されます。SMALLINT、INTEGER、BIGINT、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、TIMESTAMP、および TIMESTAMPTZ データ型では最適化がサポートされています。

この最適化の効果を確認するには、クエリの EXPLAIN 出力を調べてください。次に例を示します。

```
explain select * from sales
QUERY PLAN
--------------------------------------------------------------------
XN Seq Scan on sales  (cost=0.00..6035.96 rows=86228 width=53)
Filter: (salesid = ANY ('{1,2,3,4,5,6,7,8,9,10,11}'::integer[]))
(2 rows)
```

SQL コマンド

SQL 言語は、データベースオブジェクトの作成と操作、クエリの実行、テーブルのロード、およびテーブルデータの変更に使用するコマンドから構成されます。

Amazon Redshift は PostgreSQL に基づいています。Amazon Redshift と PostgreSQL の間には非常に重要な相違点がいくつかあり、データウェアハウスアプリケーションの開発や設計に際しては、それらを念頭に置く必要があります。Amazon Redshift SQL と PostgreSQL の違いについては、Amazon Redshift および PostgreSQLを参照してください。

> Note
>
> 単一 SQL ステートメントの最大サイズは 16 MB です。

トピック

- ABORT
- ALTER DATABASE
- ALTER DATASHARE

- DROP ID プロバイダー
- DROP LIBRARY
- DROP MASKING POLICY
- DROP MODEL
- DROP MATERIALIZED VIEW
- DROP PROCEDURE
- DROP RLS POLICY
- DROP ROLE
- DROP SCHEMA
- DROP TABLE
- DROP USER
- DROP VIEW
- END
- EXECUTE
- EXPLAIN
- FETCH
- GRANT
- INSERT
- INSERT (外部テーブル)
- LOCK
- MERGE
- PREPARE
- REFRESH MATERIALIZED VIEW
- RESET
- REVOKE
- ROLLBACK
- SELECT
- SELECT INTO
- SET
- SET SESSION AUTHORIZATION

- SET SESSION CHARACTERISTICS
- SHOW
- SHOW COLUMNS
- SHOW EXTERNAL TABLE
- SHOW DATABASES
- SHOW MODEL
- SHOW DATASHARES
- SHOW PROCEDURE
- SHOW SCHEMAS
- テーブルを表示する
- SHOW TABLES
- ビューを表示する
- START TRANSACTION
- TRUNCATE
- UNLOAD
- UPDATE
- VACUUM

ABORT

現在実行中のトランザクションを停止し、そのトランザクションで行われたすべての更新を破棄します。ABORT はすでに完了したトランザクションに対しては影響を与えません。

このコマンドは ROLLBACK コマンドと同じ機能を実行します。詳細については、ROLLBACKを参照してください。

構文

```
ABORT [ WORK | TRANSACTION ]
```

パラメータ

WORK

オプションキーワード

TRANSACTION

オプションキーワード。WORK と TRANSACTION は同義語です。

例

次の例では、テーブルを作成し、データがそのテーブルに挿入されるトランザクションを開始します。次に、ABORT コマンドはデータの挿入をロールバックし、テーブルを空にします。

次のコマンドを実行すると、MOVIE_GROSS という名前のテーブルが作成されます。

```
create table movie_gross( name varchar(30), gross bigint );
```

次のコマンドセットを実行すると、2 つのデータ行をテーブルに挿入するトランザクションが開始されます。

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

その後、次のコマンドを実行すると、テーブルからデータが選択され、挿入が正常に実行されたことが示されます。

```
select * from movie_gross;
```

コマンド出力は、両方の行が正しく挿入されたことを示します。

```
         name           |  gross
------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

このコマンドはデータ変更を、トランザクションの開始時点までロールバックします。

```
abort;
```

テーブルからデータを選択すると、空のテーブルが表示されます。

```
select * from movie_gross;

 name | gross
------+-------
(0 rows)
```

ALTER DATABASE

データベースの属性を変更します。

必要な権限

ALTER DATABASE を使用するには、次の権限のいずれかが必要です。

- スーパーユーザー
- ALTER DATASHARE 権限を持つユーザー
- データベースの所有者

構文

```
ALTER DATABASE database_name
{ RENAME TO new_name
| OWNER TO new_owner
| CONNECTION LIMIT { limit | UNLIMITED }
| COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }
| ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }
| INTEGRATION {{SET REFRESH_INTERVAL <interval>} | REFRESH {{ ALL | INERROR } TABLES
 [IN SCHEMA schema [, ...]] | TABLE schema.table [, ...]}}
}
```

パラメータ

database_name

変更するデータベースの名前。通常、現在接続されていないデータベースを変更します。いずれの場合も、変更は後のセッションにのみ反映されます。現在のデータベースの所有者を変更できますが、名前を変更することはできません。

```
alter database tickit rename to newtickit;
ERROR:  current database may not be renamed
```

RENAME TO

指定したデータベースの名前を変更します。有効な名前の詳細については、「名前と識別子」を参照してください。dev、padb_harvest、template0、template1、または sys:internal の各データベースの名前を変更することはできません。また、現在のデータベースの名前も変更できません。データベース所有者またはsuperuser (p. 911)のみがデータベース名を変更できます。スーパーユーザー以外の所有者には CREATEDB 権限も必要です。

new_name

新しいデータベース名。

OWNER TO

指定したデータベースの所有者を変更します。現在のデータベースまたは他のデータベースの所有者を変更できます。スーパーユーザーのみが所有者を変更できます。

new_owner

新しいデータベース所有者。新しい所有者は、書き込み権限を持つ既存のデータベースユーザーであることが必要です。ユーザー権限の詳細については、GRANTを参照してください。

CONNECTION LIMIT { limit | UNLIMITED }

ユーザーが同時に開けるデータベース接続の最大数。この制限はスーパーユーザーには適用されません。同時接続の最大数を許可するには、UNLIMITED キーワードを使用します。ユーザーごとの接続数の制限が適用される場合もあります。詳細については、「CREATE USER」を参照してください。デフォルトは UNLIMITED です。現在の接続を確認するには、STV_SESSIONSシステムビューに対してクエリを実行します。

> Note
>
> ユーザーとデータベースの両方の接続制限が適用される場合は、ユーザーが接続しようとしたときに、両方の制限内に未使用の接続スロットがなければなりません。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

文字列の検索または比較において、大文字と小文字を区別するか、あるいは区別しないかを指定する句。

現在、空の状態のデータベースでは、大文字と小文字を区別するかどうかの設定を変更することができます。

大文字と小文字の区別を変更するには、そのデータベースに対する権限が必要です。CREATE DATABASE 権限を持つスーパーユーザーまたはデータベース所有者も、データベースで大文字と小文字を区別するかどうかを設定できます。

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

データベースに対してクエリを実行するときに使用される分離レベルを指定する句。

- 直列化可能な分離 – 同時トランザクションの完全な直列化機能を提供します。詳細については、「直列化可能分離」を参照してください。
- スナップショットの分離 – 更新および削除の競合に対する保護機能を備えた分離レベルを提供します。

分離レベルの詳細については、「CREATE DATABASE」を参照してください。

データベースの分離レベルを変更するときは、以下の点を考慮します。

- データベースの分離レベルを変更するには、スーパーユーザー権限または CREATE DATABASE 権限が必要です。
- dev データベースの分離レベルは変更できません。
- トランザクションブロック内の分離レベルは変更できません。
- 他のユーザーがデータベースに接続している場合、分離レベルの変更コマンドは失敗します。
- 分離レベルの変更コマンドでは、現在のセッションの分離レベルの設定を変更できます。

INTEGRATION REFRESH {{ ALL | INERROR } TABLES [IN SCHEMA schema [, ...]] | TABLE schema.table [, ...]}

Amazon Redshift がすべてのテーブルを更新するか、あるいは指定されたスキーマやテーブル内でエラーのあるテーブルを更新するかどうかを指定する句。更新によって、指定されたスキーマやテーブル内のテーブルはソースデータベースから完全に複製されます。

詳細については、「Amazon Redshift 管理ガイド」の「ゼロ ETL 統合での作業」を参照してください。統合の状態の詳細については、「SVV_INTEGRATION_TABLE_STATE」と「SVV_INTEGRATION」を参照してください。

INTEGRATION {SET REFRESH_INTERVAL <interval>}

SET REFRESH_INTERVAL 句は、ゼロ ETL ソースからターゲットデータベースにデータを更新するおおよその時間間隔を秒単位で設定します。この `interval` は、ソースタイプが Aurora

MySQL、Aurora PostgreSQL、または RDS for MySQL のゼロ ETL 統合では、0～432,000 秒 (5 日) に設定できます。Amazon DynamoDB ゼロ ETL 統合の場合、`interval` は 900～432,000 秒 (15 分～5 日) に設定できます。

ゼロ ETL 統合を使用したデータベースの作成の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift でのデスティネーションデータベースの作成」を参照してください。

使用に関する注意事項

ALTER DATABASE コマンドは現在のセッションではなく、後のセッションに適用されます。変更の反映を確認するには、変更されたデータベースに再接続する必要があります。

例

次の例では、TICKIT_SANDBOX という名前のデータベースを TICKIT_TEST に変更します。

```
alter database tickit_sandbox rename to tickit_test;
```

次の例では、TICKIT データベース (現在のデータベース) の所有者を DWUSER に変更します。

```
alter database tickit owner to dwuser;
```

次の例では、sampledb データベースで大文字小文字を区別するかどうかの設定を変更しています。

```
ALTER DATABASE sampledb COLLATE CASE_INSENSITIVE;
```

次の例では、スナップショット分離レベルを使用して **sampledb** という名前のデータベースを変更します。

```
ALTER DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

次の例では、ゼロ ETL 統合で、データベース **sample_integration_db** 内のテーブル **schema1.sample_table1** と **schema2.sample_table2** を更新します。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH TABLE schema1.sample_table1,
 schema2.sample_table2;
```

次の例では、ゼロ ETL 統合で同期されたテーブルと失敗したテーブルをすべて更新します。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH ALL tables;
```

次の例では、ゼロ ETL 統合の更新間隔を 600 秒に設定します。

```
ALTER DATABASE sample_integration_db INTEGRATION SET REFRESH_INTERVAL 600;
```

次の例では、スキーマ **sample_schema** 内で `ErrorState` となっているテーブルをすべて更新します。

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH INERROR TABLES in SCHEMA
 sample_schema;
```

ALTER DATASHARE

データ共有の定義を変更します。ALTER DATASHARE を使用して、オブジェクトを追加または削除できます。現在のデータベース内のデータ共有のみを変更できます。関連付けられたデータベースからデータ共有にオブジェクトを追加または削除します。追加または削除するデータ共有オブジェクトに対して必要な許可を持つデータ共有の所有者は、データ共有を変更できます。

必要な権限

以下に、ALTER DATASHARE に必要な権限を示します。

- スーパーユーザー。
- ALTER DATASHARE の権限を持つユーザー。
- データ共有に対する ALTER または ALL の権限を持つユーザー
- 特定のオブジェクトをデータ共有に追加しようとするユーザーには、対象のオブジェクトに対する権限が必要です。この場合、ユーザーはオブジェクトの所有者であるか、オブジェクトに対する SELECT、USAGE、あるいは ALL の権限を持っている必要があります。

構文

次の構文は、データ共有に対して、オブジェクトを追加または削除する方法を示しています。

```
ALTER DATASHARE datashare_name { ADD | REMOVE } {
TABLE schema.table [, ...]
```

```
| SCHEMA schema [, ...]
| FUNCTION schema.sql_udf (argtype,...) [, ...]
| ALL TABLES IN SCHEMA schema [, ...]
| ALL FUNCTIONS IN SCHEMA schema [, ...] }
```

次の構文は、データ共有のプロパティを設定する方法を示しています。

```
ALTER DATASHARE datashare_name {
[ SET PUBLICACCESSIBLE [=] TRUE | FALSE ]
[ SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema ] }
```

パラメータ

datashare_name

変更するデータ共有の名前。

ADD | REMOVE

データ共有にオブジェクトを追加するか、データ共有からオブジェクトを削除するかを指定する句。

TABLE schema.table [, ...]

データ共有に追加する、指定されたスキーマ内のテーブルまたはビューの名前。

SCHEMA schema [, ...]

データ共有に追加するスキーマの名前。

FUNCTION schema.sql_udf (argtype,...) [, ...]

データ共有に追加する引数タイプを伴うユーザー定義の SQL 関数の名前。

ALL TABLES IN SCHEMA schema [, ...]

指定されたスキーマ内のすべてのテーブルをデータ共有に追加するかどうかを指定する句。

ALL FUNCTIONS IN SCHEMA schema [, ...] }

指定されたスキーマ内のすべての関数をデータ共有に追加することを指定する句。

[SET PUBLICACCESSIBLE [=] TRUE | FALSE]

公開でアクセス可能なクラスターとデータ共有を共有できるかどうかを指定する句。

[SET INCLUDENEW [=] TRUE | FALSE FOR SCHEMA schema]

指定したスキーマで作成される将来のテーブル、ビュー、または SQL ユーザー定義関数 (UDF) をデータ共有に追加するかどうかを指定する句。指定したスキーマにある現在のテーブル、ビュー、または SQL UDF は、データ共有に追加されません。データ共有とスキーマの各ペアについて、このプロパティを変更できるのはスーパーユーザーのみです。INCLUDENEW 句はデフォルトで false を返します。

ALTER DATASHARE の使用に関する注意事項

- 次のユーザーは、データ共有を変更できます。
  - スーパーユーザー
  - データ共有の所有者
  - データ共有に対する ALTER または ALL 権限を持つユーザー
- 特定のオブジェクトをデータ共有に追加しようとするユーザーには、そのオブジェクトに対する適切な権限が必要です。ユーザーはオブジェクトの所有者であるか、オブジェクトに対する SELECT、USAGE、ALL 権限を持っている必要があります。
- スキーマ、テーブル、通常のビュー、遅延バインディングビュー、マテリアライズドビュー、および SQL ユーザー定義関数 (UDF) を共有できます。スキーマにオブジェクトを追加する前に、まず対象のスキーマをデータ共有に追加します。

  スキーマを追加する場合、Amazon Redshift はその下にすべてのオブジェクトを追加するわけではありません。それらを明示的に追加する必要があります。
- AWS Data Exchange データ共有は、パブリックアクセス可能設定を有効にした状態で作成することをお勧めします。
- 一般的に、パブリックアクセスを無効にするために ALTER DATASHARE ステートメントを使用して AWS Data Exchange データ共有を変更することはお勧めしません。そうすると、AWS アカウントのクラスターがパブリックアクセス可能である場合に、データ共有にアクセスできるアカウントがアクセスできなくなります。このタイプの変更を実行すると、AWS Data Exchangeのデータ製品での使用条件に違反する可能性があります。この推奨事項の例外については、以下を参照してください。

  次に、パブリックにアクセス可能な設定を無効にして AWS Data Exchange データ共有を作成した場合に発生する、エラーの例を示します。

  ```
  ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
  ```

```
ERROR:  Alter of ADX-managed datashare salesshare requires session variable
 datashare_break_glass_session_var to be set to value 'c670ba4db22f4b'
```

パブリックにアクセス可能な設定を無効にできるよう AWS Data Exchange データ共有を変更するには、次の変数を設定した上で、ALTER DATASHARE ステートメントを再度実行します。

```
SET datashare_break_glass_session_var to 'c670ba4db22f4b';
```

```
ALTER DATASHARE salesshare SET PUBLICACCESSIBLE FALSE;
```

この場合、Amazon Redshift は 1 回限り有効なランダム値を生成し、その値により、AWS Data Exchangeデータ共有の ALTER DATASHARE SET PUBLICACCESSIBLE FALSE を許可するようにセッション変数を設定します。

例

次の例では、スキーマ `public` をデータ共有 `salesshare` に追加します。

```
ALTER DATASHARE salesshare ADD SCHEMA public;
```

次の例では、テーブル `public.tickit_sales_redshift` をデータ共有 `salesshare` に追加します。

```
ALTER DATASHARE salesshare ADD TABLE public.tickit_sales_redshift;
```

次の例では、すべてのテーブルをデータ共有 `salesshare` に追加します。

```
ALTER DATASHARE salesshare ADD ALL TABLES IN SCHEMA PUBLIC;
```

次の例では、データ共有 `salesshare` からテーブル `public.tickit_sales_redshift` を削除します。

```
ALTER DATASHARE salesshare REMOVE TABLE public.tickit_sales_redshift;
```

ALTER DEFAULT PRIVILEGES

指定したユーザーによって今後作成されるオブジェクトに対して、デフォルトで適用するアクセス許可のセットを定義します。デフォルトでは、ユーザーは自分のデフォルトのアクセス許可のみ変更で

きます。他のユーザーに対しては、スーパーユーザーのみがデフォルトのアクセス許可を指定できます。

デフォルト権限は、ロール、ユーザー、またはユーザーグループに適用できます。デフォルトのアクセス許可は、現在のデータベースに作成されているすべてのオブジェクトにグローバルに設定することも、指定したスキーマに作成されているオブジェクトにのみ設定することもできます。

デフォルトのアクセス許可は、新しいオブジェクトにのみ適用されます。ALTER DEFAULT PRIVILEGES を実行しても、既存のオブジェクトのアクセス許可は変更されません。データベースまたはスキーマ内の任意のユーザーが作成した現在および将来のすべてのオブジェクトに対するアクセス許可を付与するには、「スコープ設定アクセス許可」を参照してください。

データベースユーザーのデフォルト権限に関する情報を表示するには、PG_DEFAULT_ACLシステムカタログテーブルをクエリします。

権限の詳細については、「GRANT」を参照してください。

必要な権限

ALTER DEFAULT PRIVILEGES 必要な権限は以下のとおりです。

- スーパーユーザー
- ALTER DEFAULT PRIVILEGES の権限を持つユーザー
- 自身のデフォルトのアクセス権限を変更しているユーザー
- 自身がアクセス権限を持つスキーマの権限を設定しているユーザー

構文

```
ALTER DEFAULT PRIVILEGES
    [ FOR USER target_user [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    grant_or_revoke_clause

where grant_or_revoke_clause is one of:

GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
 ON TABLES
 TO { user_name [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 TO { user_name [ WITH GRANT OPTION ] |  ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

REVOKE [ GRANT OPTION FOR ] { { SELECT | INSERT | UPDATE | DELETE | REFERENCES |
 TRUNCATE } [,...] | ALL [ PRIVILEGES ] }
 ON TABLES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE  { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRUNCATE } [,...] | ALL
 [ PRIVILEGES ] }
 ON TABLES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM user_name [, ...] [ RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON FUNCTIONS
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ] { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM user_name [, ...] [ RESTRICT ]

REVOKE { EXECUTE | ALL [ PRIVILEGES ] }
 ON PROCEDURES
 FROM { ROLE role_name | GROUP group_name | PUBLIC } [, ...] [ RESTRICT ]
```

パラメータ

FOR USER target_user

省略可能。デフォルト権限が定義されているユーザーの名前。他のユーザーに対しては、スーパーユーザーのみがデフォルト権限を指定できます。デフォルト値は現在のユーザーです。

IN SCHEMA schema_name

省略可能。IN SCHEMA 句が表示されている場合、指定したデフォルト権限は、指定の schema_name で作成された新しいオブジェクトに適用されます。この場合、ALTER DEFAULT PRIVILEGES のターゲットであるユーザーまたはユーザーグループは、指定されたスキーマに対する CREATE権限が必要です。スキーマ固有のデフォルト権限は、既存のグローバルなデフォルト権限に追加されます。デフォルトでは、デフォルト権限はデータベース全体にグローバルに適用されます。

GRANT

指定したユーザーが作成するすべての新しいテーブルとビュー、関数、またはストアドプロシージャについて、指定したユーザーやグループに付与する権限のセット。GRANT 句では、[GRANT](#)コマンドと同じ権限とオプションを設定できます。

WITH GRANT OPTION

権限を付与されるユーザーが、他のユーザーにも同じ権限を付与できることを示します。WITH GRANT OPTION をグループまたは PUBLIC に付与することはできません。

TO user_name | ROLE role_name | GROUP group_name

指定したデフォルト権限が適用されるユーザー、ロール、またはユーザーグループの名前。

REVOKE

指定したユーザーが作成するすべての新しいテーブル、関数、またはストアドプロシージャについて、指定したユーザーやグループから取り消す権限のセット。REVOKE 句では、[REVOKE](#)コマンドと同じ権限とオプションを設定できます。

GRANT OPTION FOR

他のユーザーに特定の権限を付与するオプションのみを取り消し、権限自体は取り消しません。グループや PUBLIC の GRANT OPTION を取り消すことはできません。

FROM user_name | ROLE role_name | GROUP group_name

指定した権限をデフォルトで取り消すユーザー、ロール、またはユーザーグループの名前。

RESTRICT

RESTRICT オプションは、ユーザーが直接付与した権限のみを取り消します。これがデフォルトです。

例

ユーザーグループ report_readers に属する任意のユーザーに対して、ユーザー report_admin が作成したすべてのテーブルとビューを表示することを許可するとします。この場合は、スーパーユーザーとして次のコマンドを実行します。

```
alter default privileges for user report_admin grant select on tables to group
 report_readers;
```

次の例では、最初のコマンドにより、作成するすべての新しいテーブルとビューに対して SELECT 権限が付与されます。

```
alter default privileges grant select on tables to public;
```

次の例では、sales_adminスキーマで作成するすべての新しいテーブルとビューに対して、salesユーザーグループに INSERT 権限が付与されます。

```
alter default privileges in schema sales grant insert on tables to group sales_admin;
```

次の例では、前の例とは逆に ALTER DEFAULT PRIVILEGES コマンドで権限を取り消します。

```
alter default privileges in schema sales revoke insert on tables from group
 sales_admin;
```

デフォルトでは、PUBLIC ユーザーグループはすべての新しいユーザー定義関数に対する実行許可を付与されます。新しい関数に対する public の実行許可を取り消して、dev_testユーザーグループにのみ実行許可を付与するには、次のコマンドを実行します。

```
alter default privileges revoke execute on functions from public;
alter default privileges grant execute on functions to group dev_test;
```

ALTER EXTERNAL VIEW (プレビュー)

これは、プレビュー版の Amazon Redshift のデータカタログについて記載した暫定版ドキュメントです。ドキュメントと機能はどちらも変更されることがあります。この機能は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

Amazon Redshift クラスターを [プレビュー] で作成して、Amazon Redshift の新機能をテストできます。これらの機能を本番稼働で使用したり、[プレビュー] クラスターを本稼働クラスターや別のトラックのクラスターに移動したりすることはできません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

[Preview] (プレビュー) でクラスターを作成するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Provisioned clusters dashboard] (プロビジョニングされたクラスターダッシュボード) を選択し、[Clusters] (クラスター) を選択します。現在の AWS リージョンにあるアカウントのクラスターがリストされています。各クラスターのプロパティのサブセットが、リストの列に表示されます。
3. [Clusters] (クラスター) リストページに、プレビューを紹介するバナーが表示されます。[Create preview cluster] (プレビュークラスターの作成) ボタンを選択して、クラスターの作成ページを開きます。
4. クラスターのプロパティを入力します。テストしたい機能を含む [プレビュートラック] を選択します。プレビュートラックにあることを示すクラスターの名前を入力することをお勧めします。テストする機能について、-preview というラベルの付いたオプションを含む、クラスターのオプションを選択します。クラスター作成の詳細については、「Amazon Redshift 管理ガイド」の「クラスターの作成」を参照してください。
5. [クラスターを作成] を選択して、プレビューのクラスターを作成します。

> **Note**
>
> `preview_2023` トラックは、利用可能な最新のプレビュートラックです。このトラックは RA3 ノードタイプのクラスターの作成のみをサポートしています。ノードタイプ DC2 以前のノードタイプはサポートされていません。

6. プレビュークラスターが使用可能になったら、SQL クライアントを使用してデータをロードし、クエリを実行します。

データカタログビューのプレビュー機能は以下のリージョンでのみ利用できます。

- 米国東部 (オハイオ) (us-east-2)
- 米国東部 (バージニア北部) (us-east-1)

- 米国西部 (北カリフォルニア) (us-west-1)
- アジアパシフィック (東京) (ap-northeast-1)
- 欧州 (アイルランド) (eu-west-1)
- 欧州 (ストックホルム) (eu-north-1)

データカタログビューをテストするためのプレビューワークグループを作成することもできます。これらの機能を本番稼働で使用したり、ワークグループを別のワークグループに移動したりすることはできません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。プレビューワークグループの作成方法については、「プレビューワークグループの作成」を参照してください。

ALTER EXTERNAL VIEW コマンドを使用して外部ビューを更新します。使用するパラメータによっては、このビューを参照できる Amazon Athena や Amazon EMR Spark などの他の SQL エンジンが影響を受ける可能性があります。データカタログビューの詳細については、「データカタログビューの作成 (プレビュー)」を参照してください。

構文

```
ALTER EXTERNAL VIEW schema_name.view_name
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
[FORCE] { AS (query_definition) | REMOVE DEFINITION }
```

パラメータ

schema_name.view_name

AWS Glue データベースにアタッチされているスキーマ。その後にビューの名前が続きます。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

ビューを変更するときに使用するスキーマの表記法。AWS Glue Data Catalog、作成した Glue データベース、または作成した外部スキーマを使用するように指定できます。詳細については、「CREATE DATABASE」と「CREATE EXTERNAL SCHEMA」を参照してください。

FORCE

テーブルで参照されているオブジェクトが他の SQL エンジンと矛盾している場合でも、AWS Lake Formation がビューの定義を更新する必要があるかどうか。Lake Formation がビューを更新

すると、他の SQL エンジンも更新されるまで、そのビューはそれらの SQL エンジンに対して古いものと見なされます。

AS query_definition

Amazon Redshift がビューを変更するために実行する SQL クエリの定義。

REMOVE DEFINITION

ビューを削除して再作成するかどうか。`PROTECTED` としてマークするには、ビューを削除して再作成する必要があります。

例

次の例では、sample_schema.glue_data_catalog_view という名前のデータカタログビューを変更します。

```
ALTER EXTERNAL VIEW sample_schema.glue_data_catalog_view
FORCE
REMOVE DEFINITION
```

ALTER FUNCTION

関数の名前変更または所有者の変更を行います。関数名とデータタイプの両方が必要です。所有者またはスーパーユーザーのみが関数名を変更できます。スーパーユーザーのみが関数の所有者を変更できます。

構文

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
     RENAME TO new_name
```

```
ALTER FUNCTION function_name ( { [ py_arg_name py_arg_data_type | sql_arg_data_type }
 [ , ... ] ] )
     OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

パラメータ

function_name

変更する関数の名前。関数名を現在の検索パスに指定するか、schema_name.function_name 形式で特定のスキーマを使用します。

py_arg_name py_arg_data_type | sql_arg_data_type

オプション。Python ユーザー定義関数の入力引数名とデータ型のリスト、または SQL ユーザー定義関数の入力引数データ型のリスト。

new_name

ユーザー定義関数の新しい名前。

new_owner | CURRENT_USER | SESSION_USER

ユーザー定義関数の新しい所有者。

例

次の例では、関数の名前を first_quarter_revenue から quarterly_revenue に変更します。

```
ALTER FUNCTION first_quarter_revenue(bigint, numeric, int)
         RENAME TO quarterly_revenue;
```

次の例は、quarterly_revenue 関数の所有者を etl_user に変更します。

```
ALTER FUNCTION quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER GROUP

ユーザーグループを変更します。ユーザーをグループに追加するか、グループからユーザーを削除するか、グループ名を変更するには、このコマンドを使用します。

構文

```
ALTER GROUP group_name
{
ADD USER username [, ... ] |
DROP USER username [, ... ] |
RENAME TO new_name
```

```
}
```

パラメータ

group_name

変更するユーザーグループの名前。

ADD

ユーザーをユーザーグループに追加します。

DROP

ユーザーグループからユーザーを削除します。

username

グループに追加するかグループから削除するユーザーの名前。

RENAME TO

ユーザーグループの名前を変更します。2 個のアンダースコアで始まるグループ名は Amazon Redshift 内部で使用するために予約されています。有効な名前の詳細については、「名前と識別子」を参照してください。

new_name

ユーザーグループの新しい名前。

例

次の例では、DWUSER という名前のユーザーを ADMIN_GROUP グループに追加します。

```
ALTER GROUP admin_group
ADD USER dwuser;
```

次の例では、グループ名を ADMIN_GROUP から ADMINISTRATORS に変更します。

```
ALTER GROUP admin_group
RENAME TO administrators;
```

次の例では、ADMIN_GROUP グループに 2 人のユーザーを追加します。

```
ALTER GROUP admin_group
```

```
ADD USER u1, u2;
```

次の例では、ADMIN_GROUP グループから 2 人のユーザーを削除します。

```
ALTER GROUP admin_group
DROP USER u1, u2;
```

他の ID プロバイダー

ID プロバイダーを変更して、新しいパラメータと値を割り当てます。このコマンドを実行すると、新しい値が割り当てられる前に、以前に設定したパラメータ値がすべて削除されます。スーパーユーザーのみが ID プロバイダーを変更できます。

構文

```
ALTER IDENTITY PROVIDER identity_provider_name
[PARAMETERS parameter_string]
[NAMESPACE namespace]
[IAM_ROLE iam_role]
[AUTO_CREATE_ROLES
    [ TRUE [ { INCLUDE | EXCLUDE } GROUPS LIKE filter_pattern] |
      FALSE
    ]
[DISABLE | ENABLE]
```

パラメータ

identity_provider_name

ID プロバイダーの名前。有効な名前の詳細については、「名前と識別子」を参照してください。

parameter_string

特定の ID プロバイダーに必要なパラメータと値を含む、適切にフォーマットされた JSON オブジェクトを含む文字列。

名前空間

組織の名前空間。

iam_role

IAM アイデンティティセンターへの接続に対するアクセス許可を提供する IAM ロール。このパラメータは、ID プロバイダーのタイプが AWSIDC である場合にのみ適用されます。

auto_create_roles

ロールの自動作成機能を有効または無効にします。SQL でオプションが指定されていない場合、デフォルトは FALSE、オプションが指定されているが値が指定されていない場合、デフォルトは TRUE です。

グループを含めるには、INCLUDE を指定します。デフォルトは空です。つまり、AUTO_CREATES_ROLES がオンの場合、すべてのグループが含まれます。

グループを除外するには、EXCLUDE を指定します。デフォルトは空です。つまり、AUTO_CREATES_ROLES がオンの場合、グループは除外されません。

DISABLE または ENABLE

ID プロバイダーをオンまたはオフにします。デフォルトは ENABLE です。

例

次の例では、oauth_standard という名前の ID プロバイダーを変更します。Microsoft Azure AD が ID プロバイダーである場合に特に適用されます。

```
ALTER IDENTITY PROVIDER oauth_standard
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

次のサンプルは、ID プロバイダー名前空間を設定する方法を示しています。これは、Microsoft Azure AD (前のサンプルのようなステートメントに従う場合)、または別の ID プロバイダーに適用できます。マネージドアプリケーションを介して接続を設定している場合、既存の Amazon Redshift でプロビジョニングされたクラスターまたは Amazon Redshift Serverless ワークグループを IAM アイデンティティセンターに接続する場合にも適用できます。

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
NAMESPACE 'MYCO';
```

次の IAM ロールを設定する例は、Redshift と IAM アイデンティティセンターの統合を設定するユースケースで使用できます。

```
ALTER IDENTITY PROVIDER "my-redshift-idc-application"
IAM_ROLE 'arn:aws:iam::123456789012:role/myadministratorrole';
```

Redshift から IAM アイデンティティセンターへの接続の設定の詳細については、「Redshift を IAM アイデンティティセンターに接続してユーザーにシングルサインオンエクスペリエンスを提供する」を参照してください。

ID プロバイダーの無効化

次のサンプルステートメントは、ID プロバイダーを無効にする方法を示しています。無効にした場合、再度有効にするまで、ID プロバイダーのフェデレーションユーザーはクラスターにログインできません。

```
ALTER IDENTITY PROVIDER "redshift-idc-app" DISABLE;
```

ALTER MASKING POLICY

既存の動的データマスキングポリシーを変更します。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、マスキングポリシーを変更できます。

構文

```
ALTER MASKING POLICY policy_name
   USING (masking_expression);
```

パラメータ

policy_name

マスキングポリシーの名前。これは、データベースに既に存在するマスキングポリシーの名前でなければなりません。

masking_expression

ターゲット列の変換に使用される SQL 式。文字列操作関数などのデータ操作関数を使用して記述することも、SQL、Python、または AWS Lambda で記述されたユーザー定義関数と組み合わせて記述することもできます。

式は、元の式の入力列およびデータ型に一致する必要があります。例えば、元のマスキングポリシーの入力列が `sample_1 FLOAT` と `sample_2 VARCHAR(10)` である場合、3 番目の列を使用するようにマスキングポリシーを変更したり、ポリシーが FLOAT と BOOLEAN を取るようにしたりすることはできません。マスク式として定数を使用する場合は、入力型と一致する型に明示的にキャストする必要があります。

マスキング式で使用するユーザー定義関数には USAGE アクセス許可が必要です。

ALTER MATERIALIZED VIEW

マテリアライズドビューの属性を変更します。

構文

```
ALTER MATERIALIZED VIEW mv_name
[ AUTO REFRESH { YES | NO } ]
[ ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [FOR DATASHARES] ];
```

パラメータ

mv_name

変更するマテリアライズドビューの名前。

AUTO REFRESH { YES | NO }

マテリアライズドビューの自動更新をオンまたはオフにする句。マテリアライズドビューの自動更新の詳細については、「マテリアライズドビューの更新」を参照してください。

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

リレーションの行レベルのセキュリティをオンまたはオフにする句。

リレーションで行レベルのセキュリティがオンになっている場合、行レベルのセキュリティポリシーでアクセスが許可されている行のみを読み取ることができます。リレーションへのアクセス権を付与するポリシーがない場合は、リレーションから行を表示できません。ROW LEVEL SECURITY 句を設定できるのは、スーパーユーザーと、`sys:secadmin` ロールを持つユーザーまたはロールのみです。詳細については、「行レベルのセキュリティ」を参照してください。

- [CONJUNCTION TYPE { AND | OR }]

リレーションの行レベルのセキュリティポリシーの結合タイプを選択できる句。1 つのリレーションに複数の行レベルのセキュリティポリシーがアタッチされている場合、それらのポリシーを AND 句や OR 句で組み合わせることができます。デフォルトでは、Amazon Redshift は RLS ポリシーを AND 句で組み合わせます。スーパーユーザーと、`sys:secadmin` ロールを持つユーザーまたはロールは、この句を使用して、リレーションの行レベルのセキュリティポリシーの結合タイプを定義できます。詳細については、「ユーザーごとに複数ポリシーの組み合わせ」を参照してください。

- FOR DATASHARES

  RLS で保護されたリレーションにデータ共有上でアクセスできるかどうかを決定する句。デフォルトでは、RLS で保護されたリレーションにデータ共有経由でアクセスすることはできません。この句を指定して実行される ALTER MATERIALIZED VIEW ROW LEVEL SECURITY コマンドは、リレーションのデータ共有アクセシビリティプロパティにのみ影響します。ROW LEVEL SECURITY のプロパティは変更されません。

  RLS で保護されたリレーションにデータ共有上でアクセスできるようにした場合、コンシューマー側のデータ共有データベースにおいて、そのリレーションには行レベルのセキュリティが適用されません。リレーションはプロデューサー側の RLS プロパティを保持します。

例

次の例では、`tickets_mv` マテリアライズドビューを自動的に更新できます。

```
ALTER MATERIALIZED VIEW tickets_mv AUTO REFRESH YES
```

RLS ポリシーの変更

テーブル上の既存の行レベルのセキュリティポリシーを変更します。

スーパーユーザーとユーザー、または `sys:secadmin` ロールを持つロールは、ポリシーを変更できます。

構文

```
ALTER RLS POLICY policy_name
USING ( using_predicate_exp );
```

パラメータ

policy_name

ポリシーの名前。

USING (using_predicate_exp)

クエリの WHERE 句に適用されるフィルターを指定します。Amazon Redshift は、クエリレベルのユーザー述語より先にポリシー述語を適用します。例えば、**`current_user = ‘joe’ and price > 10`** は Joe に対して価格が 10 USD を超えるレコードのみを表示するように制限します。

この式は、policy_name という名前のポリシーの作成に使用された CREATE RLS POLICY ステートメントの WITH 句で宣言された変数にアクセスできます。

例

次の例では、RLS ポリシーを変更します。

```
-- First create an RLS policy that limits access to rows where catgroup is 'concerts'.
CREATE RLS POLICY policy_concerts
WITH (catgroup VARCHAR(10))
USING (catgroup = 'concerts');

-- Then, alter the RLS policy to only show rows where catgroup is 'piano concerts'.
ALTER RLS POLICY policy_concerts
USING (catgroup = 'piano concerts');
```

ALTER ROLE

ロールの名前変更または所有者の変更を行います。Amazon Redshift でのシステム定義のロールのリストについては、「the section called “Amazon Redshift でのシステム定義のロール”」を参照してください。

必要なアクセス許可

ALTER ROLE に必要なアクセス許可を以下に示します。

- スーパーユーザー

- ALTER ROLE アクセス許可を持つユーザー

構文

```
ALTER ROLE role [ WITH ]
  { { RENAME TO role } | { OWNER TO user_name } }[, ...]
  [ EXTERNALID TO external_id ]
```

パラメータ

[Role] (ロール)

変更するロールの名前。

RENAME TO

ロールの新しい名前。

OWNER TO user_name

ロールの新しい所有者。

EXTERNALID TO external_id

ID プロバイダーに関連付けられた、ロールの新しい外部 ID。詳細については、「Amazon Redshift のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

例

次の例では、ロールの名前を `sample_role1` から `sample_role2` に変更します。

```
ALTER ROLE sample_role1 WITH RENAME TO sample_role2;
```

次の例では、ロールの所有者を変更します。

```
ALTER ROLE sample_role1 WITH OWNER TO user1
```

ALTER ROLE の構文は、次の ALTER PROCEDURE の構文に類似しています。

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

次の例では、プロシージャの所有者を etl_user に変更します。

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

次の例では、ID プロバイダーに関連付けられている新しい外部 ID を使用してロール sample_role1 を更新します。

```
ALTER ROLE sample_role1 EXTERNALID TO "XYZ456";
```

ALTER PROCEDURE

プロシージャ名または所有者を変更します。プロシージャとデータタイプ (署名) の両方が必要です。所有者またはスーパーユーザーのみがプロシージャ名を変更できます。スーパーユーザーのみがプロシージャの所有者を変更できます。

構文

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    RENAME TO new_name
```

```
ALTER PROCEDURE sp_name [ ( [ [ argname ] [ argmode ] argtype [, ...] ] ) ]
    OWNER TO { new_owner | CURRENT_USER | SESSION_USER }
```

パラメータ

sp_name

変更するプロシージャ名。プロシージャ名のみを現在の検索パスに指定するか、schema_name.sp_procedure_name形式で特定のスキーマを使用します。

[argname] [argmode] argtype

引数の名前、モード、およびデータタイプのリスト。入力データタイプのみが必須です。これは、ストアドプロシージャの識別に使用されます。または、入力パラメータと出力パラメータをモードと共に含めて、プロシージャを作成するために使用する完全な署名を指定することもできます。

new_name

ストアドプロシージャの新しい名前。

new_owner | CURRENT_USER | SESSION_USER

ストアドプロシージャの新しい所有者。

例

次の例では、プロシージャ名を first_quarter_revenue から quarterly_revenue に変更します。

```
ALTER PROCEDURE first_quarter_revenue(volume INOUT bigint, at_price IN numeric,
 result OUT int) RENAME TO quarterly_revenue;
```

この例は以下と同等です。

```
ALTER PROCEDURE first_quarter_revenue(bigint, numeric) RENAME TO quarterly_revenue;
```

次の例では、プロシージャの所有者を etl_user に変更します。

```
ALTER PROCEDURE quarterly_revenue(bigint, numeric) OWNER TO etl_user;
```

ALTER SCHEMA

既存のスキーマの定義を変更します。スキーマ名を変更するかスキーマの所有者を変更するには、このコマンドを使用します。例えば、既存のスキーマの新しいバージョンを作成する予定がある場合、そのスキーマの名前を変更して、そのスキーマのバックアップコピーを保存します。スキーマの詳細については、「CREATE SCHEMA」を参照してください。

構成済みのスキーマクォータを表示するには、「SVV_SCHEMA_QUOTA_STATE」を参照してください。

スキーマクォータを超過したレコードを表示するには、「STL_SCHEMA_QUOTA_VIOLATIONS」を参照してください。

必要な権限

ALTER SCHEMA に必要な権限を以下に示します。

- スーパーユーザー
- ALTER SCHEMA 権限を持つユーザー

- スキーマの所有者

スキーマ名を変更する場合、ストアドプロシージャやマテリアライズドビューなど、古い名前を使用しているオブジェクトは、新しい名前を使用するように更新する必要があることに注意してください。

構文

```
ALTER SCHEMA schema_name
{
RENAME TO new_name |
OWNER TO new_owner |
QUOTA { quota [MB | GB | TB] | UNLIMITED }
}
```

パラメータ

schema_name

変更するデータベーススキーマの名前。

RENAME TO

スキーマの名前を変更する句。

new_name

スキーマの新しい名前。有効な名前の詳細については、「名前と識別子」を参照してください。

OWNER TO

スキーマの所有者を変更する句。

new_owner

スキーマの新しい所有者。

QUOTA

指定したスキーマが使用できる最大ディスク容量。この容量は指定したスキーマのすべてのテーブルを合わせたサイズです。Amazon Redshift は選択した値をメガバイトに変換します。値を指定しない場合、デフォルトの測定単位はギガバイトです。

スキーマクォータ設定の詳細については、「CREATE SCHEMA」を参照してください。

例

次の例では、スキーマの名前を SALES から US_SALES に変更します。

```
alter schema sales
rename to us_sales;
```

次の例では、US_SALES スキーマの所有権をユーザー DWUSER に与えます。

```
alter schema us_sales
owner to dwuser;
```

以下の例では、クォータを 300 GB に変更し、クォータを削除します。

```
alter schema us_sales QUOTA 300 GB;
alter schema us_sales QUOTA UNLIMITED;
```

ALTER SYSTEM

Amazon Redshift クラスターまたは Redshift Serverless ワークグループのシステムレベルの設定オプションを変更します。

必要な権限

次のいずれかのユーザータイプで ALTER SYSTEM コマンドを実行できます。

- スーパーユーザー
- 管理者ユーザー

構文

```
ALTER SYSTEM SET system-level-configuration = {true| t | on | false | f | off}
```

パラメータ

system-level-configuration

システムレベルの設定。有効な値: `data_catalog_auto_mount` および `metadata_security`。

{true| t | on | false | f | off}

システムレベルの設定を有効または無効にする値。true、t、または on は、設定を有効にすることを示します。false、f、または off は、設定を無効にすることを示します。

使用に関する注意事項

プロビジョニングされたクラスターの場合、data_catalog_auto_mount への変更はクラスターの次回の再起動時に有効になります。詳細については、「Amazon Redshift 管理ガイド」の「クラスターの再起動」を参照してください。

サーバーレスワークグループでは、data_catalog_auto_mount への変更はすぐには有効になりません。

例

次の例では、AWS Glue Data Catalog の自動マウントを有効にします。

```
ALTER SYSTEM SET data_catalog_auto_mount = true;
```

次の例では、メタデータのセキュリティを有効にします。

```
ALTER SYSTEM SET metadata_security = true;
```

デフォルトの ID 名前空間の設定

この例は ID プロバイダー専用です。Redshift を IAM アイデンティティセンターおよび ID プロバイダーと統合して、Redshift および AWS の他のサービスの ID 管理を一元化できます。

次のサンプルは、システムのデフォルトの ID 名前空間を設定する方法を示しています。これにより、各 ID のプレフィックスとして名前空間を含める必要がなくなるため、GRANT ステートメントと CREATE ステートメントの実行がより簡単になります。

```
ALTER SYSTEM SET default_identity_namespace = 'MYCO';
```

コマンドを実行したら、次のようなステートメントを実行できます。

```
GRANT SELECT ON TABLE mytable TO alice;

GRANT UPDATE ON TABLE mytable TO salesrole;
```

```
CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

デフォルトの ID 名前空間を設定すると、ID ごとに名前空間をプレフィックスとして指定する必要がなくなります。この例では、alice が MYCO:alice に置き換えられます。これは、含まれているすべての ID で発生します。Redshift での ID プロバイダーの使用の詳細については、「Redshift を IAM アイデンティティセンターに接続してユーザーにシングルサインオンエクスペリエンスを提供する」を参照してください。

IAM アイデンティティセンターでの Redshift 構成に関連する設定の詳細については、「SET」および「他の ID プロバイダー」を参照してください。

ALTER TABLE

このコマンドは、Amazon Redshift テーブルまたは Amazon Redshift Spectrum 外部テーブルの定義を変更します。このコマンドは、CREATE TABLE または CREATE EXTERNAL TABLE によって設定された値とプロパティを更新します。

トランザクションブロック (BEGIN ... END) 内の外部テーブルに対して ALTER TABLE を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

ALTER TABLE は、ALTER TABLE オペレーションを囲むトランザクションが完了するまで、テーブルの読み取りと書き込みのオペレーションをロックします。ただし、データに対するクエリの実行や、変更中のテーブルに対して他の処理の実行が可能であるとドキュメントに明記されている場合を除きます。

必要な権限

テーブルを変更するユーザーがコマンドを正常に実行するには、適切な権限が必要です。ALTER TABLE コマンドによっては、次のいずれかの権限が必要です。

- スーパーユーザー
- ALTER TABLE の権限を持つユーザー
- スキーマに対する USAGE 権限を持つテーブル所有者

構文

```
ALTER TABLE table_name
{
ADD table_constraint
| DROP CONSTRAINT constraint_name [ RESTRICT | CASCADE ]
```

```
| OWNER TO new_owner
| RENAME TO new_name
| RENAME COLUMN column_name TO new_name
| ALTER COLUMN column_name TYPE updated_varchar_data_type_size
| ALTER COLUMN column_name ENCODE new_encode_type
| ALTER COLUMN column_name ENCODE encode_type,
| ALTER COLUMN column_name ENCODE encode_type, .....;
| ALTER DISTKEY column_name
| ALTER DISTSTYLE ALL
| ALTER DISTSTYLE EVEN
| ALTER DISTSTYLE KEY DISTKEY column_name
| ALTER DISTSTYLE AUTO
| ALTER [COMPOUND] SORTKEY ( column_name [,...] )
| ALTER SORTKEY AUTO
| ALTER SORTKEY NONE
| ALTER ENCODE AUTO
| ADD [ COLUMN ] column_name column_type
  [ DEFAULT default_expr ]
  [ ENCODE encoding ]
  [ NOT NULL | NULL ]
  [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ] |
| DROP [ COLUMN ] column_name [ RESTRICT | CASCADE ]
| ROW LEVEL SECURITY { ON | OFF } [ CONJUNCTION TYPE { AND | OR } ] [ FOR DATASHARES ]}

where table_constraint is:

[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] )
| PRIMARY KEY ( column_name [, ... ] )
| FOREIGN KEY (column_name [, ... ] )
   REFERENCES  reftable [ ( refcolumn ) ]}

The following options apply only to external tables:

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
| SET FILE FORMAT format |
| SET TABLE PROPERTIES ('property_name'='property_value')
| PARTITION ( partition_column=partition_value [, ...] )
  SET LOCATION { 's3://bucket/folder' |'s3://bucket/manifest_file' }
| ADD [IF NOT EXISTS]
    PARTITION ( partition_column=partition_value [, ...] ) LOCATION
 { 's3://bucket/folder' |'s3://bucket/manifest_file' }
    [, ... ]
```

```
| DROP PARTITION ( partition_column=partition_value [, ...] )
```

ALTER TABLE コマンドの実行時間を短縮するために、ALTER TABLE コマンドの一部の句を組み合わせることができます。

Amazon Redshift では、ALTER TABLE の句の次の組み合わせがサポートされています。

```
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTKEY column_Id;
ALTER TABLE tablename ALTER DISTKEY column_Id, ALTER SORTKEY (column_list);
ALTER TABLE tablename ALTER SORTKEY (column_list), ALTER DISTSTYLE ALL;
ALTER TABLE tablename ALTER DISTSTYLE ALL, ALTER SORTKEY (column_list);
```

パラメータ

table_name

変更するテーブルの名前 テーブル名のみを指定するか、schema_name.table_name 形式で特定のスキーマを使用します。外部テーブルは、外部スキーマの名前によって修飾される必要があります。また、ALTER TABLE ステートメントを使用してビュー名かビューの所有者を変更する場合、ビュー名を指定することもできます。テーブル名の最大長は 127 バイトです。それより長い名前は 127 バイトまで切り詰められます。最大 4 バイトまで UTF-8 マルチバイト文字を使用できます。有効な名前の詳細については、「名前と識別子」を参照してください。

ADD table_constraint

指定した制約をテーブルに追加する句。有効な table_constraint 値については、「CREATE TABLE」を参照してください。

> **Note**
>
> プライマリキー制約を null が許容された列に追加することはできません。列が元々 NOT NULL 制約で作成された場合、プライマリキー制約を追加できます。

DROP CONSTRAINT constraint_name

テーブルから指名された制約を削除する句。制約を削除するには、制約タイプではなく制約の名前を指定します。テーブルの制約名を表示するには、次のクエリを実行します。

```
select constraint_name, constraint_type
from information_schema.table_constraints;
```

RESTRICT

指定の制約のみを削除する句。RESTRICT は DROP CONSTRAINT のオプションです。RESTRICT を CASCADE と併用することはできません。

CASCADE

指定の制約と同制約に依存するすべてを削除する句。CASCADE は DROP CONSTRAINT のオプションです。CASCADE を RESTRICT と併用することはできません。

OWNER TO new_owner

テーブル (またはビュー) の所有者を new_owner 値に変更する句。

RENAME TO new_name

テーブル (またはビュー) の名前を new_name で指定された値に変更する句。テーブル名の最大長は 127 バイトです。それより長い名前は 127 文字まで切り詰められます。

永続テーブルの名前を「#」で始まる名前に変更することはできません。「#」で始まるテーブル名は、一時テーブルを示します。

外部テーブル名を変更することはできません。

ALTER COLUMN column_name TYPE updated_varchar_data_type_size

VARCHAR データ型と定義されている列のサイズを変更する句。この句は VARCHAR データ型のサイズの変更のみをサポートします。次の制限事項を考慮してください。

- 圧縮エンコード (BYTEDICT、RUNLENGTH、TEXT255、TEXT32K) で列を変更することはできません。
- 既存データの最大サイズよりサイズを小さくすることはできません。
- デフォルト値を含む列は変更できません。
- UNIQUE、PRIMARY KEY、または FOREIGN KEY を含む列は変更できません。
- トランザクションブロック (BEGIN ... END) 内の列を変更することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

ALTER COLUMN column_name ENCODE new_encode_type

列の圧縮エンコードを変更する句。列に圧縮エンコードを指定すると、テーブルは ENCODE AUTO に設定されなくなります。圧縮エンコードに関する詳細は、「保存データのサイズを削減するための列圧縮」を参照してください。

列のための圧縮エンコードを変更した後も、テーブルに対しクエリを実行することが可能です。

次の制限事項を考慮してください。

- 列を現在その列に定義されているものと同じエンコードに変更することはできません。
- インターリーブされたソートキーを使用して、テーブルの列のエンコードを変更することはできません。

ALTER COLUMN column_name ENCODE encode_type, ALTER COLUMN column_name ENCODE encode_type, ...;

複数の列で、圧縮エンコードを単一のコマンドにより変更するための句。圧縮エンコードに関する詳細は、「保存データのサイズを削減するための列圧縮」を参照してください。

列のための圧縮エンコードを変更した後も、テーブルに対しクエリを実行することが可能です。

次の制限事項を考慮してください。

- 1 つのコマンドで、列のエンコーディングを複数回にわたり、同じまたは異なるタイプに変更することはできません。
- 列を現在その列に定義されているものと同じエンコードに変更することはできません。
- インターリーブされたソートキーを使用して、テーブルの列のエンコードを変更することはできません。

ALTER DISTSTYLE ALL

テーブルの既存のディストリビューションスタイルを ALL に変更する句。以下の点を考慮します。

- ALTER DISTSTYLE、ALTER SORTKEY、およびVACUUM は、同じテーブルで同時に実行することはできません。
  - VACUUM を実行中に、ALTER DISTSTYLE ALL を実行すると、エラーが返されます。
  - ALTER DISTKEY が実行されている場合は、テーブルでバックグラウンドバキュームは開始されません。
- ALTER DISTSTYLE ALL コマンドは、インターリーブソートキーおよび一時テーブルを持つテーブルではサポートされません。
- ディストリビューションスタイルが以前に AUTO として定義されていた場合、テーブルは自動テーブル最適化の候補ではなくなります。

DISTSTYLE ALL の詳細については、「CREATE TABLE」を参照してください。

ALTER DISTSTYLE EVEN

テーブルの既存のディストリビューションスタイルを EVEN に変更する句。以下の点を考慮します。

- ALTER DISTSYTLE、ALTER SORTKEY、およびVACUUM は、同じテーブルで同時に実行することはできません。
  - VACUUM を実行中である場合、ALTER DISTSTYLE EVEN を実行すると、エラーが返されます。
  - ALTER DISTKEY EVEN が実行中である場合、テーブルでバックグラウンドバキュームは開始されません。
- ALTER DISTSTYLE EVEN コマンドは、インターリーブされたソートキーを持つテーブルや一時テーブルではサポートされません。
- ディストリビューションスタイルが以前に AUTO として定義されていた場合、テーブルは自動テーブル最適化の候補ではなくなります。

DISTSTYLE EVEN の詳細については、「CREATE TABLE」を参照してください。

ALTER DISTKEY column_name または ALTER DISTSTYLE KEY DISTKEY column_name

テーブルの分散キーとして使用される列を変更する句。以下の点を考慮します。

- 同じテーブルで、VACUUM と ALTER DISTKEY を同時に実行することはできません。
  - VACUUM がすでに実行されている場合は、ALTER DISTKEY よりエラーが返ります。
  - ALTER DISTKEY が実行されている場合は、テーブルでバックグラウンドバキュームは開始されません。
  - ALTER DISTKEY が実行されている場合は、フォアグラウンドバキュームよりエラーが返ります。
- ALTER DISTKEY コマンドは、1 つのテーブルに対して一度に 1 回のみ実行することができます。
- インターリーブソートキーを使用するテーブルについては、ALTER DISTKEY コマンドはサポートされていません。
- ディストリビューションスタイルが以前に AUTO として定義されていた場合、テーブルは自動テーブル最適化の候補ではなくなります。

DISTSTYLE KEY を指定する場合、データは、DISTKEY 列の値で分散されます。DISTSTYLE の詳細については、「CREATE TABLE」を参照してください。

ALTER DISTSTYLE AUTO

テーブルの既存のディストリビューションスタイルを AUTO に変更する句。

ディストリビューションスタイルを AUTO に変更すると、テーブルのディストリビューションスタイルは次のように設定されます。

- DISTSTYLE ALL を持つ小さなテーブルは、AUTO(ALL) に変換されます。
- DISTSTYLE EVEN を持つ小さなテーブルは、AUTO(ALL) に変換されます。
- DISTSTYLE KEY を持つ小さなテーブルは、AUTO(ALL) に変換されます。
- DISTSTYLE ALL を持つ大きなテーブルは、AUTO(EVEN) に変換されます。
- DISTSTYLE EVEN を持つ大きなテーブルは、AUTO(EVEN) に変換されます。
- DISTSTYLE KEY を持つ大きなテーブルは、AUTO(KEY) に変換され、DISTKEY は保持されます。この場合、Amazon Redshift はテーブルに変更を加えません。

Amazon Redshift が、新しいディストリビューションスタイルまたはキーによってクエリのパフォーマンスが向上すると判断した場合、Amazon Redshift は、将来、テーブルのディストリビューションスタイルまたはキーを変更する可能性があります。例えば、Amazon Redshift は、DISTSTYLE が AUTO (KEY) のテーブルを AUTO(EVEN) に、またはその逆に変更する場合があります。データの再配布やロックなど、配布キーが変更されたときの動作の詳細については、「Amazon Redshift Advisor の推奨事項」を参照してください。

DISTSTYLE AUTO の詳細については、「CREATE TABLE」を参照してください。

テーブルのディストリビューションスタイルを表示するには、SVV_TABLE_INFO システムカタログビューに対してクエリを実行します。詳細については、「SVV_TABLE_INFO」を参照してください。テーブルの Amazon Redshift Advisor のレコメンデーションを表示するには、SVV_ALTER_TABLE_RECOMMENDATIONS システムカタログビューをクエリします。詳細については、「SVV_ALTER_TABLE_RECOMMENDATIONS」を参照してください。Amazon Redshift が実行したアクションを表示するには、SVL_AUTO_WORKER_ACTION システムカタログビューにクエリを実行します。詳細については、「SVL_AUTO_WORKER_ACTION」を参照してください。

ALTER [COMPOUND] SORTKEY (column_name [,...])

テーブルで使用されるソートキーを変更または追加する句。ALTER SORTKEY は一時テーブルではサポートされていません。

ソートキーを変更すると、新しいソートキーまたは元のソートキーの列の圧縮エンコードが変更される場合があります。テーブルにエンコードが明示的に定義されていない場合、Amazon Redshift は、次のように圧縮エンコードを自動的に割り当てます。

- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、または DOUBLE PRECISION データ型として定義されている列には、RAW 圧縮が割り当てられます。

- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ として定義された列には AZ64 圧縮が割り当てられます。
- CHAR または VARCHAR として定義された列には、LZO 圧縮が割り当てられます。

以下の点を考慮します。

- テーブルあたりのソートキーには最大 400 列を定義できます。
- インターリーブソートキーは、複合ソートキーに変更することが可能です。あるいは、ソートキーなしに変更することもできます。ただし、複合ソートキーをインターリーブソートキーに変更することはできません。
- ソートキーが以前に AUTO として定義されていた場合、テーブルは自動テーブル最適化の候補ではなくなります。
- Amazon Redshift では、ソートキーとして定義された列に RAW エンコード (圧縮なし) を使用することをお勧めします。列を変更してソートキーとして選択すると、列の圧縮が RAW 圧縮 (圧縮なし) に変更されます。これにより、テーブルに必要なストレージ量が増加する可能性があります。テーブルのサイズがどれだけ増加するかは、特定のテーブル定義とテーブルの内容によって異なります。圧縮の詳細については、「圧縮エンコード」を参照してください。

データがテーブルに読み込まれる際、データはソートキーの順序で読み込まれます。ソートキーが変更されると、Amazon Redshift によってデータの順序が変更されます。SORTKEY の詳細については、「CREATE TABLE」を参照してください。

ALTER SORTKEY AUTO

ターゲットテーブルのソートキーを AUTO に変更または追加する句。ALTER SORTKEY AUTO は一時テーブルではサポートされていません。

ソートキーを AUTO に変更すると、Amazon Redshift はテーブルの既存のソートキーを保持します。

Amazon Redshift が新しいソートキーによってクエリのパフォーマンスが向上すると判断した場合、Amazon Redshift は今後、テーブルのソートキーを変更する可能性があります。

SORTKEY AUTO の詳細については、「CREATE TABLE」を参照してください。

テーブルのソートキーを表示するには、SVV_TABLE_INFO システムカタログビューに対してクエリを実行します。詳細については、「SVV_TABLE_INFO」を参照してください。テーブルの Amazon Redshift Advisor のレコメンデーションを表示するには、SVV_ALTER_TABLE_RECOMMENDATIONS システムカタログビューをクエリします。詳細については、「SVV_ALTER_TABLE_RECOMMENDATIONS」を参照してください。Amazon

Redshift が実行したアクションを表示するには、SVL_AUTO_WORKER_ACTION システムカタログビューにクエリを実行します。詳細については、「SVL_AUTO_WORKER_ACTION」を参照してください。

ALTER SORTKEY NONE

ターゲットテーブルのソートキーを削除する句。

ソートキーが以前に AUTO として定義されていた場合、テーブルは自動テーブル最適化の候補ではなくなります。

ALTER ENCODE AUTO

ターゲットテーブルの列のエンコードタイプを AUTO に変更する句。エンコードを AUTO に変更すると、Amazon Redshift はテーブル内の列の既存のエンコードタイプを保持します。その後、Amazon Redshift が新しいエンコードタイプによってクエリのパフォーマンスが向上すると判断した場合、Amazon Redshift はテーブル列のエンコードタイプを変更できます。

1 つ以上の列を変更してエンコードを指定した場合、Amazon Redshift はテーブル内のすべての列のエンコードを自動的に調整しなくなりました。列は現在のエンコード設定を保持します。

次のアクションは、テーブルの ENCODE AUTO 設定には影響しません。

- テーブルの名前を変更します。
- テーブルの DISTSTYLE または SORTKEY 設定を変更します。
- ENCODE 設定で列を追加または削除します。
- COPY コマンドの COMPUPDATE オプションを使用します。詳細については、「データのロード操作」を参照してください。

テーブルのエンコードを表示するには、SVV_TABLE_INFO システムカタログビューにクエリを実行します。詳細については、「SVV_TABLE_INFO」を参照してください。

RENAME COLUMN column_name TO new_name

列の名前を new_name で指定された値に変更する句。列名の最大長は 127 バイトです。それより長い名前は 127 文字まで切り詰められます。有効な名前の詳細については、「名前と識別子」を参照してください。

ADD [COLUMN] column_name

指定した名前を持つ列をテーブルに追加する句。各 ALTER TABLE ステートメントでは 1 列しか追加できません。

テーブルの分散キー (DISTKEY) またはソートキー (SORTKEY) である列は追加できません。

ALTER TABLE ADD COLUMN コマンドを使用して次のテーブルと列の属性を変更することはできません。

- UNIQUE
- PRIMARY KEY
- REFERENCES (外部キー)
- IDENTITY または GENERATED BY DEFAULT AS IDENTITY

列名の最大長は 127 バイトです。それより長い名前は 127 文字まで切り詰められます。1 つのテーブルで定義できる列の最大数は 1,600 です。

外部テーブルに列を追加する場合、次の制限が適用されます。

- DEFAULT、ENCODE、NOT NULL または NULL を制約する列がある外部テーブルに列を追加することはできません。
- AVRO ファイル形式を使用して定義した外部テーブルに列を追加することはできません。
- 擬似列が有効になっている場合、1 つの外部テーブルで定義できる列の最大数は 1,598 です。擬似列が有効でない場合、1 つのテーブルで定義できる列の最大数は 1,600 です。

詳細については、「CREATE EXTERNAL TABLE」を参照してください。

column_type

追加する列のデータ型。CHAR および VARCHAR の列の場合、最大長を宣言する代わりに MAX キーワードを使用できます。MAX は、最大長を CHAR では 4096 バイト、VARCHAR では 65535 バイトに設定します。GEOMETRY オブジェクトの最大サイズは 1,048,447 バイトです。

Amazon Redshift でサポートされているデータ型の詳細については、「データ型」を参照してください。

DEFAULT default_expr

列のデフォルトのデータ値を割り当てる句。default_expr のデータ型は列のデータ型に一致する必要があります。DEFAULT 値は、変数を使用しない式にする必要があります。サブクエリ、現在のテーブルに含まれる他の列の相互参照、およびユーザー定義の関数は使用できません。

default_expr は、列の値を指定しないすべての INSERT 操作で使用されます。デフォルト値を指定しなかった場合、列のデフォルト値は null です。

COPY 操作により、DEFAULT 値と NOT NULL 制約が設定された列で null フィールドが見つかった場合、COPY コマンドは default_expr の値を挿入します。

DEFAULT は外部テーブルでサポートされていません。

ENCODE encoding

列の圧縮エンコード。デフォルトでは、テーブル内のどの列にも圧縮エンコードを指定しない場合、またはテーブルに ENCODE AUTO オプションを指定した場合、Amazon Redshift はテーブル内のすべての列の圧縮エンコードを自動的に管理します。

テーブルで任意の列に圧縮エンコードを指定する場合、またはテーブルに ENCODE AUTO オプションを指定しない場合、Amazon Redshift は次のように圧縮エンコードを指定しない列に圧縮エンコードを自動的に割り当てます。

- 一時テーブルのすべての列には RAW 圧縮がデフォルトで割り当てられます。
- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY、もしくは GEOGRAPHY の各データ型で定義されている列には、RAW 圧縮が割り当てられます。
- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ として定義された列には AZ64 圧縮が割り当てられます。
- CHAR、VARCHAR、または VARBYTE として定義されている列には、LZO 圧縮が割り当てられます。

> **Note**
>
> 列を圧縮しない場合は、明示的に RAW エンコードを指定します。

次の compression encodings (p. 65) がサポートされています。

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO
- MOSTLY8

- MOSTLY16
- MOSTLY32
- RAW (非圧縮)
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

ENCODE は外部テーブルでサポートされていません。

NOT NULL | NULL

NOT NULL は、列に null 値を使用できないことを指定します。NULL はデフォルトであり、列で null 値を使用できることを指定します。

NOT NULL と NULL は外部テーブルでサポートされていません。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

列での文字列検索または比較が、CASE_SENSITIVE か CASE_INSENSITIVE かを指定する句。デフォルト値は、大文字と小文字を区別する、現行のデータベースの設定と同じです。

データベース照合に関する情報を検索するには、次のコマンドを使用します。

```
SELECT db_collation();

db_collation
----------------
 case_sensitive
(1 row)
```

DROP [COLUMN] column_name

テーブルから削除する列の名前。

テーブルの末尾列は削除できません。テーブルには少なくとも 1 つの列が必要です。

テーブルの分散キー (DISTKEY) またはソートキー (SORTKEY) である列は削除できません。ビュー、プライマリキー、外部キー、UNIQUE 制約などの依存オブジェクトが列にある場合、DROP COLUMN のデフォルト動作は RESTRICT です。

外部テーブルから列を削除する場合は、次の制限が適用されます。

- 列をパーティションとして使用している場合、外部テーブルから列を削除することはできません。
- AVRO ファイル形式を使用して定義した外部テーブルから列を削除することはできません。
- RESTRICT と CASCADE は外部テーブルに無視されます。
- ポリシーを削除するかデタッチしない限り、ポリシー定義内で参照されるポリシーテーブルの列を削除することはできません。これは、CASCADE オプションが指定されている場合にも当てはまります。ポリシーテーブル内の他の列を削除できます。

詳細については、「CREATE EXTERNAL TABLE」を参照してください。

RESTRICT

RESTRICT を DROP COLUMN とともに使用すると、以下の場合に、ドロップされる列がドロップされません。

- 定義されたビューがドロップされる列を参照している場合
- 外部キーが列を参照している場合
- 列がマルチパートキーに属している場合

RESTRICT を CASCADE と併用することはできません。

RESTRICT と CASCADE は外部テーブルに無視されます。

CASCADE

DROP COLUMN と共に使用すると、指定した列およびその列に依存するすべてのものを削除します。CASCADE を RESTRICT と併用することはできません。

RESTRICT と CASCADE は外部テーブルに無視されます。

以下のオプションは、外部テーブルにのみ適用されます。

SET LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }

データファイルを含む Amazon S3 フォルダ、または Amazon S3 オブジェクトパスのリストを含むマニフェストファイルへのパス。バケットは、Amazon Redshift クラスターと同じ AWS リージョン内に置かれている必要があります。サポートされている AWS リージョンの一覧は、「Amazon Redshift Spectrum の制限事項」でご確認ください。マニフェストファイルの使用に関

する詳細は、CREATE EXTERNAL TABLE パラメータ リファレンスの LOCATION を参照してください。

SET FILE FORMAT format

外部データファイルのファイル形式。

有効な形式は次のとおりです。

- AVRO
- PARQUET
- RCFILE
- SEQUENCEFILE
- TEXTFILE

SET TABLE PROPERTIES ('property_name'='property_value')

外部テーブルのテーブルプロパティのテーブル定義を設定する句。

> **Note**
> テーブルのプロパティでは、大文字と小文字が区別されます。

'numRows'='row_count'

テーブル定義の numRows 値を設定するプロパティ。外部テーブルの統計を明示的に更新するには、テーブルのサイズを示す numRows プロパティを設定します。Amazon Redshift は、外部テーブルを分析して、クエリオプティマイザがクエリプランを生成するために使用するテーブル統計を生成することはありません。外部テーブルに対してテーブル統計が設定されていない場合、Amazon Redshift はクエリ実行プランを生成します。このプランは、外部テーブルの方が大きくローカルテーブルの方が小さいという前提に基づきます。

'skip.header.line.count'='line_count'

各ソースファイルの最初に省略する行数を設定するプロパティ。

PARTITION (partition_column=partition_value [, ...] SET LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' }

1 つ以上のパーティション列の新しい場所を設定する句。

ADD [IF NOT EXISTS] PARTITION (partition_column=partition_value [, ...]) LOCATION { 's3://bucket/folder' | 's3://bucket/manifest_file' } [, ...]

1 つ以上のパーティションを追加する句。複数の PARTITION 句を指定するには、単一の ALTER TABLE … ADD ステートメントを使用します。

> **Note**
>
> AWS Glue を使用する場合、単一の ALTER TABLE ステートメントを使用して、最大 100 パーティションまで追加できます。

IF NOT EXISTS 句は、指定されたパーティションが既に存在する場合はコマンドが変更を加えないことを示します。また、コマンドが、エラーで終了するのではなく、パーティションが存在することを示すメッセージを返すことも示します。この句は、ALTER TABLE で既存のパーティションを追加しようとしてもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

DROP PARTITION (partition_column=partition_value [, ...])

指定のパーティションを削除する句。パーティションを削除すると、外部テーブルのメタデータのみが変化します。Amazon S3 のデータは影響を受けません。

ROW LEVEL SECURITY { ON | OFF } [CONJUNCTION TYPE { AND | OR }] [FOR DATASHARES]

リレーションの行レベルのセキュリティをオンまたはオフにする句。

リレーションで行レベルのセキュリティがオンになっている場合、行レベルのセキュリティポリシーでアクセスが許可されている行のみを読み取ることができます。リレーションへのアクセス権を付与するポリシーがない場合は、リレーションから行を表示できません。ROW LEVEL SECURITY 句を設定できるのは、スーパーユーザーと、`sys:secadmin` ロールを持つユーザーまたはロールのみです。詳細については、「行レベルのセキュリティ」を参照してください。

- [CONJUNCTION TYPE { AND | OR }]

  リレーションの行レベルのセキュリティポリシーの結合タイプを選択できる句。1 つのリレーションに複数の行レベルのセキュリティポリシーがアタッチされている場合、それらのポリシーを AND 句や OR 句で組み合わせることができます。デフォルトでは、Amazon Redshift は RLS ポリシーを AND 句で組み合わせます。スーパーユーザーと、`sys:secadmin` ロールを持つユーザーまたはロールは、この句を使用して、リレーションの行レベルのセキュリティポリシーの結合タイプを定義できます。詳細については、「ユーザーごとに複数ポリシーの組み合わせ」を参照してください。

- FOR DATASHARES

  RLS で保護されたリレーションにデータ共有上でアクセスできるかどうかを決定する句。デフォルトでは、RLS で保護されたリレーションにデータ共有経由でアクセスすることはできません。この句を指定して実行される ALTER TABLE ROW LEVEL SECURITY コマンドは、リレーションのデータ共有アクセシビリティプロパティにのみ影響します。ROW LEVEL SECURITY のプロパティは変更されません。

  RLS で保護されたリレーションにデータ共有上でアクセスできるようにした場合、コンシューマー側のデータ共有データベースにおいて、そのリレーションには行レベルのセキュリティが適用されません。リレーションはプロデューサー側の RLS プロパティを保持します。

例

ALTER TABLE コマンドの使用方法を示す例については、以下を参照してください。

- ALTER TABLE の例
- ALTER EXTERNAL TABLE 例
- ALTER TABLE ADD および DROP COLUMN の例

ALTER TABLE の例

次の例は、ALTER TABLE コマンドの基本用法を示しています。

テーブルまたはビューの名前の変更

次のコマンドは、USERS テーブルを USERS_BKUP に名前変更します。

```
alter table users
rename to users_bkup;
```

このタイプのコマンドを使用して、ビューの名前を変更することもできます。

テーブルまたはビューの所有者の変更

次のコマンドは VENUE テーブルの所有者をユーザー DWUSER に変更します。

```
alter table venue
owner to dwuser;
```

次のコマンドは、ビューを作成した後、その所有者を変更します。

```
create view vdate as select * from date;
alter table vdate owner to vuser;
```

列名の変更

次のコマンドは VENUE テーブルの列名を VENUESEATS から VENUESIZE に変更します。

```
alter table venue
rename column venueseats to venuesize;
```

テーブルの制約を削除する

プライマリキー、外部キーまたは一意の制約のようなテーブルの制約を削除するには、まず内部の制約名を見つけます。その後、ALTER TABLE コマンドで制約名を指定します。次の例では、CATEGORY テーブルの制約を見つけてから、`category_pkey`という名前のプライマリキーを削除します。

```
select constraint_name, constraint_type
from information_schema.table_constraints
where constraint_schema ='public'
and table_name = 'category';

constraint_name | constraint_type
----------------+----------------
category_pkey   | PRIMARY KEY

alter table category
drop constraint category_pkey;
```

VARCHAR 列の変更

ストレージを節約するため、最初に現在のデータ要件に必要な最小サイズの VARCHAR 列を使用してテーブルを定義することができます。後でより長い文字列を収容するには、テーブルを変更して列のサイズを大きくすることができます。

次の例では、EVENTNAME 列のサイズを VARCHAR(300) に増やします。

```
alter table event alter column eventname type varchar(300);
```

列の圧縮エンコードを変更する

列の圧縮エンコードは変更できます。以下に、このアプローチを示す一連の例を示します。これらの例のテーブルは次のように定義されています。

```
create table t1(c0 int encode lzo, c1 bigint encode zstd, c2 varchar(16) encode lzo, c3
 varchar(32) encode zstd);
```

次のステートメントは、列 c0 の圧縮エンコードを LZO エンコードから AZ64 エンコードに変更します。

```
alter table t1 alter column c0 encode az64;
```

次のステートメントは、列 c1 の圧縮エンコードを Zstandard エンコードから AZ64 エンコードに変更します。

```
alter table t1 alter column c1 encode az64;
```

次のステートメントは、列 c2 の圧縮エンコードを LZO エンコードからバイトディクショナリエンコードに変更します。

```
alter table t1 alter column c2 encode bytedict;
```

次のステートメントは、列 c3 の圧縮エンコードを Zstandard エンコードから Runlength エンコードに変更します。

```
alter table t1 alter column c3 encode runlength;
```

DISTSTYLE KEY DISTKEY 列の変更

次の例は、テーブルの DISTSTYLE および DISTKEY を変更する方法を示しています。

EVEN 分散スタイルを指定してテーブルを作成します。SVV_TABLE_INFO ビューは、DISTSTYLE が EVEN であることを示しています。

```
create table inventory(
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
```

```
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+----------------
 inventory |     EVEN
```

DISTKEY テーブルを inv_warehouse_sk に変更してください。SVV_TABLE_INFO ビューは、inv_warehouse_sk列を結果の分散キーとして示しています。

```
alter table inventory alter diststyle key distkey inv_warehouse_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+-----------------------
 inventory | KEY(inv_warehouse_sk)
```

DISTKEY テーブルを inv_item_sk に変更してください。SVV_TABLE_INFO ビューは、inv_item_sk列を結果の分散キーとして示しています。

```
alter table inventory alter distkey inv_item_sk;

select "table", "diststyle" from svv_table_info;

   table   |       diststyle
-----------+-----------------------
 inventory | KEY(inv_item_sk)
```

テーブルを DISTSTYLE ALL に変更する

次の例では、テーブルを DISTSTYLE ALL に変更する方法を説明します。

EVEN 分散スタイルを指定してテーブルを作成します。SVV_TABLE_INFO ビューは、DISTSTYLE が EVEN であることを示しています。

```
create table inventory(
```

```
  inv_date_sk int4 not null ,
  inv_item_sk int4 not null ,
  inv_warehouse_sk int4 not null ,
  inv_quantity_on_hand int4
) diststyle even;

Insert into inventory values(1,1,1,1);

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+----------------
 inventory |     EVEN
```

DISTSTYLE テーブルを ALL に変更します。SVV_TABLE_INFO ビューには、変更された DISTSYTLE が表示されます。

```
alter table inventory alter diststyle all;

select "table", "diststyle" from svv_table_info;

   table   |   diststyle
-----------+----------------
 inventory |     ALL
```

テーブルの SORTKEY を変更する

複合ソートキーを使用する、あるいはソートキーを使用しないようにテーブルを変更できます。

次のテーブル定義では、テーブル `t1` は、インターリーブソートキーを使用するように定義されます。

```
create table t1 (c0 int, c1 int) interleaved sortkey(c0, c1);
```

次のコマンドは、インターリーブされたソートキーの使用から複合ソートキーの使用に、テーブルを変更します。

```
alter table t1 alter sortkey(c0, c1);
```

次のコマンドは、インターリーブソートキーを削除するためにテーブルを変更します。

```
alter table t1 alter sortkey none;
```

次のテーブル定義では、テーブル t1 のソートキーとして c0 列を定義しています。

```
create table t1 (c0 int, c1 int) sortkey(c0);
```

次のコマンドは、テーブル t1 を複合ソートキーの使用に変更します。

```
alter table t1 alter sortkey(c0, c1);
```

テーブルを ENCODE AUTO に変更する

次の例は、テーブルを ENCODE AUTO に変更する方法を示しています。

例えば、テーブルは次のように定義されています。列 c0 はエンコードタイプ AZ64 で定義され、列 c1 はエンコードタイプ LZO で定義されます。

```
create table t1(c0 int encode AZ64, c1 varchar encode LZO);
```

このテーブルの場合、次のステートメントはエンコードを AUTO に変更します。

```
alter table t1 alter encode auto;
```

次の例は、テーブルを変更して ENCODE AUTO 設定を削除する方法を示しています。

例えば、テーブルは次のように定義されています。テーブルの列は、エンコードなしで定義されます。この場合、エンコードはデフォルトで ENCODE AUTO になります。

```
create table t2(c0 int, c1 varchar);
```

このテーブルの場合、次のステートメントは列 c0 のエンコードを LZO に変更します。テーブルのエンコードが ENCODE AUTO に設定されなくなりました。

```
alter table t2 alter column c0 encode lzo;;
```

行レベルのセキュリティコントロールの変更

次のコマンドは、テーブルの RLS をオフにします。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY OFF;
```

次のコマンドは、テーブルの RLS をオンにします。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
```

次のコマンドは、テーブルの RLS をオンにし、データ共有上でテーブルにアクセスできるようにします。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES OFF;
```

次のコマンドは、テーブルの RLS をオンにして、データ共有上でテーブルにアクセスできなくします。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON;
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY FOR DATASHARES ON;
```

次のコマンドは、RLS をオンにし、テーブルの RLS 結合タイプを OR に設定します。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE OR;
```

次のコマンドは、RLS をオンにし、テーブルの RLS 結合タイプを AND に設定します。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON CONJUNCTION TYPE AND;
```

ALTER EXTERNAL TABLE 例

次の例では、米国東部 (バージニア北部) リージョン (`us-east-1`) AWS リージョン にある Amazon S3 バケットおよび CREATE TABLE の [例](#) で作成されたテーブル例を使用します。外部テーブルでパーティションを使用する方法の詳細については、「[Redshift Spectrum 外部テーブルのパーティション化](#)」を参照してください。

次の例では、SPECTRUM.SALES 外部テーブルの numRows テーブルプロパティを 170,000 行に設定します。

```
alter table spectrum.sales
```

```
set table properties ('numRows'='170000');
```

次の例では、SPECTRUM.SALES 外部テーブルの場所を変更します。

```
alter table spectrum.sales
set location 's3://redshift-downloads/tickit/spectrum/sales/';
```

次の例では、SPECTRUM.SALES 外部テーブルの型式を Parquet に変更します。

```
alter table spectrum.sales
set file format parquet;
```

次の例では、SPECTRUM.SALES_PART テーブルに対して 1 つのパーティションを追加します。

```
alter table spectrum.sales_part
add if not exists partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
```

次の例では、SPECTRUM.SALES_PART テーブルに対して 3 つのパーティションを追加します。

```
alter table spectrum.sales_part add if not exists
partition(saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/'
partition(saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/'
partition(saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
```

次の例では、SPECTRUM.SALES_PART を変更して `saledate='2008-01-01''` のパーティションを削除します。

```
alter table spectrum.sales_part
drop partition(saledate='2008-01-01');
```

次の例では、`saledate='2008-01-01'` のパーティションに新しい Amazon S3 パスを設定します。

```
alter table spectrum.sales_part
```

```
partition(saledate='2008-01-01')
set location 's3://redshift-downloads/tickit/spectrum/sales_partition/
saledate=2008-01-01/';
```

次の例では、sales_dateの名前を transaction_date に変更します。

```
alter table spectrum.sales rename column sales_date to transaction_date;
```

次の例では、最適化された行列 (ORC) 形式を使用する外部テーブルのために、列マッピングを位置マッピングに設定します。

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='position');
```

次の例では、最適化された ORC 形式を使用する外部テーブルのために、列マッピングを名前マッピングに設定します。

```
alter table spectrum.orc_example
set table properties('orc.schema.resolution'='name');
```

ALTER TABLE ADD および DROP COLUMN の例

次の例は、ALTER TABLE を使用して基本テーブル列を追加した後に削除する方法、および依存オブジェクトがある列を削除する方法を示しています。

基本列の追加と削除

次の例では、スタンドアロンの FEEDBACK_SCORE 列を USERS テーブルに追加します。この列には 1 つの整数が含まれます。この列のデフォルト値は NULL です (フィードバックスコアなし)。

まず、PG_TABLE_DEF カタログテーブルにクエリして、USERS テーブルのスキーマを表示します。

```
column         | type                   | encoding | distkey | sortkey
---------------+------------------------+----------+---------+--------
userid         | integer                | delta    | true    |       1
username       | character(8)           | lzo      | false   |       0
firstname      | character varying(30)  | text32k  | false   |       0
lastname       | character varying(30)  | text32k  | false   |       0
```

```
city          | character varying(30)  | text32k  | false    |       0
state         | character(2)           | bytedict | false    |       0
email         | character varying(100) | lzo      | false    |       0
phone         | character(14)          | lzo      | false    |       0
likesports    | boolean                | none     | false    |       0
liketheatre   | boolean                | none     | false    |       0
likeconcerts  | boolean                | none     | false    |       0
likejazz      | boolean                | none     | false    |       0
likeclassical | boolean                | none     | false    |       0
likeopera     | boolean                | none     | false    |       0
likerock      | boolean                | none     | false    |       0
likevegas     | boolean                | none     | false    |       0
likebroadway  | boolean                | none     | false    |       0
likemusicals  | boolean                | none     | false    |       0
```

次に、feedback_score 列を追加します。

```
alter table users
add column feedback_score int
default NULL;
```

USERS から FEEDBACK_SCORE 列を選択し、追加されたことを確認します。

```
select feedback_score from users limit 5;

feedback_score
----------------
NULL
NULL
NULL
NULL
NULL
```

列を削除して元の DDL に戻します。

```
alter table users drop column feedback_score;
```

依存オブジェクトがある列の削除

以下の例では、依存オブジェクトがある列を削除します。その結果、依存オブジェクトも削除されます。

初めに、もう一度 FEEDBACK_SCORE 列を USERS テーブルに追加します。

```
alter table users
add column feedback_score int
default NULL;
```

次に、USERS テーブルから USERS_VIEW というビューを作成します。

```
create view users_view as select * from users;
```

次に、USERS テーブルから FEEDBACK_SCORE 列の削除を試みます。この DROP ステートメントではデフォルト動作 (RESTRICT) を使用します。

```
alter table users drop column feedback_score;
```

Amazon Redshift は列に別のオブジェクトが依存しているため、列を削除できないことを示すエラーメッセージが表示されます。

今度はすべての依存オブジェクトを削除するため CASCADE を指定して、FEEDBACK_SCORE 列の削除を再度試みます。

```
alter table users
drop column feedback_score cascade;
```

ALTER TABLE APPEND

既存のソーステーブルのデータを移動して、ターゲットテーブルに行を追加します。ソーステーブル内のデータはターゲットテーブルの一致する列に移動されます。列の順序は関係ありません。データがターゲットテーブルに正常に追加されると、ソーステーブルは空になります。通常、ALTER TABLE APPEND は、データを複製するのではなく移動するため、同様の CREATE TABLE AS または INSERT の INTO オペレーションよりもはるかに高速です。

> **Note**
>
> ALTER TABLE APPEND は、ソーステーブルとターゲットテーブル間でデータブロックを移動します。パフォーマンスを向上させるため、ALTER TABLE APPEND は追加操作の一部としてストレージを圧縮しません。その結果、ストレージ使用量は一時的に増加します。スペースを回復するには、VACUUM操作を実行します。

同じ名前の列には、同じ列属性も必要です。ソーステーブルまたはターゲットテーブルに、他のテーブルに含まれる列が含まれない場合は、IGNOREEXTRA または FILLTARGET パラメータを使用して、追加の列を管理する方法を指定します。

IDENTITY 列を追加することはできません。テーブルの両方に IDENTITY 列が含まれる場合、コマンドは失敗します。1 つのテーブルのみに IDENTITY 列がある場合は、FILLTARGET パラメータまたは IGNOREEXTRA パラメータを含めます。詳細については、「ALTER TABLE APPEND の使用に関する注意事項」を参照してください。

GENERATED BY DEFAULT AS IDENTITY 列を追加できます。GENERATED BY DEFAULT AS IDENTITY として定義された列を、指定した値で更新できます。詳細については、「ALTER TABLE APPEND の使用に関する注意事項」を参照してください。

ターゲットテーブルは永続テーブルである必要があります。ただし、ソースはパーマネントテーブルでも、ストリーミング取り込み用に設定されたマテリアライズドビューでもかまいません。両方のオブジェクトは同じ分散スタイルと分散キーを使用する必要があります (定義されている場合)。オブジェクトがソートされている場合、両方のオブジェクトは同じソート形式を使用し、ソートキーとして同じ列を定義する必要があります。

ALTER TABLE APPEND コマンドは、オペレーションが完了するとすぐに自動的にコミットします。ロールバックすることはできません。トランザクションブロック (BEGIN ... END) 内で ALTER TABLE APPEND を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

必要な権限

ALTER TABLE APPEND コマンドによっては、次のいずれかの権限が必要です。

- スーパーユーザー
- ALTER TABLE システム権限を持つユーザー
- ソーステーブルに対する DELETE 権限と SELECT 権限、およびターゲットテーブルに対する INSERT 権限を持つユーザー

構文

```
ALTER TABLE target_table_name APPEND FROM [ source_table_name
 | source_materialized_view_name ]
[ IGNOREEXTRA | FILLTARGET ]
```

マテリアライズドビューからの追加は、マテリアライズドビューが [マテリアライズドビューへのストリーミング取り込み](#) に設定されている場合にのみ機能します。

パラメータ

target_table_name

列が追加されるテーブルの名前。テーブル名のみを指定するか、schema_name.table_name 形式で特定のスキーマを使用します。ターゲットテーブルは既存の永続テーブルである必要があります。

FROM source_table_name

追加する行を提供するテーブルの名前。テーブル名のみを指定するか、schema_name.table_name 形式で特定のスキーマを使用します。ソーステーブルは既存の永続テーブルである必要があります。

source_materialized_view_name から

追加する行を提供するマテリアライズドビューの名前。マテリアライズドビューからの追加は、マテリアライズドビューが [マテリアライズドビューへのストリーミング取り込み](#) に設定されている場合にのみ機能します。ソースマテリアライズドビューは既に存在している必要があります。

IGNOREEXTRA

ターゲットテーブルにない列がソーステーブルに含まれている場合に、追加の列のデータを破棄することを指定するキーワード。IGNOREEXTRA を FILLTARGET と使用することはできません。

FILLTARGET

ソーステーブルにない列がターゲットテーブルに含まれている場合に、列に [DEFAULT](#) 列値を入れるか (定義されている場合)、NULL にすることを指定するキーワード。IGNOREEXTRA を FILLTARGET と使用することはできません。

ALTER TABLE APPEND の使用に関する注意事項

ALTER TABLE APPEND はソーステーブルからターゲットテーブルに同一の列のみを移動します。列の順序は関係ありません。

ソーステーブルまたはターゲットテーブルに追加の列が含まれている場合、以下のルールに従って FILLTARGET または IGNOREEXTRA を使用します。

- ターゲットテーブルに存在しない列がソーステーブルに含まれている場合は、IGNOREEXTRA を含めます。コマンドは、ソーステーブルの追加の列を無視します。
- ソーステーブルに存在しない列がターゲットテーブルに含まれている場合は、FILLTARGET を含めます。コマンドは、デフォルトの列値または IDENTITY 値 (定義されている場合)、あるいは NULL をターゲットテーブルの追加の列に入れます。
- ソーステーブルとターゲットテーブルの両方に追加の列が含まれる場合、コマンドは失敗します。FILLTARGET と IGNOREEXTRA の両方を使用することはできません。

同じ名前で別の属性を持つ列が両方のテーブルに存在する場合、コマンドは失敗します。同様の名前の列には、共通して次の属性が必要です。

- データ型
- 列のサイズ
- 圧縮エンコード
- null でない
- ソート形式
- ソートキーの列
- 分散スタイル
- 分散キー列

IDENTITY 列を追加することはできません。ソーステーブルとターゲットテーブルの両方に IDENTITY 列がある場合、コマンドは失敗します。ソーステーブルのみに IDENTITY 列がある場合、IDENTITY 列が無視されるように IGNOREEXTRA パラメータを含めます。ターゲットテーブルのみに IDENTITY 列がある場合は、テーブルに対して定義されている IDENTITY 句に従って IDENTITY 列が入力されるように FILLTARGET パラメータを含めます。詳細については、「DEFAULT」を参照してください。

ALTER TABLE APPEND ステートメントを使用して、デフォルトの IDENTITY 列を追加できます。詳細については、「CREATE TABLE」を参照してください。

ALTER TABLE APPEND の例

最新の売上を把握するために、組織で SALES_MONTHLY テーブルを維持しているとします。毎月、トランザクションテーブルから SALES テーブルにデータを移動するとします。

次の INSERT INTO または TRUNCATE コマンドを使用して、このタスクを完了できます。

```
insert into sales (select * from sales_monthly);
truncate sales_monthly;
```

ただし、ALTER TABLE APPEND コマンドを使用すると、同じオペレーションをはるかに効率的に実行できます。

最初に、[PG_TABLE_DEF](#)システムカタログテーブルにクエリを実行し、両方のテーブルに、同じ列属性を持つ同じ列があることを確認します。

```
select trim(tablename) as table, "column", trim(type) as type,
encoding, distkey, sortkey, "notnull"
from pg_table_def where tablename like 'sales%';

table       | column     | type                        | encoding | distkey | sortkey |
 notnull
-----------+------------+-----------------------------+----------+---------+---------
+--------
sales       | salesid    | integer                     | lzo      | false   |       0 |
 true
sales       | listid     | integer                     | none     | true    |       1 |
 true
sales       | sellerid   | integer                     | none     | false   |       2 |
 true
sales       | buyerid    | integer                     | lzo      | false   |       0 |
 true
sales       | eventid    | integer                     | mostly16 | false   |       0 |
 true
sales       | dateid     | smallint                    | lzo      | false   |       0 |
 true
sales       | qtysold    | smallint                    | mostly8  | false   |       0 |
 true
sales       | pricepaid  | numeric(8,2)                | delta32k | false   |       0 |
 false
sales       | commission | numeric(8,2)                | delta32k | false   |       0 |
 false
sales       | saletime   | timestamp without time zone | lzo      | false   |       0 |
 false
salesmonth | salesid    | integer                     | lzo      | false   |       0 |
 true
salesmonth | listid     | integer                     | none     | true    |       1 |
 true
salesmonth | sellerid   | integer                     | none     | false   |       2 |
 true
```

```
salesmonth | buyerid    | integer                     | lzo      | false    |        0 |
 true
salesmonth | eventid    | integer                     | mostly16 | false    |        0 |
 true
salesmonth | dateid     | smallint                    | lzo      | false    |        0 |
 true
salesmonth | qtysold    | smallint                    | mostly8  | false    |        0 |
 true
salesmonth | pricepaid  | numeric(8,2)                | delta32k | false    |        0 |
 false
salesmonth | commission | numeric(8,2)                | delta32k | false    |        0 |
 false
salesmonth | saletime   | timestamp without time zone | lzo      | false    |        0 |
 false
```

次に、各テーブルのサイズを確認します。

```
select count(*) from sales_monthly;
 count
-------
  2000
(1 row)

select count(*) from sales;
 count
-------
 412,214
(1 row)
```

ここで、次の ALTER TABLE APPEND コマンドを実行します。

```
alter table sales append from sales_monthly;
```

もう一度、各テーブルのサイズを確認します。SALES_MONTHLY テーブルは 0 行になり、SALES テーブルは 2000 行増えています。

```
select count(*) from sales_monthly;
 count
-------
     0
(1 row)
```

```
select count(*) from sales;
 count
-------
 414214
(1 row)
```

ソーステーブルにターゲットテーブルより多くの列がある場合は、IGNOREEXTRA パラメータを指定します。次の例では、IGNOREEXTRA パラメータを使用して、SALES テーブルに追加するときに SALES_LISTING テーブル列の追加の列を無視します。

```
alter table sales append from sales_listing ignoreextra;
```

ターゲットテーブルにソーステーブルより多くの列がある場合は、FILLTARGET パラメータを指定します。次の例では、FILLTARGET パラメータを使用して、SALES_MONTH テーブルに存在しない列を SALES_REPORT テーブルに入力します。

```
alter table sales_report append from sales_month filltarget;
```

次の例は、マテリアライズドビューをソースとして ALTER TABLE APPEND を使用する方法の例を示しています。

```
ALTER TABLE target_tbl APPEND FROM my_streaming_materialized_view;
```

この例のテーブル名とマテリアライズドビュー名はサンプルです。マテリアライズドビューからの追加は、マテリアライズドビューが [マテリアライズドビューへのストリーミング取り込み](#) に設定されている場合にのみ機能します。ソースマテリアライズドビューのすべてのレコードを、マテリアライズドビューと同じスキーマのターゲットテーブルに移動し、マテリアライズドビューはそのまま残します。これは、データのソースがテーブルである場合と同じ動作です。

ALTER USER

データベースユーザーを変更します。

必要な権限

以下に、ALTER USER に必要な権限を示します。

- スーパーユーザー

- ALTER USER の権限を持つユーザー
- 自身のパスワードを変更しようとする現在のユーザー

構文

```
ALTER USER username [ WITH ] option [, ... ]

where option is

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| PASSWORD { 'password' | 'md5hash' | DISABLE }
[ VALID UNTIL 'expiration_date' ]
| RENAME TO new_name |
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit | RESET SESSION TIMEOUT
| SET parameter { TO | = } { value | DEFAULT }
| RESET parameter
| EXTERNALID external_id
```

パラメータ

username

ユーザーの名前。

WiTH

オプションキーワード

CREATEDB | NOCREATEDB

CREATEDB オプションを使用すると、ユーザーは新しいデータベースを作成できます。NOCREATEDB がデフォルトです。

CREATEUSER | NOCREATEUSER

CREATEUSER オプションを使用すると、CREATE USER を含め、データベースに関するすべての権限を持つスーパーユーザーが作成されます。デフォルトは NOCREATEUSER です。詳細については、「superuser」を参照してください。

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

Amazon Redshift のシステムテーブルとビューに対するユーザーのアクセスレベルを指定する句です。

SYSLOG ACCESS RESTRICTED アクセス許可を持つ通常のユーザーは、ユーザーが表示できるシステムテーブルとビューで自分が生成した行のみを表示できます。デフォルトは RESTRICTED です。

SYSLOG ACCESS UNRESTRICTED アクセス許可を持つ通常のユーザーは、ユーザーが表示できるシステムテーブルとビューのすべての行 (別のユーザーが生成した行を含む) を表示できます。UNRESTRICTED を指定しても、スーパーユーザーが表示可能なテーブルへのアクセス権が、通常のユーザーに付与されるわけではありません。スーパーユーザーが表示可能なテーブルを表示できるのはスーパーユーザーだけです。

> **Note**
>
> システムテーブルに対する無制限のアクセス権限を付与されたユーザーは、別のユーザーが生成したデータへの可視性が提供されます。例えば、STL_QUERY と STL_QUERYTEXT には INSERT、UPDATE、および DELETE ステートメントのフルテキストが含まれており、ユーザーが生成した機密データがこれらに含まれている可能性があります。

SVV_TRANSACTIONS のすべての行は、すべてのユーザーが表示可能です。

詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

PASSWORD { 'password' | 'md5hash' | DISABLE }

ユーザーのパスワードを設定します。

デフォルトでは、ユーザーはパスワードが無効になっていない限り、自分のパスワードを変更できます。ユーザーのパスワードを無効にするには、DISABLE を指定します。ユーザーのパスワードが無効になると、パスワードはシステムから削除され、ユーザーは AWS Identity and Access Management (IAM) ユーザーの一時的認証情報を使用してのみログオンできます。詳細については、「IAM 認証を使用したデータベースユーザー認証情報の生成」を参照してください。スーパーユーザーのみが、パスワードを有効または無効にできます。スーパーユーザーのパスワードを無効にすることはできません。パスワードを有効にするには、ALTER USER を実行し、パスワードを指定します。

PASSWORD パラメータの使用の詳細については、「CREATE USER」を参照してください。

VALID UNTIL 'expiration_date'

パスワードに失効日があることを指定します。値 'infinity' を使用すると、失効日を設定しないようにすることができます。このパラメータの有効なデータ型はタイムスタンプです。

このパラメータを使用できるのはスーパーユーザーだけです。

RENAME TO

ユーザーの名前を変更します。

new_name

ユーザーの新しい名前。有効な名前の詳細については、「名前と識別子」を参照してください。

> **Important**
>
> ユーザーの名前を変更すると、ユーザーのパスワードもリセットする必要があります。リセット後のパスワードは、以前のパスワードと異ならなくても構いません。ユーザー名はパスワード暗号化の一部として使用されるため、ユーザーの名前を変更すると、パスワードは消去されます。パスワードをリセットするまで、ユーザーはログオンできません。次に例を示します。
>
> ```
> alter user newuser password 'EXAMPLENewPassword11';
> ```

CONNECTION LIMIT { limit | UNLIMITED }

ユーザーが同時に開けるデータベース接続の最大数。この制限はスーパーユーザーには適用されません。同時接続の最大数を許可するには、UNLIMITED キーワードを使用します。データベースごとの接続数の制限が適用される場合もあります。詳細については、「CREATE DATABASE」を参照してください。デフォルトは UNLIMITED です。現在の接続を確認するには、STV_SESSIONSシステムビューに対してクエリを実行します。

> **Note**
>
> ユーザーとデータベースの両方の接続制限が適用される場合は、ユーザーが接続しようとしたときに、両方の制限内に未使用の接続スロットがなければなりません。

SESSION TIMEOUT limit | RESET SESSION TIMEOUT

セッションが非アクティブまたはアイドル状態を維持する最大時間 (秒) です。指定できる範囲は 60 秒 (1 分) から 1,728,000 秒 (20 日) です。ユーザーに対しセッションタイムアウトが設定されていない場合は、クラスターの設定値が適用されます。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクォータと制限」を参照してください。

設定されたセッションタイムアウトは、新しいセッションにのみ適用されます。

開始時刻、ユーザー名、セッションタイムアウトなど、アクティブなユーザーセッションに関する情報を表示するには、STV_SESSIONSシステムビューを参照します。ユーザーセッションの履歴に関する情報を表示するには、STL_SESSIONSビューを参照します。セッションタイムアウト値など、データベースユーザーに関する情報を取得するには、SVL_USER_INFOビューを参照します。

SET

指定したユーザーによって実行されるすべてのセッションに対して、設定パラメータを新しいデフォルト値に設定します。

RESET

指定したユーザーに対して、設定パラメータを元のデフォルト値にリセットします。

parameter

設定またはリセットするパラメータの名前。

value

パラメータの新しい値。

DEFAULT

指定したユーザーによって実行されるすべてのセッションに対して、設定パラメータをデフォルト値に設定します。

EXTERNALID external_id

ID プロバイダーに関連付けられているユーザーの識別子。ユーザーはパスワードを無効にする必要があります。詳細については、「Amazon Redshift 用のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

使用に関する注意事項

- rdsdb を変更する試行 — `rdsdb` という名前のユーザーは変更できません。
- 不明なパスワードの作成 – AWS Identity and Access Management (IAM) 認証情報を使用してデータベースのユーザー認証情報を作成する場合、一時的認証情報を使用してのみログオンできるスーパーユーザーを作成することをお勧めします。スーパーユーザーのパスワードを無効にすることはできませんが、ランダムに生成された MD5 ハッシュ文字列を使って不明なパスワードを作成することはできます。

```
alter user iam_superuser password 'md51234567890123456780123456789012';
```

- search_path の設定 – ALTER USER コマンドに search_path パラメータを設定した場合、指定したユーザーの次のログイン時に変更が反映されます。現在のユーザーとセッションの search_path 値を変更する場合は、SET コマンドを使用します。
- タイムゾーンの設定 – ALTER USER コマンドに SET TIMEZONE を使用する場合、指定したユーザーの次のログイン時に変更が反映されます。
- 動的データマスキングと行レベルのセキュリティポリシーとの連携 – プロビジョニングされたクラスターまたはサーバーレス名前空間に動的データマスキングまたは行レベルのセキュリティポリシーが適用されている場合、次のコマンドは標準ユーザーに対してブロックされます。

```
ALTER <current_user> SET enable_case_sensitive_super_attribute/
enable_case_sensitive_identifier/downcase_delimited_identifier
```

  これらの設定オプションを設定できるのは、スーパーユーザーと、ALTER USER 権限を持つユーザーのみです。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

例

次の例では、ユーザー ADMIN にデータベースを作成する権限を与えます。

```
alter user admin createdb;
```

次の例では、ユーザー ADMIN のパスワードを `adminPass9` に設定し、パスワードの有効期限切れの日時を設定します。

```
alter user admin password 'adminPass9'
valid until '2017-12-31 23:59';
```

次の例では、ユーザー名を ADMIN から SYSADMIN に変更します。

```
alter user admin rename to sysadmin;
```

次の例では、ユーザーのアイドルセッションタイムアウトを 300 秒に変更しています。

```
ALTER USER dbuser SESSION TIMEOUT 300;
```

ユーザーのアイドルセッションタイムアウトをリセットします。これをリセットすると、クラスターの設定内容が適用されます。このコマンドを実行するには、データベースのスーパーユーザー権限を持つ必要があります。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクォータと制限」を参照してください。

```
ALTER USER dbuser RESET SESSION TIMEOUT;
```

次の例では、bob という名前のユーザーの外部 ID を更新します。名前空間は myco_aad です。名前空間が登録された ID プロバイダーに関連付けられていない場合、エラーが発生します。

```
ALTER USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

次の例では、特定のデータベースユーザーが実行するすべてのセッションのタイムゾーンを設定します。現在のセッションではなく、後のセッションのタイムゾーンを変更します。

```
ALTER USER odie SET TIMEZONE TO 'Europe/Zurich';
```

次の例では、ユーザー bob が開くことができるデータベース接続の最大数を設定します。

```
ALTER USER bob CONNECTION LIMIT 10;
```

ANALYZE

クエリプランナーで使用するテーブル統計を更新します。

必要な権限

ANALYZE に必要な権限を以下に示します。

- スーパーユーザー
- ANALYZE の権限を持つユーザー
- 関連付けの所有者
- テーブルの共有先であるデータベース所有者

構文

```
ANALYZE [ VERBOSE ]
[ [ table_name [ ( column_name [, ...] ) ] ]
[ PREDICATE COLUMNS | ALL  COLUMNS ]
```

パラメータ

VERBOSE

ANALYZE オペレーションに関する進捗情報メッセージを返す句。このオプションは、テーブルを指定しないときに役立ちます。

table_name

一時テーブルを含む、特定のテーブルを分析できます。テーブルをそのスキーマ名で修飾することができます。また、table_name を指定して単一のテーブルを分析することもできます。1 つの ANALYZE table_name ステートメントで複数の table_name を指定することはできません。table_name 値を指定しなかった場合、システムカタログの永続テーブルを含め、現在接続されているデータベースのすべてのテーブルが分析されます。最後の ANALYZE 以降に変更された行の割合が分析のしきい値よりも低い場合、Amazon Redshift はテーブルの分析をスキップします。詳細については、「分析のしきい値」を参照してください。

Amazon Redshift のシステムテーブル (STL テーブルと STV テーブル) を分析する必要はありません。

column_name

table_name を指定する場合、テーブルの 1 つ以上の列を指定することもできます (括弧内に列をカンマ区切りリストとして指定します)。列リストが指定された場合、リストされている列のみが分析されます。

PREDICATE COLUMNS | ALL COLUMNS

ANALYZE に述語列のみを含めるかどうかを示す句。PREDICATE COLUMNS を指定すると、前のクエリで述語として使用されている列、または述語として使用される可能性が高い列のみが

分析されます。すべての行を分析するには、ALL COLUMNS を指定します。デフォルトは ALL COLUMNS です。

次のいずれかが当てはまる場合、列は述語列のセットに含まれます。

- この列は、フィルタ、結合条件、または GROUP BY 句の一部としてクエリで使用されています。
- 列が分散キーです。
- 列はソートキーの一部です。

例えば、テーブルがまだ照会されていないなどの理由で列が述語列としてマークされていない場合は、PREDICATE COLUMNS が指定されていてもすべての列が分析されます。この場合、Amazon Redshift は「*table-name* の述語列が見つかりません」「すべての列を分析しています」などのメッセージで応答することがあります。述語の列の詳細については、テーブルを分析するを参照してください。

使用に関する注意事項

Amazon Redshift では、以下のコマンドを使用して作成したテーブルで自動的に ANALYZE を実行します。

- CREATE TABLE AS
- CREATE TEMP TABLE AS
- SELECT INTO

外部テーブルを分析することはできません。

最初に作成したとき、これらのテーブルに ANALYZE コマンドを実行する必要はありません。変更する場合は、他のテーブルと同じように分析する必要があります。

分析のしきい値

処理時間を短縮し、システム全体のパフォーマンスを向上させるために、最後の ANALYZE コマンドの実行以降に変更された行の割合が、analyze_threshold_percent パラメータで指定された分析のしきい値よりも低い場合、Amazon Redshift はテーブルの分析をスキップします。デフォルトでは、`analyze_threshold_percent`は 10 です。現在のセッションの `analyze_threshold_percent` を変更するには、SETコマンドを実行します。次の例では、`analyze_threshold_percent`を 20 パーセントに変更します。

```
set analyze_threshold_percent to 20;
```

数行のみを変更した場合にテーブルを分析するには、analyze_threshold_percentに任意の小さい数字を設定します。たとえば、analyze_threshold_percentを 0.01 に設定すると、少なくとも 10,000 行が変更された場合は 100,000,000 行のテーブルをスキップしません。

```
set analyze_threshold_percent to 0.01;
```

ANALYZE でテーブルがスキップされるのは、分析のしきい値を満たしていないためです。Amazon Redshift は次のメッセージを返します。

```
ANALYZE SKIP
```

行が変更されていない場合でもすべてのテーブルを分析するには、analyze_threshold_percentを 0 に設定します。

ANALYZE オペレーションの結果を表示するには、STL_ANALYZEシステムテーブルに対してクエリを実行します。

テーブル分析の詳細については、「テーブルを分析する」を参照してください。

例

TICKIT データベースのすべてのテーブルを分析し、進捗情報を返します。

```
analyze verbose;
```

LISTING テーブルのみを分析します。

```
analyze listing;
```

VENUE テーブルの VENUEID 列と VENUENAME 列を分析します。

```
analyze venue(venueid, venuename);
```

VENUE テーブルの述語の列のみを分析します。

```
analyze venue predicate columns;
```

ANALYZE COMPRESSION

圧縮分析を行い、分析されたテーブルの推奨列エンコードスキームのレポートを生成します。レポートには、列ごとに RAW エンコードと比較したディスク容量の圧縮可能率の推定値が含まれます。

構文

```
ANALYZE COMPRESSION
[ [ table_name ]
[ ( column_name [, ...] ) ] ]
[COMPROWS numrows]
```

パラメータ

table_name

一時テーブルを含む、特定のテーブルの圧縮を分析できます。テーブルをそのスキーマ名で修飾することができます。また、table_name を指定して単一のテーブルを分析することもできます。table_name を指定しなかった場合、現在接続されているデータベースがすべて分析されます。1 つの ANALYZE COMPRESSION ステートメントで複数の table_name を指定することはできません。

column_name

table_name を指定する場合、テーブルの 1 つ以上の列を指定することもできます (括弧内に列をカンマ区切りリストとして指定します)。

COMPROWS

圧縮分析のサンプルサイズとして使用される行数。分析は各データスライスの行に対して実行されます。例えば、COMPROWS 1000000 (1,000,000) を指定し、システムに合計 4 つのスライスが含まれている場合、スライスごとに 250,000 行のみが読み取られ、分析されます。COMPROWS を指定しない場合、サンプルサイズはデフォルトでスライスごとに 100,000 になります。COMPROWS の値がスライスごとに 100,000 行のデフォルト値より小さい場合、自動的にデフォルト値にアップグレードされます。ただし、テーブルのデータが不十分であるため有意のサンプルを作成できない場合、圧縮分析では推奨を作成しません。COMPROWS 数がテーブルの行数より大きい場合でも、ANALYZE COMPRESSION コマンドは続行し、利用可能なすべての行に対して圧縮分析を実行します。テーブルが指定されていない場合、COMPROWS を使用するとエラーが発生します。

numrows

圧縮分析のサンプルサイズとして使用される行数。numrows の許容範囲は 1000～1000000000 (1,000,000,000) の数値です。

使用に関する注意事項

ANALYZE COMPRESSION は排他的テーブルロックを取得し、テーブルに対する同時読み取り書き込みが防止されます。ANALYZE COMPRESSION コマンドは、テーブルがアイドル状態になっている場合にのみ実行してください。

テーブルの内容サンプルに基づいて推奨列エンコードスキームを取得するには、ANALYZE COMPRESSION を実行します。ANALYZE COMPRESSION はアドバイスツールであり、テーブルの列エンコードは変更しません。推奨エンコードは、テーブルを再作成するか、同じスキーマを持つ新しいテーブルを作成することにより適用できます。適切なエンコードスキームを使用して未圧縮テーブルを再作成すると、オンディスクフットプリントを大幅に減らすことができます。このアプローチにより、ディスクスペースを節約でき、I/O 関連ワークロードのクエリパフォーマンスが向上します。

ANALYZE COMPRESSION は、実際の分析フェーズをスキップし、SORTKEY として指定されている任意の列で元のエンコードタイプを直接返します。これは、SORTKEY 列が他の列よりも大幅に圧縮されている場合に、範囲制限されたスキャンのパフォーマンスが低下する可能性があるためです。

例

次の例は、LISTING テーブルのみの列に対するエンコードおよび推定減少パーセントを示しています。

```
analyze compression listing;

  Table  |     Column     | Encoding | Est_reduction_pct
---------+----------------+----------+------------------
 listing | listid         | az64     | 40.96
 listing | sellerid       | az64     | 46.92
 listing | eventid        | az64     | 53.37
 listing | dateid         | raw      | 0.00
 listing | numtickets     | az64     | 65.66
 listing | priceperticket | az64     | 72.94
 listing | totalprice     | az64     | 68.05
```

```
 listing | listtime       | az64     | 49.74
```

次の例は､SALES テーブルの QTYSOLD、COMMISSION、SALETIME の各列を分析します。

```
analyze compression sales(qtysold, commission, saletime);

 Table |   Column   | Encoding | Est_reduction_pct
-------+------------+----------+-------------------
 sales | salesid    | N/A      | 0.00
 sales | listid     | N/A      | 0.00
 sales | sellerid   | N/A      | 0.00
 sales | buyerid    | N/A      | 0.00
 sales | eventid    | N/A      | 0.00
 sales | dateid     | N/A      | 0.00
 sales | qtysold    | az64     | 83.06
 sales | pricepaid  | N/A      | 0.00
 sales | commission | az64     | 71.85
 sales | saletime   | az64     | 49.63
```

ATTACH MASKING POLICY

既存の動的データマスキングポリシーを列にアタッチします。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、マスキングポリシーをアタッチできます。

構文

```
ATTACH MASKING POLICY policy_name
   ON { relation_name }
   ( {output_columns_names | output_path} ) [ USING ( {input_column_names | input_path
 )} ]
   TO { user_name | ROLE role_name | PUBLIC }
   [ PRIORITY priority ];
```

パラメータ

policy_name

アタッチするマスキングポリシーの名前。

relation_name

マスキングポリシーをアタッチするリレーションの名前。

output_column_names

マスキングポリシーが適用される列の名前。

output_paths

マスキングポリシーが適用される SUPER オブジェクトのフルパス (列名を含む)。例えば、`person` という SUPER 型の列を使用したリレーションの場合、output_path は `person.name.first_name` になります。

input_column_names

マスキングポリシーが入力として受け取る列の名前。このパラメータはオプションです。指定しない場合、マスキングポリシーは output_column_names を入力として使用します。

input_paths

マスキングポリシーが入力として受け取る SUPER オブジェクトのフルパス。このパラメータはオプションです。指定しない場合、マスキングポリシーは output_path を入力に使用します。

user_name

マスキングポリシーをアタッチするユーザーの名前。ユーザーと列、またはロールと列の同じ組み合わせに 2 つのポリシーをアタッチすることはできません。ポリシーをユーザーに、別のポリシーをユーザーのロールにアタッチできます。この場合、優先度の高いポリシーが適用されます。

1 回の ATTACH MASKING POLICY コマンドで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

role_name

マスキングポリシーがアタッチされるロールの名前。同じ列/ロールのペアに 2 つのポリシーをアタッチすることはできません。ポリシーをユーザーに、別のポリシーをユーザーのロールにアタッチできます。この場合、優先度の高いポリシーが適用されます。

1 回の ATTACH MASKING POLICY コマンドで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

PUBLIC

テーブルにアクセスするすべてのユーザーにマスキングポリシーをアタッチします。特定の列/ユーザーまたは列/ロールのペアにアタッチされている他のマスキングポリシーを適用するには、PUBLIC ポリシーよりも高い優先度を設定する必要があります。

1 回の ATTACH MASKING POLICY コマンドで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

priority

マスキングポリシーの優先度。特定のユーザーのクエリに複数のマスキングポリシーが適用される場合、最も優先度の高いポリシーが適用されます。

2 つの異なるポリシーを同じ優先順位で同じ列にアタッチすることはできません。2 つの異なるポリシーが別々のユーザーまたはロールにアタッチされている場合でも同様です。ポリシーをアタッチするユーザーまたはロールが毎回異なる場合に限り、同じポリシーを同じテーブル、出力列、入力列、優先度のパラメータのセットに複数回アタッチできます。

ロールが異なっていても、その列にアタッチされている別のポリシーと同じ優先度の列にポリシーを適用することはできません。このフィールドはオプションです。優先度を指定しない場合、マスキングポリシーのアタッチの優先度はデフォルトで 0 に設定されます。

ATTACH RLS POLICY

行レベルのセキュリティポリシーをテーブルで 1 人以上のユーザーまたはロールにアタッチします。

スーパーユーザーとユーザー、または `sys:secadmin` ロールを持つロールは、ポリシーをアタッチできます。

構文

```
ATTACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
TO { user_name | ROLE role_name | PUBLIC } [, ...]
```

パラメータ

policy_name

ポリシーの名前。

ON [TABLE] table_name [, ...]

行レベルのセキュリティポリシーがアタッチされているリレーション。

TO { user_name | ROLE role_name | PUBLIC} [, ...]

ポリシーが指定した 1 つ以上のユーザーまたはロールにアタッチするかどうか指定します。

使用に関する注意事項

ATTACH RLS POLICY ステートメントを使用する際、次の点に注意してください。

- アタッチされるテーブルには、ポリシー作成ステートメントの WITH 句に一覧表示されているすべての列が含まれている必要があります。
- Amazon Redshift RLS は、以下のオブジェクトへの RLS ポリシーのアタッチをサポートしていません。
  - カタログテーブル
  - データベース間のリレーション
  - 外部テーブル
  - 一時テーブル
  - ルックアップテーブル
- RLS ポリシーをスーパーユーザーまたは `sys:secadmin` アクセス許可を持つユーザーにアタッチすることはできません。

例

次の例では、ポリシーをロールのテーブルにアタッチします。

```
ATTACH RLS POLICY policy_concerts ON tickit_category_redshift TO ROLE analyst, ROLE
 dbadmin;
```

BEGIN

トランザクションを開始します。START TRANSACTION と同義です。

トランザクションは、1 つのコマンドまたは複数のコマンドから構成される、1 つの論理的作業単位です。通常、1 つのトランザクションのすべてのコマンドはデータベースの 1

つのスナップショットに対して実行されます。その開始時刻は、システム設定パラメータ transaction_snapshot_begin に設定されている値によって決定されます。

デフォルトでは、個々の Amazon Redshift オペレーション (クエリ、DDL ステートメント、ロード) はデータベースに自動的にコミットされます。後続の作業が完了するまでオペレーションのコミットを停止する場合、BEGIN ステートメントでトランザクションを開き、必要なコマンドを実行し、COMMITまたは END ステートメントでトランザクションを閉じます。必要に応じて、ROLLBACKステートメントを使用して、進行中のトランザクションを停止できます。この動作の例外は TRUNCATE コマンドです。このコマンドは実行されているトランザクションをコミットします。ロールバックすることはできません。

構文

```
BEGIN [ WORK | TRANSACTION ] [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]

START TRANSACTION [ ISOLATION LEVEL option ] [ READ WRITE | READ ONLY ]

Where option is

SERIALIZABLE
| READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ

Note: READ UNCOMMITTED, READ COMMITTED, and REPEATABLE READ have no
operational impact and map to SERIALIZABLE in Amazon Redshift. You can see database
 isolation levels on your cluster
by querying the stv_db_isolation_level table.
```

パラメータ

WORK

オプションキーワード

TRANSACTION

オプションキーワード。WORK と TRANSACTION は同義語です。

ISOLATION LEVEL SERIALIZABLE

デフォルトで直列化可能な分離がサポートされているため、この構文がステートメントに含まれているかどうかに関係なく、トランザクションの動作は同じです。詳細については、「[同時書き込み操作を管理する](#)」を参照してください。他の分離レベルはサポートされていません。

> **Note**
>
> SQL 標準では、以下の状態を避けるため、4 つのレベルのトランザクション分離を定義しています: ダーティリード (トランザクションが未コミットの同時トランザクションによって書き込まれたデータを読み取ります)、非再現リード (トランザクションが以前に読み取ったデータを再読み取りし、初回読み取り後にコミットされた別のトランザクションによってデータが変更されたことを検出します)、およびファントムリード (トランザクションがクエリを再実行し、検索条件を満たす行セットを返した後、最近コミットされた別のトランザクションのために行セットが変更されていることを検出します)。
>
> - 未コミット読み取り: ダーティリード、非再現リード、およびファントムリードの可能性があります。
> - コミット済み読み取り: 非再現リードおよびファントムリードの可能性があります。
> - 再現可能読み取り: ファントムリードの可能性があります。
> - 直列化可能: ダーティリード、非再現リード、およびファントムリードを防止します。
>
> 4 つのトランザクション分離レベルのいずれも使用できますが、Amazon Redshift ではすべての分離レベルを直列化可能として処理します。

READ WRITE

トランザクションに読み取り書き込み権限を与えます。

READ ONLY

トランザクションに読み取り専用権限を与えます。

例

次の例では、直列化可能なトランザクションブロックを開始します。

```
begin;
```

次の例では、直列化可能分離レベルと読み取り書き込み権限を持つトランザクションブロックを開始します。

```
begin read write;
```

CALL

ストアドプロシージャを実行します。CALL コマンドには、プロシージャ名と入力引数の値を含める必要があります。CALL ステートメントを使用してストアドプロシージャを呼び出す必要があります。

> **Note**
> CALL を通常のクエリの一部にすることはできません。

構文

```
CALL sp_name ( [ argument ] [, ...] )
```

パラメータ

sp_name

実行するプロシージャの名前。

argument

入力引数の値。このパラメータは、関数名 (`pg_last_query_id()`) とすることもできます。クエリは CALL の引数として使用できません。

使用に関する注意事項

Amazon Redshift のストアドプロシージャは、以下に説明するように、ネストされた呼び出しと再帰呼び出しをサポートしています。さらに、以下に説明するように、ドライバーのサポートが最新であることを確認します。

トピック

- ネストされた呼び出し

- [ドライバーのサポート]

ネストされた呼び出し

Amazon Redshift のストアドプロシージャは、ネストされた呼び出しと再帰呼び出しをサポートしています。ネストレベルの最大許容数は 16 です。ネストされた呼び出しは、ビジネスロジックをより小さいプロシージャにカプセル化できます。これらのプロシージャを複数の発信者が共有できます。

ネストされたプロシージャに出力パラメータが含まれていると、このプロシージャを呼び出す場合、内部プロシージャは INOUT 引数を定義する必要があります。この場合、内部プロシージャを非定数変数で渡す必要があります。OUT 引数は許可されません。この動作が発生するのは、内部呼び出しの出力を保持するために変数が必要であるためです。

内部プロシージャと外部プロシージャの関係は、SVL_STORED_PROC_CALLの `from_sp_call` 列に記録されます。

次の例は、INOUT 引数を通じて、ネストされたプロシージャ呼び出しに渡される変数を示しています。

```
CREATE OR REPLACE PROCEDURE inner_proc(INOUT a int, b int, INOUT c int) LANGUAGE
 plpgsql
AS $$
BEGIN
  a := b * a;
  c := b * c;
END;
$$;

CREATE OR REPLACE PROCEDURE outer_proc(multiplier int) LANGUAGE plpgsql
AS $$
DECLARE
  x int := 3;
  y int := 4;
BEGIN
  DROP TABLE IF EXISTS test_tbl;
  CREATE TEMP TABLE test_tbl(a int, b varchar(256));
  CALL inner_proc(x, multiplier, y);
  insert into test_tbl values (x, y::varchar);
END;
$$;

CALL outer_proc(5);
```

```
SELECT * from test_tbl;
 a  | b
----+----
 15 | 20
(1 row)
```

ドライバーのサポート

Java Database Connectivity (JDBC) ドライバーと Open Database Connectivity (ODBC) ドライバーを、Amazon Redshift のストアドプロシージャをサポートする最新バージョンにアップグレードすることをお勧めします。

クライアントツールで使用しているドライバーの API オペレーションが CALL ステートメントをサーバーにパススルーする場合は、既存のドライバーを使用できることがあります。出力パラメータ (ある場合) は、1 行の結果セットとして返されます。

最新バージョンの Amazon Redshift JDBC ドライバーと ODBC ドライバーには、ストアドプロシージャを検出するためのメタデータサポートが含まれています。カスタム Java アプリケーション用の `CallableStatement` サポートも含まれています。ドライバーの詳細については、「Amazon Redshift 管理ガイド」の「SQL クライアントツールを使用して Amazon Redshift クラスターに接続する」を参照してください。

以下の例は、ストアドプロシージャ呼び出しで JDBC ドライバーの複数の異なる API オペレーションを使用する方法を示しています。

```
void statement_example(Connection conn) throws SQLException {
  statement.execute("CALL sp_statement_example(1)");
}

void prepared_statement_example(Connection conn) throws SQLException {
  String sql = "CALL sp_prepared_statement_example(42, 84)";
  PreparedStatement pstmt = conn.prepareStatement(sql);
  pstmt.execute();
}

void callable_statement_example(Connection conn) throws SQLException {
  CallableStatement cstmt = conn.prepareCall("CALL sp_create_out_in(?,?)");
  cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
  cstmt.setInt(2, 42);
  cstmt.executeQuery();
  Integer out_value = cstmt.getInt(1);
```

```
}
```

例

次の例では、プロシージャ名 `test_spl` を呼び出します。

```
call test_sp1(3,'book');
INFO:  Table "tmp_tbl" does not exist and will be skipped
INFO:  min_val = 3, f2 = book
```

次の例では、プロシージャ名 `test_spl2` を呼び出します。

```
call test_sp2(2,'2019');

         f2          | column2
---------------------+---------
 2019+2019+2019+2019 | 2
(1 row)
```

CANCEL

現在実行中のデータベースクエリをキャンセルします。

CANCEL コマンドは実行中のクエリのプロセス ID またはセッション ID を必要とし、クエリがキャンセルされたことを確認する確認メッセージを表示します。

必要な権限

CANCEL に必要な権限を以下に示します。

- 自身のクエリをキャンセルしているスーパーユーザー
- ユーザーのクエリをキャンセルしているスーパーユーザー
- CANCEL の権限を持ち、ユーザーのクエリをキャンセルしているユーザー
- 自身のクエリをキャンセルしているユーザー

構文

```
CANCEL process_id [ 'message' ]
```

パラメータ

process_id

Amazon Redshift クラスターで実行されているクエリをキャンセルするには、キャンセルするクエリに対応する STV_RECENTS の pid (プロセス ID) を使用します。

Amazon Redshift Serverless ワークグループで実行されているクエリをキャンセルするには、キャンセルするクエリに対応する SYS_QUERY_HISTORY の session_id を使用します。

'message'

オプションの確認メッセージ。クエリのキャンセルが完了したときに表示されます。メッセージを指定しなかった場合、Amazon Redshift は確認としてデフォルトメッセージを表示します。メッセージは一重引用符で囲む必要があります。

使用に関する注意事項

クエリ ID を指定してクエリをキャンセルすることはできません。クエリのプロセス ID (PID) またはセッション ID を指定する必要があります。ユーザーによって現在実行されているクエリのみキャンセルできます。スーパーユーザーはすべてのクエリをキャンセルできます。

複数のセッション内のクエリが同じテーブルでロックを保持する場合は、PG_TERMINATE_BACKEND 関数を使用してセッションの 1 つを終了できます。これを行うと、終了したセッションで現在実行されているトランザクションがすべてのロックを解除し、トランザクションがロールバックされます。STV_LOCKS システムテーブルに対してクエリを実行して、現在保持しているロックを表示します。

Amazon Redshift は特定の内部イベントに続いてアクティブなセッションを再起動し、新しい PID を割り当てる場合があります。PID が変更されている場合は、次のようなエラーメッセージが表示されることがあります。

```
Session <PID> does not exist. The session PID might have changed. Check the
 stl_restarted_sessions system table for details.
```

新しい PID を見つけるには、STL_RESTARTED_SESSIONSシステムテーブルに対してクエリを実行し、oldpid列でフィルタリングします。

```
select oldpid, newpid from stl_restarted_sessions where oldpid = 1234;
```

例

Amazon Redshift クラスターで現在実行されているクエリをキャンセルするには、キャンセルするクエリのプロセス ID を最初に取得します。現在実行されているすべてのクエリのプロセス ID を確認するには、次のコマンドを入力します。

```
select pid, starttime, duration,
trim(user_name) as user,
trim (query) as querytxt
from stv_recents
where status = 'Running';

pid |         starttime          | duration |   user   |    querytxt
-----+----------------------------+----------+----------+-----------------
802 | 2008-10-14 09:19:03.550885 |      132 | dwuser | select
venuename from venue where venuestate='FL', where venuecity not in
('Miami' , 'Orlando');
834 | 2008-10-14 08:33:49.473585 |  1250414 | dwuser | select *
from listing;
964 | 2008-10-14 08:30:43.290527 |   326179 | dwuser | select
sellerid from sales where qtysold in (8, 10);
```

クエリテキストをチェックし、キャンセルするクエリに対応するプロセス ID (PID) を確認します。

次のコマンドを入力して、PID 802 を使用してクエリをキャンセルします。

```
cancel 802;
```

クエリが実行されていたセッションは、次のメッセージを表示します。

```
ERROR:  Query (168) cancelled on user's request
```

ここでは、168はクエリ ID です (クエリをキャンセルするために使用されるプロセス ID ではありません)。

また、デフォルトメッセージの代わりに表示するカスタム確認メッセージを指定することもできます。カスタムメッセージを指定するには、CANCEL コマンドの最後に一重引用符で囲んだメッセージを付けます。

```
cancel 802 'Long-running query';
```

クエリが実行されていたセッションは、次のメッセージを表示します。

```
ERROR:  Long-running query
```

CLOSE

(オプション) 開いているカーソルと関連付けられたすべての空きリソースを閉じます。COMMIT、END、および ROLLBACK は自動的にカーソルを閉じるため、CLOSE コマンドを使用して明示的にカーソルを閉じる必要はありません。

詳細については、「DECLARE」、「FETCH」を参照してください。

構文

```
CLOSE cursor
```

パラメータ

cursor

閉じるカーソルの名前。

CLOSE の例

次のコマンドはカーソルを閉じ、コミットを実行してトランザクションを終了します。

```
close movie_cursor;
commit;
```

COMMENT

データベースオブジェクトに関するコメントを作成するか変更します。

構文

```
COMMENT ON
{
TABLE object_name |
COLUMN object_name.column_name |
CONSTRAINT constraint_name ON table_name |
```

```
DATABASE object_name |
VIEW object_name
}
IS 'text' | NULL
```

パラメータ

object_name

コメント対象のデータベースオブジェクトの名前。コメントは次のオブジェクトに追加できます。

- TABLE
- COLUMN (column_name も取ります)
- CONSTRAINT (constraint_name と table_name も取ります)
- DATABASE
- VIEW
- SCHEMA

IS 'text' | NULL

指定したオブジェクトに追加または置換するコメントテキスト。テキスト文字列のデータ型は TEXT です。コメントは一重引用符で囲みます。コメントテキストを削除するには、値を NULL に設定します。

column_name

コメント対象の列の名前。COLUMN のパラメータ。`object_name` で指定するテーブルの後に指定します。

constraint_name

コメント対象の制約の名前。CONSTRAINT のパラメータ。

table_name

制約を含むテーブルの名前。CONSTRAINT のパラメータ。

使用に関する注意事項

コメントを追加または更新するには、スーパーユーザーまたはデータベースオブジェクトの所有者である必要があります。

データベースに関するコメントは現在のデータベースにのみ適用できます。異なるデータベースにコメントしようとすると、警告メッセージが表示されます。存在しないデータベースに関するコメントに対しても、同じ警告が表示されます。

外部テーブル、外部列、遅延バインドビューの列に関するコメントはサポートされていません。

例

次の使用例は、SALES テーブルにコメントを追加します。

```
COMMENT ON TABLE sales IS 'This table stores tickets sales data';
```

次の使用例は、SALES テーブルにコメントを追加します。

```
select obj_description('public.sales'::regclass);

obj_description
-------------------------------------
This table stores tickets sales data
```

次の使用例は、SALES テーブルからコメントを削除します。

```
COMMENT ON TABLE sales IS NULL;
```

次の使用例は、SALES テーブルの EVENTID 列にコメントを追加します。

```
COMMENT ON COLUMN sales.eventid IS 'Foreign-key reference to the EVENT table.';
```

次の使用例は、SALES テーブルの EVENTID 列 (列番号 5) にコメントを表示します。

```
select col_description( 'public.sales'::regclass, 5::integer );

col_description
-----------------------------------------
Foreign-key reference to the EVENT table.
```

次の例では、説明的なコメントを EVENT テーブルに追加します。

```
comment on table event is 'Contains listings of individual events.';
```

コメントを表示するには、PG_DESCRIPTION システムカタログをクエリします。次の例は、EVENT テーブルの説明を返します。

```
select * from pg_catalog.pg_description
where objoid =
(select oid from pg_class where relname = 'event'
and relnamespace =
(select oid from pg_catalog.pg_namespace where nspname = 'public') );

objoid | classoid | objsubid | description
-------+----------+----------+----------------------------------------
116658 |     1259 |        0 | Contains listings of individual events.
```

COMMIT

現在のトランザクションをデータベースにコミットします。このコマンドはトランザクションからのデータベース更新を永続的なものにします。

構文

```
COMMIT [ WORK | TRANSACTION ]
```

パラメータ

WORK

オプションキーワード このキーワードは、ストアドプロシージャ内ではサポートされていません。

TRANSACTION

オプションキーワード WORK と TRANSACTION は同義語です。ストアドプロシージャ内ではいずれもサポートされていません。

ストアドプロシージャ内での COMMIT の使用方法については、[トランザクションの管理](#)を参照してください。

例

次の各例では、現在のトランザクションをデータベースにコミットします。

```
commit;
```

```
commit work;
```

```
commit transaction;
```

COPY

データファイルまたは Amazon DynamoDB テーブルから、テーブルにデータをロードします。ファイルは Amazon Simple Storage Service (Amazon S3) バケット、Amazon EMR クラスターまたは Secure Shell (SSH) 接続を使用したリモートホストに配置できます。

> **Note**
>
> Amazon Redshift Spectrum の外部テーブルは読み込み専用です。外部テーブルには COPY できません。

COPY コマンドは、入力データを追加の行としてテーブルに付加します。

どのソースからであっても、単一の入力行の最大サイズは 4 MB です。

トピック

必要なアクセス許可

COPY コマンドを使用するには、Amazon Redshift テーブルに対する INSERT 権限が必要です。

COPY 構文

```
COPY table-name
[ column-list ]
FROM data_source
authorization
[ [ FORMAT ] [ AS ] data_format ]
[ parameter [ argument ] [, ... ] ]
```

テーブル名、データソース、データにアクセスするための許可のわずか 3 つのパラメータで COPY オペレーションを実行できます。

Amazon Redshift は COPY コマンドの機能を拡張し、マルチデータソースから複数のサービスデータ形式でのデータのロード、ロードデータへのアクセス制御、データ変換の管理、ロードオペレーションの管理を可能にします。

以下のセクションでは、COPY コマンドの必須パラメータと、機能別に分類したオプションのパラメータを示します。また、各パラメータについて説明し、さまざまなオプションがどのように連携するかについても説明します。アルファベット順のパラメータリストを使用すると、パラメータの説明に直接移動できます。

必須パラメータ

COPY コマンドには 3 つの要素が必要です。

- Table Name
- Data Source
- Authorization

最も単純な COPY コマンドは次の形式を使用します。

```
COPY table-name
FROM data-source
authorization;
```

次の例では、CATDEMO というテーブルを作成し、Amazon S3 の `category_pipe.txt` というデータファイルからサンプルデータを含むテーブルをロードします。

```
create table catdemo(catid smallint, catgroup varchar(10), catname varchar(10), catdesc
 varchar(50));
```

以下の例では、COPY コマンドのデータソースは、`redshift-downloads`という名前の Amazon S3 バケットの `tickit` フォルダ内の `category_pipe.txt` というデータファイルです。COPY コマンドには、AWS Identity and Access Management(IAM) ロールを通して、Amazon S3 バケットにアクセスすることが許可されています。クラスターに Amazon S3 にアクセスする権限を持つ既存の IAM ロールがある場合、次の COPY コマンドに ロールの Amazon Resource Name (ARN) を置換して実行できます。

```
copy catdemo
from 's3://redshift-downloads/tickit/category_pipe.txt'
iam_role 'arn:aws:iam::<aws-account-id>:role/<role-name>'
region 'us-east-1';
```

他の AWS リージョンからデータをロードする手順など、COPY コマンドを使用してサンプルデータをロードする方法の詳細については、「Amazon Redshift 入門ガイド」の「Amazon S3 からサンプルデータをロードする」を参照してください。

table-name

COPY コマンドのターゲットテーブル名です。テーブルはすでにデータベースに存在する必要があります。テーブルは一時テーブルまたは永続的テーブルです。COPY コマンドは、新しい入力データをテーブルの既存の行に追加します。

FROM data_source

ターゲットテーブルにロードするソースデータの場所です。マニフェストファイルは、いくつかのデータソースで指定できます。

最もよく使われるデータリポジトリは Amazon S3 バケットです。Amazon EMR クラスター、Amazon EC2 インスタンス、またはクラスターが SSH 接続を使用してアクセスできるリモートホストにあるデータファイルからロードすることもできます。または、DynamoDB テーブルから直接ロードすることもできます。

- Amazon S3 からの COPY
- Amazon EMR からの COPY
- リモートホスト (SSH) からの COPY

- Amazon DynamoDB からの COPY

承認

他の AWS リソースにアクセスするための承認と許可のために、クラスターが使用する方法を示す句。COPY コマンドが、他の AWS リソース (Amazon S3、Amazon EMR、Amazon DynamoDB、Amazon EC2 など) のデータにアクセスするにためには承認が必要です。クラスターにアタッチされた IAM ロールを参照して、または IAM ユーザーのアクセスキー ID とシークレットアクセスキーを提供して、そのアクセス権限を提供できます。

- 認可パラメータ
- ロールベースアクセスコントロール
- キーベースのアクセスコントロール

任意指定のパラメータ

オプションで、COPY でターゲットテーブルの列にフィールドデータをマッピングする方法の指定、COPY コマンドで正しく読み込み解析できるソースデータ属性の定義、ロード処理中に COPY コマンドが実行する操作の管理ができます。

- 列のマッピングオプション
- データ形式パラメータ
- データ変換パラメータ
- データのロード操作

列のマッピング

デフォルトでは、COPY はデータファイルで発生したフィールドと同じ順序でターゲットテーブルの列にフィールド値を挿入します。デフォルトの列順序が機能しない場合は、列リストを指定するか、JSONPath 式を使用してソースデータフィールドをターゲット列にマッピングできます。

- Column List
- JSONPaths File

データ形式パラメータ

固定幅、文字区切り形式、カンマ区切り値 (CSV) のテキストファイル、JSON 形式、または Avro ファイルからデータをロードできます。

デフォルトでは、COPY コマンドはソースデータを文字区切り形式 UTF-8 のテキストファイルと見なします。デフォルトの区切り文字はパイプ文字です。ソースデータが別の形式である場合は、以下のパラメータを使用してデータ形式を指定します。

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- ENCRYPTED
- BZIP2
- GZIP
- LZOP
- PARQUET
- ORC
- ZSTD

データ変換パラメータ

テーブルをロードする際に、COPY は暗黙的にソースデータの文字列をターゲット列のデータ型に変換しようとします。デフォルトの動作とは異なる変換を指定する必要がある場合、またはデフォルトの変換がエラーになった場合、次のパラメータを指定してデータ変換を管理できます。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL
- ENCODING
- ESCAPE
- EXPLICIT_IDS

- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

データのロード操作

次のパラメータを指定して、トラブルシューティングの際のロード操作のデフォルトの動作を管理したり、ロード時間を短縮します。

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS
- MAXERROR
- NOLOAD
- STATUPDATE

COPY コマンドの使用上の注意とその他のリソース

COPY コマンドの使用方法の詳細については、次のトピックを参照してください。

- 使用に関する注意事項
- チュートリアル: Amazon S3 からデータをロードする
- データをロードするための Amazon Redshift のベストプラクティス
- COPY コマンドを使ってテーブルをロードする
  - Amazon S3 からデータをロードする
  - Amazon EMR からのデータのロード
  - リモートホストからデータをロードする

  - Amazon DynamoDB テーブルからのデータのロード
- データロードのトラブルシューティング

COPY コマンドの例

さまざまなソースから、異なる形式で、さまざまな COPY オプションを使用して COPY を実行する方法を示すその他の例については、「COPY の例」を参照してください。

COPY JOB (プレビュー)

これは、プレビューで使用できる自動コピー (SQL COPY JOB) に関する、プレリリースドキュメントです。 ドキュメントと機能はどちらも変更されることがあります。この機能については、テスト環境のみで使用し、本番環境では使用しないことをお勧めします。パブリックプレビューは 2024 年 10 月 31 日に終了します。プレビュークラスターは、プレビュー終了 2 週間後に自動的に削除されます。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

プレビューでのこのコマンドの使用の詳細については、「Amazon S3 からの継続的なファイル取り込みによるテーブルのロード (プレビュー)」を参照してください。

データをテーブルにロードする COPY コマンドを管理します。COPY JOB コマンドは COPY コマンドの拡張であり、Amazon S3 バケットからのデータロードを自動化します。COPY ジョブを作成すると、Amazon Redshift は、指定されたパスに新しい Amazon S3 ファイルが作成されたことを検出し、ユーザーの操作なしで自動的にロードします。データをロードするときには、元の COPY コマンドで使用されているのと同じパラメータが使用されます。Amazon Redshift は、ロードされたファイルを記録して、ロードされたのは 1 回だけであることを確認します。

> **Note**
>
> 使用方法、パラメータ、権限など、COPY コマンドの詳細については、「COPY」を参照してください。

必要なアクセス許可

COPY JOB の COPY コマンドを実行するには、ロードするテーブルの INSERT 権限が必要です。

COPY コマンドで指定した IAM ロールにはロードするデータへのアクセス権限が必要です。詳細については、「COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可」を参照してください。

構文

コピージョブを作成します。COPY コマンドのパラメータは、コピージョブと共に保存されます。

```
COPY copy-command JOB CREATE job-name
[AUTO ON | OFF]
```

コピージョブの設定を変更します。

```
COPY JOB ALTER job-name
[AUTO ON | OFF]
```

コピージョブを実行します。保存されている COPY コマンドパラメータが使用されます。

```
COPY JOB RUN job-name
```

すべてのコピージョブを一覧表示します。

```
COPY JOB LIST
```

コピージョブの詳細を表示します。

```
COPY JOB SHOW job-name
```

コピージョブを削除します。

```
COPY JOB DROP job-name
```

パラメータ

copy-command

Amazon S3 から Amazon Redshift にデータをロードする COPY コマンドです。この句には、Amazon S3 バケット、ターゲットテーブル、IAM ロール、およびデータをロードするとき

に使用されるその他のパラメータを定義する COPY パラメータが含まれています。Amazon S3 データロードの COPY コマンドパラメータは、以下を除いてすべてサポートされています。

- COPY JOB は、COPY コマンドで指定されたフォルダー内の既存のファイルを取り込みません。COPY JOB 作成タイムスタンプ以降に作成されたファイルのみが取り込まれます。
- COPY コマンドに対しては、MAXERROR オプションまたは IGNOREALLERRORS オプションを指定することはできません。
- マニフェストファイルは指定できません。COPY JOB では、新しく作成されたファイルをモニタリングするために、指定された Amazon S3 ロケーションが必要です。
- アクセスキーやシークレットキーなどの認可タイプで COPY コマンドを指定することはできません。この `IAM_ROLE` パラメータを認可に使用する COPY コマンドのみがサポートされます。詳細については、「認可パラメータ」を参照してください。
- COPY JOB は、クラスターに関連付けられたデフォルトの IAM ロールをサポートしていません。COPY コマンド内で `IAM_ROLE` を指定する必要があります。

詳細については、「Amazon S3 からの COPY」を参照してください。

job-name

COPY ジョブを参照するために使用されるジョブの名前です。

[AUTO ON | OFF]

Amazon S3 データが Amazon Redshift テーブルに自動的にロードされるかどうかを示す句です。

- `ON` の場合、Amazon Redshift はソースの Amazon S3 パスで新しく作成されたファイルをモニタリングし、見つかった場合は、ジョブ定義の COPY パラメータを使用して COPY コマンドが実行されます。これがデフォルトです。
- `OFF` の場合、Amazon Redshift は COPY JOB を自動的に実行しません。

使用に関する注意事項

COPY コマンドのオプションは実行時まで検証されません。たとえば、無効な `IAM_ROLE` や Amazon S3 データソースがあると、COPY JOB の開始時にランタイムエラーが発生します。

クラスターが一時停止している場合、COPY JOB は実行されません。

ロードされた COPY コマンドファイルとロードエラーをクエリするには「STL_LOAD_COMMITS、STL_LOAD_ERRORS、STL_LOADERROR_DETAIL」を参照してください。詳細については、「データが正しくロードされたことを確認する」を参照してください。

例

次の例では、Amazon S3 バケットからデータをロードするための COPY JOB を作成しています。

```
COPY public.target_table
FROM 's3://amzn-s3-demo-bucket/staging-folder'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyLoadRoleName'
JOB CREATE my_copy_job_name
AUTO ON;
```

COPY パラメータリファレンス

COPY には、さまざまな状況で使用できるパラメータが多数あります。ただし、すべてのパラメータがそれぞれの状況でサポートされているわけではありません。例えば、ORC ファイルや PARQUET ファイルからロードする場合、サポートされるパラメーターの数は限られています。詳細については、「列データ形式の COPY」を参照してください。

トピック

- データソース
- 認可パラメータ
- 列のマッピングオプション
- データ形式パラメータ
- ファイル圧縮パラメータ
- データ変換パラメータ
- データのロード操作
- パラメータリスト (アルファベット順)

データソース

Amazon S3 バケット、Amazon EMR クラスター、またはクラスターが SSH 接続を使用してアクセスできるリモートホストのテキストファイルからデータをロードできます。また、DynamoDB テーブルから直接データをロードすることもできます。

どのソースからであっても、単一の入力行の最大サイズは 4 MB です。

テーブルから Amazon S3 の一連のファイルにデータをエクスポートするには、UNLOAD コマンドを使用します。

トピック

- [Amazon S3 からの COPY]
- [Amazon EMR からの COPY]
- [リモートホスト (SSH) からの COPY]
- [Amazon DynamoDB からの COPY]

Amazon S3 からの COPY

S3 バケットに配置したファイルからデータをロードするには、FROM 句を使用して COPY が Amazon S3 にあるファイルを見つける方法を指定します。FROM 句の一部としてデータファイルのオブジェクトパスを指定できます。または、Amazon S3 オブジェクトパスのリストを含むマニフェストファイルの場所を指定できます。Amazon S3 からの COPY では、HTTPS 接続が使用されます。S3 IP 範囲が許可リストに追加されていることを確認します。必要な S3 IP 範囲の詳細については、「ネットワークの隔離」を参照してください。

> **Important**
>
> データファイルを保持する Amazon S3 バケットがクラスターと同じ AWS リージョンに存在しない場合は、REGIONパラメータを使用して、データがあるリージョンを指定する必要があります。

トピック

- 構文
- 例
- 任意指定のパラメータ
- サポートされないパラメータ

構文

```
FROM { 's3://objectpath' | 's3://manifest_file' }
authorization
| MANIFEST
| ENCRYPTED
| REGION [AS] 'aws-region'
```

```
| optional-parameters
```

例

次の例では、オブジェクトパスを使用して Amazon S3 からデータをロードします。

```
copy customer
from 's3://amzn-s3-demo-bucket/customer'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

次の例では、マニフェストファイルを使用して Amazon S3 からデータをロードします。

```
copy customer
from 's3://amzn-s3-demo-bucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

パラメータ

FROM

ロードするデータのソースです。Amazon S3 ファイルのエンコードの詳細については、データ変換パラメータ を参照してください。

's3://copy_from_s3_objectpath'

データを含む Amazon S3 オブジェクトへのパスを指定します (例: `'s3://amzn-s3-demo-bucket/custdata.txt'`)。s3://copy_from_s3_objectpath パラメータは、1 つのファイルを参照することも、同じキープレフィックスを持つオブジェクトまたはフォルダの集合を参照することもできます。たとえば、名前 `custdata.txt` は `custdata.txt`、`custdata.txt.1`、`custdata.txt.2`、`custdata.txt.bak` など、複数の物理ファイルを参照するキープレフィックスです。キープレフィックスは複数のフォルダを参照することもできます。たとえば、`'s3://amzn-s3-demo-bucket/custfolder'` は `custfolder` フォルダや `custfolder_1` フォルダなどを参照します。`custfolder_2`キープレフィックスが複数のフォルダを参照する場合、フォルダ内のすべてのファイルがロードされます。キープレフィックスが `custfolder.log` などのフォルダだけでなくファイルとも一致する場合、COPY はファイルのロードも試みます。キープレフィックスが原因で COPY が不要なファイルをロードしようとした場合は、マニフェストファイルを使用します。詳細については、copy_from_s3_manifest_fileを参照してください。

> **Important**
>
> データファイルを保持する S3 バケットがクラスターと同じ AWS リージョンに存在しない場合は、REGIONパラメータを使用して、データがあるリージョンを指定する必要があります。

詳細については、「Amazon S3 からデータをロードする」を参照してください。

's3://copy_from_s3_manifest_file'

ロードするデータファイルをリストするマニフェストファイルの Amazon S3 オブジェクトキーを指定します。's3://copy_from_s3_manifest_file' 引数は、単一のファイル (`'s3://amzn-s3-demo-bucket/manifest.txt'` など) を明示的に参照する必要があります。キープレフィックスを参照することはできません。

マニフェストは、Amazon S3 からロードする各ファイルの URL をリストする、JSON 形式のテキストファイルです。URL にはバケット名およびファイルの完全オブジェクトパスが含まれます。マニフェストで指定するファイルの場所は異なるバケットでもかまいませんが、すべてのバケットは Amazon Redshift クラスターと同じ AWS リージョンに置かれている必要があります。ファイルが 2 回リストされている場合、ファイルは 2 回ロードされます。次の例は、3 つのファイルをロードするマニフェストの JSON を示しています。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1","mandatory":false}
  ]
}
```

二重引用符は必須であり、傾きの付いた "高機能な" 引用符ではなくシンプルな引用符 (0x22) にする必要があります。マニフェストの各エントリには、オプションとして `mandatory` フラグを含めることができます。`mandatory` が `true` に設定されている場合、そのエントリのファイルが見つからなければ、COPY は終了します。ファイルが見つかれば、COPY 処理は継続します。`mandatory` のデフォルト値は `false` です。

以下の例に示すように、Parquet または ORC 形式のデータファイルからロードする場合、`meta`フィールドは必須です。

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket1/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://amzn-s3-demo-bucket2/orc/2013-10-05-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

ENCRYPTED、GZIP、LZOP、BZIP2、または ZSTD オプションを指定している場合でも、マニフェストファイルの暗号化または圧縮は行わないでください。指定したマニフェストファイルが見つからないか、マニフェストファイルの形式が適切ではない場合、COPY はエラーを返します。

マニフェストファイルを使用する場合、COPY コマンドに MANIFEST パラメータを指定する必要があります。MANIFEST パラメータを指定しない場合、COPY では、FROM で指定されたファイルがデータファイルであると想定します。

詳細については、「Amazon S3 からデータをロードする」を参照してください。

authorization

COPY コマンドが、他の AWS リソース (Amazon S3、Amazon EMR、Amazon DynamoDB、Amazon EC2 など) のデータにアクセスするためには承認が必要です。この認可を付与するには、クラスターにアタッチした AWS Identity and Access Management (IAM) ロールを参照 (ロールベースのアクセスコントロール) するか、ユーザーのアクセス認証情報を指定 (キーベースのアクセスコントロール) します。セキュリティと柔軟性を強化するために、IAM ロールベースのアクセスコントロールを使用することをお勧めします。詳細については、「認可パラメータ」を参照してください。

MANIFEST

Amazon S3 からロードするデータファイルの識別にマニフェストを使用することを指定します。MANIFEST パラメータが使用されている場合、COPY は 's3://copy_from_s3_manifest_file' によって参照されるマニフェストに記載されているファイルからデータをロードします。マニフェストファイルが見つからない場合、または形式が正しくない場合、COPY は失敗します。詳細については、「マニフェストを使用し、データファイルを指定する」を参照してください。

ENCRYPTED

Amazon S3 にある入力ファイルの暗号化が、カスタマーマネージドのキーを使用したクライアント側の暗号化であることを指定する句。詳細については、「暗号化されたデータファイルを Amazon S3 からロードする」を参照してください。入力ファイルが Amazon S3 サーバー側の暗号化 (SSE-KMS または SSE-S3) を使用して暗号化されている場合は、ENCRYPTED を指定しないでください。COPY では、サーバー側で暗号化されたファイルを自動的に読み取ります。

ENCRYPTED パラメータを指定する場合は、MASTER_SYMMETRIC_KEYパラメータも指定するか、**master_symmetric_key**値を CREDENTIALS 文字列に含める必要があります。

暗号化されたファイルが圧縮形式である場合は、GZIP、LZOP、BZIP2、ZSTD パラメータを追加してください。

ENCRYPTED を指定している場合でも、マニフェストファイルと JSONPaths ファイルの暗号化は行わないでください。

MASTER_SYMMETRIC_KEY 'root_key'

Amazon S3 のデータファイルの暗号化に使用されたルート対称キー。MASTER_SYMMETRIC_KEY を指定する場合、ENCRYPTEDパラメータも指定する必要があります。MASTER_SYMMETRIC_KEY は CREDENTIALS パラメータと併用できません。詳細については、「暗号化されたデータファイルを Amazon S3 からロードする」を参照してください。

暗号化されたファイルが圧縮形式である場合は、GZIP、LZOP、BZIP2、ZSTD パラメータを追加してください。

REGION [AS] 'aws-region'

ソースデータが配置されている AWS のリージョンを指定します。REGION は、データを含む AWS のリソースが Amazon Redshift クラスターと同じリージョンにない場合に、Amazon S3 バケットまたは DynamoDB テーブルから COPY を実行するために必要となります。

aws_region の値は、Amazon Redshift リージョンとエンドポイントテーブルに示されているリージョンと一致している必要があります。

REGION パラメータが指定されている場合、マニフェストファイルや複数の Amazon S3 バケットを含むすべてのリソースが指定されたリージョンに存在している必要があります。

> **Note**
>
> リージョン間でデータを転送する場合、Amazon S3 バケットやデータを含む DynamoDB テーブルに対して追加料金が発生します。料金の詳細については、「Amazon S3 の料金」ページの別の AWS リージョンへの「Amazon S3 からのデータ転送 (アウト)」、および「Amazon DynamoDB の料金」ページの「データ転送 (アウト)」を参照してください。

デフォルトでは、COPY はデータが Amazon Redshift クラスターと同じリージョンにあると見なします。

任意指定のパラメータ

Amazon S3 からの COPY では、オプションで次のパラメータを指定できます。

- 列のマッピングオプション
- データ形式パラメータ
- データ変換パラメータ
- データのロード操作

サポートされないパラメータ

Amazon S3 からの COPY では、次のパラメータは使用できません。

- SSH
- READRATIO

Amazon EMR からの COPY

COPY コマンドを使用することで、クラスターの Hadoop Distributed File System (HDFS) に、固定幅ファイル、文字区切りファイル、CSV ファイル、JSON 形式ファイル、または Avro ファイルでテ

キストファイルを書き込むように設定された Amazon EMR クラスターから、データを並列にロードできます。

トピック

- Syntax
- 例
- パラメータ
- サポートされているパラメータ
- サポートされないパラメータ

Syntax

```
FROM 'emr://emr_cluster_id/hdfs_filepath'
authorization
[ optional_parameters ]
```

例

次の例では、Amazon EMR クラスターからデータをロードします。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

パラメータ

FROM

ロードするデータのソースです。

'emr://emr_cluster_id/hdfs_file_path'

Amazon EMR クラスターの一意の識別子、および COPY コマンドのデータファイルを参照する HDFS ファイルパスです。HDFS データファイル名には、ワイルドカード文字のアスタリスク (*) および疑問符 (?) を含めることはできません。

> **Note**
>
> Amazon EMR クラスターは、COPY 操作が完了するまで稼動している必要があります。COPY 操作が完了する前に HDFS データファイルのいずれかが変更または削除されると、予期しない結果を招いたり、COPY 操作が失敗したりする可能性があります。

ファイル名の引数には、ロードする複数のファイルを指定する hdfs_file_path 引数の一部としてアスタリスク (*) および疑問符 (?) を使用できます。たとえば、`'emr://j-SAMPLE2B500FC/myoutput/part*'`であれば、`part-0000`、`part-0001`などのファイルが識別されます。ファイルパスにワイルドカード文字が含まれていない場合は、文字列リテラルとして処理されます。COPY コマンドでフォルダー名のみを指定した場合には、フォルダー内のすべてのファイルがロードされます。

> **Important**
>
> ワイルドカード文字を使用する場合、またはフォルダ名のみを使用する場合は、不要なファイルがロードされないことを確認してください。例えば、一部のプロセスでは出力フォルダにログファイルが書き込まれることがあります。

詳細については、「Amazon EMR からのデータのロード」を参照してください。

authorization

COPY コマンドが、他の AWS リソース (Amazon S3、Amazon EMR、Amazon DynamoDB、Amazon EC2 など) のデータにアクセスするにためには承認が必要です。この認可を付与するには、クラスターにアタッチした AWS Identity and Access Management (IAM) ロールを参照 (ロールベースのアクセスコントロール) するか、ユーザーのアクセス認証情報を指定 (キーベースのアクセスコントロール) します。セキュリティと柔軟性を強化するために、IAM ロールベースのアクセスコントロールを使用することをお勧めします。詳細については、「認可パラメータ」を参照してください。

サポートされているパラメータ

Amazon EMR からの COPY では、オプションで次のパラメータを指定できます。

- 列のマッピングオプション
- データ形式パラメータ

- データ変換パラメータ
- データのロード操作

サポートされないパラメータ

Amazon EMR からの COPY では、次のパラメータは使用できません。

- ENCRYPTED
- MANIFEST
- REGION
- READRATIO
- SSH

リモートホスト (SSH) からの COPY

COPY コマンドでは、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスをはじめとするコンピュータなど、1 つ以上のリモートホストから同時にデータをロードすることができます。COPY では Secure Shell (SSH) を使用してリモートホストに接続し、そのホストのコマンドを実行してテキスト出力を生成します。リモートホストになることができるのは、EC2 の Linux インスタンスか、SSH 接続を許可するように設定されている Unix コンピュータまたは Linux コンピュータです。Amazon Redshift は複数のホストに接続でき、各ホストに対して複数の SSH 接続を開くことができます。Amazon Redshift は、各接続を介して一意のコマンドを送信し、ホストの標準出力にテキスト出力を生成します。Amazon Redshift は、テキストファイルと同じようにそれを読み込みます。

FROM 句を使用してマニフェストファイルの Amazon S3 オブジェクトキーを指定します。そのマニフェストファイルは、COPY が SSH 接続を開いてリモートコマンドを実行するために使用する情報を提供します。

トピック

- 構文
- 例
- パラメータ
- 任意指定のパラメータ
- サポートされないパラメータ

> **Important**
>
> マニフェストファイルを保持する S3 バケットがクラスターと同じ AWS リージョンに存在しない場合は、REGION パラメータを使用して、バケットがあるリージョンを指定する必要があります。

構文

```
FROM 's3://'ssh_manifest_file' }
authorization
SSH
| optional-parameters
```

例

次の例では、マニフェストファイルを使用し、SSH を使用してリモートホストからデータをロードします。

```
copy sales
from 's3://amzn-s3-demo-bucket/ssh_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
ssh;
```

パラメータ

FROM

ロードするデータのソースです。

's3://copy_from_ssh_manifest_file'

COPY コマンドは、SSH を使用して複数のホストに接続できるだけでなく、各ホストに対して複数の SSH 接続を作成できます。COPY はそれぞれのホスト接続を介してコマンドを実行し、コマンドからの出力を並列的にテーブルにロードします。s3://copy_from_ssh_manifest_file 引数は、マニフェストファイルの Amazon S3 オブジェクトキーを指定します。そのマニフェストファイルは、COPY が SSH 接続を開いてリモートコマンドを実行するために使用する情報を提供します。

s3://copy_from_ssh_manifest_file 引数は 1 つのファイルを明示的に参照する必要があります。これをキープレフィックスにすることはできません。例を以下に示します。

```
's3://amzn-s3-demo-bucket/ssh_manifest.txt'
```

マニフェストファイルは、Amazon Redshift がホストに接続する際に使用する JSON 形式のテキストファイルです。マニフェストファイルでは、SSH ホストのエンドポイント、ならびに、Amazon Redshift にデータを返すためにホストで実行されるコマンドを指定します。このほか、ホストのパブリックキー、ログインユーザー名、および各エントリの必須フラグを記載することもできます。次の例は、2 つの SSH 接続を作成するマニフェストファイルを示しています。

```
{
    "entries": [
     {"endpoint":"<ssh_endpoint_or_IP>",
           "command": "<remote_command>",
           "mandatory":true,
           "publickey": "<public_key>",
           "username": "<host_user_name>"},
     {"endpoint":"<ssh_endpoint_or_IP>",
           "command": "<remote_command>",
           "mandatory":true,
           "publickey": "<public_key>",
           "username": "<host_user_name>"}
     ]
}
```

マニフェストファイルには、SSH 接続ごとに 1 つずつ `"entries"` 構造が含まれます。単一のホストに対して接続を複数作成することも、複数のホストに対して複数の接続を作成することもできます。例に示すように、フィールド名と値のどちらにも二重引用符が必要です。引用符の文字は、傾きの付いた "高機能な" 引用符ではなくシンプルな引用符 (0x22) にする必要があります。`"mandatory"` フィールドの中で二重引用符を必要としない値は、`true` または `false` のブール値のみです。

次のリストでは、マニフェストファイルのフィールドについて説明します。

endpoint

　ホストの URL アドレスまたは IP アドレス (`"ec2-111-222-333.compute-1.amazonaws.com"` や `"198.51.100.0"` など)。

コマンド

　テキスト出力またはバイナリ出力を gzip、lzop、bzip2、zstd 形式で生成する際にホストが実行するコマンド。コマンドは、ユーザー "host_user_name" が実行権限を持つコマンドであ

れば、どれでも指定できます。ファイルを印刷するなどのシンプルなコマンドでも、データベースにクエリを実行したり、スクリプトを実行したりするコマンドでもかまいません。出力 (テキストファイル、gzip バイナリファイル、lzop バイナリファイル、または bzip2 バイナリファイル) は、Amazon Redshift の COPY コマンドが取り込める形式にする必要があります。詳細については、「入力データを準備する」を参照してください。

publickey

(オプション) ホストのパブリックキー。公開キーが指定されている場合、Amazon Redshift は公開キーを使用してホストを特定します。公開キーが指定されていなければ、Amazon Redshift がホストの特定を試みることはありません。例えば、リモートホストのパブリックキーが `ssh-rsa AbcCbaxxx…Example root@amazon.com` であれば、パブリックキーのフィールドには `"AbcCbaxxx…Example"` と入力してください。

mandatory

(オプション) 接続ができなかった場合に COPY コマンドを失敗とするかどうかを示す句です。デフォルト: `false`。Amazon Redshift が接続を 1 つも正常に確立できなかった場合に、COPY コマンドが失敗になります。

username

(オプション) ホストシステムにログオンし、リモートコマンドを実行する際に使用するユーザー名。ユーザーログイン名は、ホストの認可されたキーファイルに Amazon Redshift クラスターの公開キーを追加するときに使用したログイン名と同じものにする必要があります。デフォルトのユーザー名は `redshift` です。

マニフェストファイルの作成の詳細については、「データをロードする手順」を参照してください。

リモートホストから COPY を実行するには、COPY コマンドに SSH パラメータを指定する必要があります。SSH パラメータを指定しない場合、COPY では、FROM で指定されたファイルがデータファイルであると想定され、COPY は失敗します。

自動圧縮を使用する場合には、COPY コマンドでデータの読み込みオペレーションが 2 回実行されます。つまり、COPY コマンドではリモートコマンドが 2 回実行されることになります。初回の読み取り操作は圧縮の分析用データサンプルを提供するためのものであり、実際にデータがロードされるのは 2 回目の読み取り操作です。リモートコマンドを 2 回実行することが問題になるようであれば、自動圧縮は無効にする必要があります。自動圧縮を無効にするには、COMPUPDATE パラメータを OFF に設定して COPY コマンドを実行します。詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

SSH から COPY を使用するための詳細な手順については、「リモートホストからデータをロードする」を参照してください。

authorization

COPY コマンドが、他の AWS リソース (Amazon S3、Amazon EMR、Amazon DynamoDB、Amazon EC2 など) のデータにアクセスするためには承認が必要です。この認可を付与するには、クラスターにアタッチした AWS Identity and Access Management (IAM) ロールを参照 (ロールベースのアクセスコントロール) するか、ユーザーのアクセス認証情報を指定 (キーベースのアクセスコントロール) します。セキュリティと柔軟性を強化するために、IAM ロールベースのアクセスコントロールを使用することをお勧めします。詳細については、「認可パラメータ」を参照してください。

SSH

SSH プロトコルを使用してリモートホストからデータがロードされることを指定する句です。SSH を指定する場合は、s3://copy_from_ssh_manifest_file引数を使用してマニフェストファイルを指定する必要もあります。

> **Note**
>
> SSH を使用してリモート VPC でプライベート IP アドレスを使用しているホストからコピーしている場合、VPC は拡張された VPC ルーティングを有効化する必要があります。拡張された VPC ルーティングの詳細については、「Amazon Redshift 拡張 VPC ルーティング」を参照してください。

任意指定のパラメータ

SSH からの COPY では、オプションで次のパラメータを指定できます。

- 列のマッピングオプション
- データ形式パラメータ
- データ変換パラメータ
- データのロード操作

サポートされないパラメータ

SSH からの COPY では、次のパラメータは使用できません。

- ENCRYPTED
- MANIFEST
- READRATIO

Amazon DynamoDB からの COPY

既存の DynamoDB テーブルからデータをロードするには、FROM 句を使用して DynamoDB テーブル名を指定します。

トピック

- 構文
- 例
- 任意指定のパラメータ
- サポートされないパラメータ

> **Important**
>
> DynamoDB テーブルが Amazon Redshift クラスターと同じリージョンに存在しない場合は、REGION パラメータを使用して、データがあるリージョンを指定する必要があります。

構文

```
FROM 'dynamodb://table-name'
authorization
READRATIO ratio
| REGION [AS] 'aws_region'
| optional-parameters
```

例

次の例では、DynamoDB テーブルからデータをロードします。

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

パラメータ

FROM

ロードするデータのソースです。

'dynamodb://table-name'

データが入った DynamoDB テーブルの名前 (`'dynamodb://ProductCatalog'` など)。DynamoDB で Amazon Redshift 列に属性がマッピングされる方法の詳細については、Amazon DynamoDB テーブルからのデータのロードを参照してください。

DynamoDB テーブル名は AWS アカウントに固有のものです。アカウントは AWS アクセス認証情報によって識別されます。

authorization

COPY コマンドには、Amazon S3、Amazon EMR、DynamoDB、Amazon EC2 を含む、別の AWS リソースのデータにアクセスするための許可が必要になります。この認可を付与するには、クラスターにアタッチした AWS Identity and Access Management (IAM) ロールを参照 (ロールベースのアクセスコントロール) するか、ユーザーのアクセス認証情報を指定 (キーベースのアクセスコントロール) します。セキュリティと柔軟性を強化するために、IAM ロールベースのアクセスコントロールを使用することをお勧めします。詳細については、「認可パラメータ」を参照してください。

READRATIO [AS] ratio

データロードに使用する DynamoDB テーブルのプロビジョニングされたスループットの比率です。READRATIO は DynamoDB からの COPY では必須です。Amazon S3 からの COPY の実行には使用できません。この割合については、未使用のプロビジョニングされたスループットの平均よりも小さい値に設定することを強くお勧めします。有効な値は、1～200 の整数です。

> **Important**
>
> READRATIO を 100 以上に設定すると、Amazon Redshift で DynamoDB テーブルのプロビジョニングされたスループット全体を使用できるようになり、COPY セッション中に同じテーブルに対する同時読み込みオペレーションのパフォーマンスが大きく低下します。書き込みトラフィックは影響を受けません。Amazon Redshift がテーブルのプロビジョニングされたスループットを満たさないまれなシナリオをトラブルシューティングする場合には、100 を超える値を使用できます。DynamoDB から Amazon Redshift に

> 継続的にデータをロードする場合、DynamoDB テーブルを時系列テーブルとして編成し、COPY 操作からライブトラフィックを分離することを検討してください。

任意指定のパラメータ

Amazon DynamoDB からの COPY では、オプションで次のパラメータを指定できます。

- 列のマッピングオプション
- 次のデータ変換パラメータがサポートされています。
  - ACCEPTANYDATE
  - BLANKSASNULL
  - DATEFORMAT
  - EMPTYASNULL
  - ROUNDEC
  - TIMEFORMAT
  - TRIMBLANKS
  - TRUNCATECOLUMNS
- データのロード操作

サポートされないパラメータ

DynamoDB からの COPY では、次のパラメータは使用できません。

- 全データ形式パラメータ
- ESCAPE
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL
- REMOVEQUOTES
- ACCEPTINVCHARS
- MANIFEST
- ENCRYPTED

認可パラメータ

COPY コマンドが、他の AWS リソース (Amazon S3、Amazon EMR、Amazon DynamoDB、Amazon EC2 など) のデータにアクセスするためには承認が必要です。この承認は、クラスターにアタッチされた AWS Identity and Access Management (IAM) ロール (ロールベースのアクセスコントロール) を参照することで提供できます。Amazon S3 のロードデータを暗号化できます。

以下のトピックでは、認証オプションの詳細と例をさらに示します。

- COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可
- ロールベースアクセスコントロール
- キーベースのアクセスコントロール

以下のいずれかを使用して COPY コマンドに認可を提供します。

- IAM_ROLE パラメータ
- ACCESS_KEY_ID and SECRET_ACCESS_KEY パラメータ
- CREDENTIALS 句

IAM_ROLE { default | 'arn:aws:iam::*<AWS #####-id>*:role/*<role-name>*' }

デフォルトキーワードを使用して、COPY コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。IAM_ROLE を指定すると、ACCESS_KEY_ID および SECRET_ACCESS_KEY、SESSION_TOKEN、または CREDENTIALS は使用できません。

以下に、IAM_ROLE パラメータの構文を示します。

```
IAM_ROLE { default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' }
```

詳細については、「ロールベースアクセスコントロール」を参照してください。

ACCESS_KEY_ID 'access-key-id ' SECRET_ACCESS_KEY 'secret-access-key'

この認可方法は推奨されません。

> **Note**
>
> アクセス認証情報をプレーンテキストで提供するのではなく、IAM_ROLE パラメータを指定してロールベースの認証を使用することを強くお勧めします。詳細については、「ロールベースアクセスコントロール」を参照してください。

SESSION_TOKEN 'temporary-token'

一時的アクセス認証情報で使用するセッショントークン。SESSION_TOKEN を指定した場合、ACCESS_KEY_ID と SECRET_ACCESS_KEY も使用して一時的アクセスキー認証情報を指定する必要があります。SESSION_TOKEN を指定した場合、IAM_ROLE または CREDENTIALS は使用できません。詳細については、IAM ユーザーガイドの「一時的な認証情報」を参照してください。

> **Note**
>
> 一時的セキュリティ認証情報を作成するのではなく、ロールベースの認証を使用することを強くお勧めします。IAM ロールを使用して認可すると、Amazon Redshift が自動的に各セッション用の一時的ユーザー認証情報を作成します。詳細については、「ロールベースアクセスコントロール」を参照してください。

以下に、ACCESS_KEY_ID と SECRET_ACCESS_KEY パラメータを使用した SESSION_TOKEN パラメータの構文を示します。

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

SESSION_TOKEN を指定した場合、CREDENTIALS または IAM_ROLE は使用できません。

[WITH] CREDENTIALS [AS] 'credentials-args'

クラスターが、データファイルまたはマニフェストファイルを含む他の AWS リソースにアクセスする際の方法を示す句。CREDENTIALS パラメータは、IAM_ROLE または ACCESS_KEY_ID と SECRET_ACCESS_KEY との併用はできません。

> **Note**
>
> 柔軟性を強化するために、CREDENTIALS パラメータの代わりに IAM_ROLE パラメータを使用することをお勧めします。

必要に応じて ENCRYPTED パラメータを使用する場合は、credentials-args 文字列が、暗号化キーを提供します。

credentials-args 文字列では大文字と小文字が区別され、空白を含めることはできません。

キーワード WITH および AS はオプションで、無視されます。

role-based access control または key-based access control のどちらかを指定できます。どちらの場合も、IAM ロールまたはユーザーは、指定された AWS リソースにアクセスするために必要なアクセス許可が必要です。詳細については、「COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可」を参照してください。

> **Note**
>
> AWS 認証情報および機密データを保護するには、ロールベースのアクセスコントロールを使用することを強くお勧めします。

ロールベースのアクセスコントロールを指定するには、次の形式で credentials-args 文字列を指定します。

```
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
```

一時トークン認証情報を使用するには、一時アクセスキー ID、一時秘密アクセスキー、および一時トークンを提供する必要があります。credentials-args 文字列は次の形式になります。

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-
secret-access-key>;token=<temporary-token>'
```

詳細については、「一時的な認証情報」を参照してください。

ENCRYPTED パラメータを使用する場合、credentials-args 文字列は下記のような形式になります。ここで *<root-key>* はファイルの暗号化に使用されたルートキーの値です。

```
CREDENTIALS
'<credentials-args>;master_symmetric_key=<root-key>'
```

例えば、次の COPY コマンドが、暗号化キーを含むロールベースのアクセスコントロールを使用するとします。

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<account-id>:role/<role-name>;master_symmetric_key=<root-key>'
```

次の COPY コマンドでは、暗号化キーを使用したロールベースのアクセスコントロールを出力します。

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>;master_symmetric_key=<root-key>'
```

列のマッピングオプション

デフォルトでは、COPY はデータファイルで発生したフィールドと同じ順序でターゲットテーブルの列に値を挿入します。デフォルトの列順序が機能しない場合は、列リストを指定するか、JSONPath 式を使用してソースデータフィールドをターゲット列にマッピングできます。

- Column List
- JSONPaths File

列リスト

ソースデータフィールドを特定のターゲット列にロードするには、列名のカンマ区切りリストを指定します。COPY ステートメントで列は任意の順序に指定できますが、Amazon S3 バケットなどにあるフラットファイルからロードする場合、ソースデータの順序に一致する必要があります。

Amazon DynamoDB テーブルからロードする場合、順序は関係ありません。COPY コマンドは、DynamoDB テーブルから取得した項目の属性名と、Amazon Redshift テーブルの列名を一致させます。詳細については、「Amazon DynamoDB テーブルからのデータのロード」を参照してください。

列リストの形式は次のとおりです。

```
COPY tablename (column1 [,column2, ...])
```

ターゲットテーブルの列が列リストから削除された場合は、COPY はターゲット列の DEFAULT 式をロードします。

ターゲット列にデフォルトがない場合は、COPY は NULL をロードします。

COPY を実行し、NOT NULL として定義されている列に NULL を割り当てようとすると、COPY コマンドは失敗します。

列リストに IDENTITY 列が含まれている場合、EXPLICIT_IDS も指定する必要があります。IDENTITY 列を省略した場合、EXPLICIT_IDS は指定できません。列リストを指定しない場合、コマンドは、完全で順序正しい列リストが指定されたように動作します。EXPLICIT_IDS も指定しなかった場合、IDENTITY 列は省略されます。

列がGENERATED BY DEFAULT AS IDENTITYで定義されている場合は、コピーできます。値は、指定した値で生成または更新されます。EXPLICIT_IDS オプションは必須ではありません。COPY は IDENTITY のハイウォーターマークを更新しません。詳細については、「GENERATED BY DEFAULT AS IDENTITY」を参照してください。

JSONPaths ファイル

JSON または Avro 形式からデータファイルをロードする場合、COPY は JSON または Avro ソースデータのデータ要素をターゲットテーブルの列に自動的にマッピングします。Avro スキーマのフィールド名をターゲットテーブルまたは列リストの列名に一致させることで、行われます。

場合によっては、列名とフィールド名が一致しないか、データ階層のより深いレベルにマッピングする必要があります。このような場合、明示的に列に JSON または Avro のデータ要素をマッピングするには、JSONPaths ファイルを使用できます。

詳細については、「JSONPaths ファイル」を参照してください。

データ形式パラメータ

デフォルトでは、COPY コマンドはソースデータを文字区切り形式 UTF-8 のテキストと見なします。デフォルトの区切り文字はパイプ文字です。ソースデータが別の形式である場合は、以下のパラメータを使用してデータ形式を指定します。

- FORMAT
- CSV
- DELIMITER
- FIXEDWIDTH
- SHAPEFILE
- AVRO
- JSON
- PARQUET
- ORC

標準データ形式に加えて、COPY は、Amazon S3 から COPY の以下の列データ形式をサポートしています。

- ORC
- PARQUET

列形式の COPY は、特定の制限でサポートされています。詳細については、「列データ形式の COPY」を参照してください。

データ形式パラメータ

FORMAT [AS]

(オプション) データ形式キーワードを識別します。FORMAT 引数については以下で説明します。

CSV [QUOTE [AS] 'quote_character']

入力データで CSV 形式の使用を有効にします。区切り記号、改行文字、およびキャリッジリターンを自動的にエスケープするには、QUOTE パラメータで指定した文字でフィールドを囲みます。デフォルトの引用文字は二重引用符 (") です。フィールド内で引用文字が使用されている場合、引用文字を追加してその文字をエスケープします。例えば、引用文字が二重引用符である場合、文字列 `A "quoted" word` を挿入するには、入力ファイルに文字列 `"A ""quoted"" word"` を含める必要があります。CSV パラメータを使用する場合、デフォルトの区切り記号はカンマ (,) です。DELIMITER パラメータを使用して、異なる区切り記号を指定できます。

フィールドを引用符で囲んだ場合、区切り記号と引用文字との間の空白は無視されます。区切り記号がタブなどの空白文字である場合、その区切り記号は空白として扱われません。

CSV は FIXEDWIDTH、REMOVEQUOTES、または ESCAPE と共に使用することはできません。

QUOTE [AS] 'quote_character'

省略可能。CSV パラメータを使用する場合に引用文字として使用する文字を指定します。デフォルトは二重引用符 (") です。QUOTE パラメータを使用して二重引用符以外の引用文字を定義した場合、フィールド内で二重引用符をエスケープする必要はありません。QUOTE パラメータは CSV パラメータと共にしか使用できません。AS キーワードはオプションです。

DELIMITER [AS] ['delimiter_char']

パイプ文字 (|)、カンマ (,)、タブ (\t)、複数の文字 (|~|) など、入力ファイル内のフィールドを区切るときに使用する文字を指定します。印刷不可の文字がサポートされています。文字は UTF-8 コード単位として 8 進数で表すこともできます。8 進数の場合は、「\ddd」の形式を使用します。「d」は 8 進数 (0 ~ 7) です。デフォルトの区切り記号はパイプ文字 (|) です。ただし、CSV パラメータを使用する場合、デフォルトの区切り記号はカンマ (,) です。AS キーワードはオプションです。DELIMITER と FIXEDWIDTH は併用できません。

FIXEDWIDTH 'fixedwidth_spec'

列を区切り記号で区切らずに各列幅を固定長にしたファイルからデータをロードします。fixedwidth_spec はユーザー定義の列ラベルと列幅を指定する文字列です。列ラベルには、ユーザーの選択に従って、テキスト文字列または整数を指定できます。列ラベルは列名と関係ありません。ラベル/幅のペアの順序はテーブルの列の順序に正確に一致する必要があります。FIXEDWIDTH と CSV または DELIMITER を併用することはできません。Amazon Redshift では、CHAR 列および VARCHAR 列の長さがバイト単位で表されるため、ロードするファイルを準備する際には、指定する列幅がマルチバイト文字のバイナリ長に対応できることを確認してください。詳細については、「文字型」を参照してください。

fixedwidth_spec の形式を次に示します。

```
'colLabel1:colWidth1,colLabel:colWidth2, ...'
```

SHAPEFILE [SIMPLIFY [AUTO] ['tolerance']]

入力データで SHAPEFILE 形式の使用を有効にします。デフォルトでは、シェープファイルの最初の列は `GEOMETRY` 列または `IDENTITY` 列のいずれかです。後続のすべての列は、シェープファイルで指定された順序に従います。

SHAPEFILE を FIXEDWIDTH、EMOVEQUOTES、または ESCAPE と一緒に使用することはできません。

`COPY FROM SHAPEFILE` で `GEOGRAPHY` オブジェクトを使用するには、まずオブジェクトを `GEOMETRY` 列に取り込んだ後に、そのオブジェクトを `GEOGRAPHY` オブジェクトにキャストします。

SIMPLIFY [tolerance]

(オプション) Ramer-Douglas-Peucker アルゴリズムと指定された許容値を使用して、取り込みプロセス中のすべてのジオメトリを簡略化します。

SIMPLIFY AUTO [tolerance]

(オプション) 最大ジオメトリのサイズより大きいジオメトリのみを簡略化します。この簡略化では、Ramer-Douglas-Peucker アルゴリズムと、指定した許容値を超えない場合に自動的に計算された許容値が使用されます。このアルゴリズムは、指定された許容値内でオブジェクトを保存するためのサイズを計算します。許容値は任意の値です。

シェープファイルのロードの例については、「シェープファイルを Amazon Redshift にロードする」を参照してください。

AVRO [AS] 'avro_option'

ソースデータが Avro 形式であることを指定します。

Avro 形式はここに挙げるサービスおよびプロトコルから実行する COPY でサポートされます。

- Amazon S3
- Amazon EMR
- リモートホスト (SSH)

Avro は DynamoDB から実行する COPY ではサポートされません。

Avro はデータのシリアル化プロトコルです。Avro のソースファイルには、データ構造を定義するスキーマが含まれています。Avro のスキーマ型は `record` である必要があります。COPY は、デフォルトの非圧縮コーデック、および `deflate` と `snappy` の圧縮コーデックを使用して作成された、Avro ファイルを受け入れます。Avro に関する詳細については、「Apache Avro」を参照してください。

avro_option の有効な値は次のとおりです。

- `'auto'`
- `'auto ignorecase'`

- 's3://*jsonpaths_file*'

デフォルト: 'auto'。

COPY は Avro ソースデータのデータ要素をターゲットテーブルの列に自動的にマッピングします。Avro スキーマのフィールド名をターゲットテーブルの列名に一致させることで、行われます。一致で、'auto'では大文字と小文字が区別され、'auto ignorecase'では大文字と小文字が区別されません。

Amazon Redshift テーブルの列名は常に小文字になるため、'auto' オプションを使用する場合、対応するフィールド名も小文字である必要があります。フィールド名がすべて小文字でない場合は、'auto ignorecase'オプションを使用できます。デフォルトの 'auto' 引数を使用すると、COPY は構造内の最初のレベルのフィールドまたは外部フィールドのみを認識します。

列名を Avro フィールド名に明示的にマップするには、JSONPaths ファイル を使用できます。

デフォルトでは、COPY はターゲットテーブルのすべての列が Avro のフィールド名に一致するように試みます。列のサブセットをロードするには、オプションで列リストを指定できます。ターゲットテーブルの列が列リストから削除された場合は、COPY はターゲット列の DEFAULT 式をロードします。ターゲット列にデフォルトがない場合は、COPY は NULL をロードしようとします。列が列リストに含まれており、COPY が Avro データで一致するフィールドを見つけることができなかった場合、COPY はその列に NULL をロードしようと試みます。

COPY を実行し、NOT NULL として定義されている列に NULL を割り当てようとすると、COPY コマンドは失敗します。

Avro スキーマ

Avro のソースデータファイルには、データ構造を定義するスキーマが含まれています。COPY は Avro のソースデータファイルの一部であるスキーマを読み込み、ターゲットテーブルの列にデータ要素をマッピングします。次の例で Avro のスキーマを示します。

```
{
    "name": "person",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "guid", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"}]
```

```
}
```

Avro スキーマは JSON 形式を使用して定義されています。最上位の JSON オブジェクトには、名前または キーの付いた 3 つの名前と値のペア `"name"`、`"type"`、および `"fields"` があります。

オブジェクトの配置を含む `"fields"` キーペアは、データ構造の各フィールド名とデータ型を定義します。デフォルトでは、COPY はフィールド名を列名に自動的に一致させます。列名は常に小文字であるため、`'auto ignorecase'` オプションを指定しない限り、一致させるフィールド名もまた小文字にする必要があります。列名と一致しないフィールド名はすべて無視されます。順序は関係ありません。前の例では、COPY は列名 `id`、`guid`、`name`および `address` にマッピングしています。

デフォルトの `'auto'` 引数を使用すると、COPY は列の第 1 レベルのオブジェクトのみと一致します。スキーマのより深いレベルにマッピングする場合、またはフィールド名と列名が一致しない場合は、JSONPaths ファイルを使用してマッピングを定義します。詳細については、「JSONPaths ファイル」を参照してください。

キーに関連付けられた値が Avro の複雑なデータ型 (バイト、配列、レコード、マップ、リンクなど) である場合、COPY は値を文字列としてロードします。ここで、文字列はデータの JSON 表現です。COPY は Avro ENUM データ型を文字列としてロードします。文字列の内容は型名です。例については、「JSON 形式からの COPY」を参照してください。

スキーマおよびファイルメタデータを含む Avro ファイルヘッダーの最大サイズは 1 MB です。

単一の Avro データブロックの最大サイズは 4 MB です。これは、行の最大サイズとは異なります。単一の Avro データブロックの最大サイズを超えた場合は、最終的な行サイズが 4 MB 未満であっても COPY コマンドは失敗します。

行のサイズを計算する際、Amazon Redshift では、パイプ文字 (|) を 2 回内部的にカウントします。入力データに非常に多くのパイプ文字が含まれる場合、データブロックが 4 MB 未満であっても行サイズが 4 MB を超えることは可能です。

JSON [AS] 'json_option'

ソースデータは JSON 形式です。

JSON 形式は次のサービスおよびプロトコルからの COPY でサポートされています。

- Amazon S3

- Amazon EMR からの COPY
- SSH からの COPY

JSON は DynamoDB からの COPY ではサポートされません。

json_option の有効な値は次のとおりです。

- `'auto'`
- `'auto ignorecase'`
- `'s3://jsonpaths_file'`
- `'noshred'`

デフォルト: `'auto'`。Amazon Redshift は、JSON ドキュメントのロード中に JSON 構造の属性を複数の列に細分化しません。

デフォルトでは、COPY はターゲットテーブルのすべての列を JSON のフィールド名キーに一致させるように試みます。列のサブセットをロードするには、オプションで列リストを指定できます。JSON フィールド名のキーがすべて小文字でない場合は、`'auto ignorecase'` オプションまたは JSONPaths ファイル を使用して、明示的に列名を JSON フィールド名のキーにマッピングすることができます。

ターゲットテーブルの列が列リストから削除された場合は、COPY はターゲット列の DEFAULT 式をロードします。ターゲット列にデフォルトがない場合は、COPY は NULL をロードしようとします。列が列リストに含まれており、COPY が JSON データで一致するフィールドを見つけることができなかった場合、COPY はその列に NULL をロードしようと試みます。

COPY を実行し、NOT NULL として定義されている列に NULL を割り当てようとすると、COPY コマンドは失敗します。

COPY は、JSON ソースデータ内のデータ要素をターゲットテーブル内の列にマッピングします。これは、ソースの名前と値のペアのオブジェクトキーまたは名前を、ターゲットテーブルの列の名前と一致させることによって行われます。

各 json_option 値については、次の詳細を参照してください。

'auto'

> このオプションでは、一致では大文字と小文字が区別されます。Amazon Redshift テーブルの列名は常に小文字になるため、`'auto'` オプションを使用する場合、対応する JSON フィールド名も小文字である必要があります。

'auto ignorecase'

このオプションでは、一致では大文字と小文字は区別されません。Amazon Redshift テーブルの列名は常に小文字であるため、`'auto ignorecase'` オプションを使用する場合、対応する JSON フィールド名は小文字、大文字、または大文字と小文字を混在させることができます。

's3://jsonpaths_file'

このオプションでは、COPY は指定された JSONPaths ファイルを使用して、JSON ソースデータ内のデータ要素をターゲットテーブル内の列にマッピングします。*s3://jsonpaths_file* 引数は、単一のファイルを明示的に参照する Amazon S3 オブジェクトキーである必要があります。例:「`'s3://amzn-s3-demo-bucket/jsonpaths.txt`」。引数をキープレフィックスにすることはできません。JSONPaths ファイルの使用方法の詳細については、「the section called “JSONPaths ファイル”」を参照してください。

場合によっては、`jsonpaths_file` で指定されたファイルのプレフィックスが、`copy_from_s3_objectpath` で指定されたデータファイルのパスと同じになります。その場合、COPY は JSONPaths ファイルをデータファイルとして読み込み、エラーを返します。たとえば、データファイルがオブジェクトパス `s3://amzn-s3-demo-bucket/my_data.json` を使用し、JSONPaths ファイルが `s3://amzn-s3-demo-bucket/my_data.jsonpaths` であるとします。この場合、COPY は `my_data.jsonpaths` をデータファイルとしてロードしようとします。

'noshred'

このオプションを使用すると、Amazon Redshift は JSON ドキュメントのロード中に JSON 構造の属性を複数の列に細分化しません。

JSON データファイル

JSON データファイルには、一連のオブジェクトまたは配列が含まれます。COPY は、それぞれの JSON オブジェクトまたは JSON 配列をターゲットテーブルの 1 つの行にロードします。行に対応する各オブジェクトまたは配列は、スタンドアロンのルートレベル構造である必要があります。つまり、別の JSON 構造のメンバーではない必要があります。

JSON オブジェクトの先頭と末尾には中括弧 ({ }) が付き、順序が設定されていない一連の名前と値のペアが含まれます。ペアの名前と値はコロンで区切られ、各ペアはカンマで区切られます。デフォルトでは、名前と値のペアに含まれるオブジェクトキー (名前) は、テーブル内の対応する列の名前に一致する必要があります。Amazon Redshift テーブルの列名は常に小文字であるため、一致す

る JSON フィールド名のキーも小文字である必要があります。列名と JSON キーが一致しない場合、the section called “JSONPaths ファイル”を使用して明示的に列をキーにマッピングします。

JSON オブジェクト内の順序は問題ではありません。列名と一致しない名前はすべて無視されます。次に、簡単な JSON オブジェクトの構造を示します。

```
{
  "column1": "value1",
  "column2": value2,
  "notacolumn" : "ignore this value"
}
```

JSON 配列の先頭と末尾には角括弧 ([ ]) が付き、順序が設定された一連のカンマ区切りの値が含まれます。データファイルで配列を使用している場合は、JSONPaths ファイルを指定して、値を列に一致させる必要があります。次に、簡単な JSON 配列の構造を示します。

```
["value1", value2]
```

JSON は正しい形式になっている必要があります。例えば、オブジェクトまたは配列をカンマまたは空白以外の他の文字で区切ることはできません。文字列は、二重引用文字で囲む必要があります。引用符は、傾きの付いた "高機能な" 引用符ではなくシンプルな引用符 (0x22) にする必要があります。

1 つの JSON オブジェクトまたは JSON 配列の最大サイズ (中括弧または角括弧を含む) は 4 MB です。これは、行の最大サイズとは異なります。単一の JSON オブジェクトまたは配列の最大サイズを超えた場合は、最終的な行サイズが 4 MB 未満であっても COPY コマンドは失敗します。

行のサイズを計算する際、Amazon Redshift では、パイプ文字 (|) を 2 回内部的にカウントします。入力データに非常に多くのパイプ文字が含まれる場合、オブジェクトサイズが 4 MB 未満であっても行サイズが 4 MB を超えることは可能です。

COPY は改行文字として `\n` を、タブ文字として `\t` をロードします。バックスラッシュをロードするには、バックスラッシュをバックスラッシュでエスケープします (\\)。

COPY は、指定した JSON ソースにおいて、正しい形式の有効な JSON オブジェクトまたは JSON 配列を検索します。使用可能な JSON 構造体を見つける前に、または有効な JSON オブジェクトまたは配列の間で COPY が空白以外の文字を検出した場合、COPY は各インスタンスについてエラーを返します。このエラーは、MAXERROR のエラー数としてカウントされます。エラー数が MAXERROR 以上に達すると、COPY は失敗します。

それぞれのエラーについて、Amazon Redshift は STL_LOAD_ERRORS システムテーブルの行を記録します。LINE_NUMBER 列には、エラーの原因となった JSON オブジェクトの最後の行が記録されます。

IGNOREHEADER を指定した場合、COPY は JSON データの指定数の行を無視します。JSON データに含まれる改行文字は、常に IGNOREHEADER の計算にカウントされます。

デフォルトでは、COPY は空の文字列を空のフィールドとしてロードします。EMPTYASNULL を指定した場合、COPY は CHAR および VARCHAR フィールドの空の文字列を NULL としてロードします。INT など、他のデータ型の空の文字列は常に NULL でロードされます。

次のオプションは JSON ではサポートされません。

- CSV
- DELIMITER
- ESCAPE
- FILLRECORD
- FIXEDWIDTH
- IGNOREBLANKLINES
- NULL AS
- READRATIO
- REMOVEQUOTES

詳細については、「JSON 形式からの COPY」を参照してください。JSON データ構造の詳細については、www.json.org を参照してください。

JSONPaths ファイル

JSON 形式または Avro ソースのデータからロードする場合、デフォルトでは COPY はソースデータ内の第 1 レベルのデータ要素をターゲットテーブル内の列にマッピングします。これは、名前と値のペアの各名前またはオブジェクトキーを、ターゲットテーブルの列の名前と一致させることによって行われます。

列名とオブジェクトキーが一致しない場合、またはデータ階層のより深いレベルまでマッピングする場合は、JSONPaths ファイルを使用して明示的に JSON または Avro のデータ要素を列にマッピングできます。JSONPaths ファイルは、ターゲットテーブルまたは列リストで列の順序を一致させることで、JSON データ要素を列にマッピングします。

JSONPaths には、単一の JSON オブジェクト (配列ではない) を格納しなければなりません。JSON オブジェクトは、名前と値のペアです。オブジェクトキー (名前と値のペアの名前) は、"jsonpaths"にする必要があります。名前の値のペアに含まれる値は、JSONPath 式の配列です。各 JSONPath 式は、JSON データ階層内または Avro スキーマの単一の要素を参照します。XPath 式が XML ドキュメント内の要素を参照する方法と似ています。詳細については、「JSONPath 式」を参照してください。

JSONPaths ファイルを使用するには、JSON または AVRO キーワードを COPY コマンドに追加します。次の形式を使用して、JSONPaths ファイルの S3 バケット名とオブジェクトパスを指定します。

```
COPY tablename
FROM 'data_source'
CREDENTIALS 'credentials-args'
FORMAT AS { AVRO | JSON } 's3://jsonpaths_file';
```

`s3://jsonpaths_file` 値は、`'s3://amzn-s3-demo-bucket/jsonpaths.txt'`などの単一のファイルを明示的に参照する Amazon S3 オブジェクトキーである必要があります。キープレフィックスにすることはできません。

場合によっては、Amazon S3 からロードする場合、`jsonpaths_file` で指定されたファイルのプレフィックスは、`copy_from_s3_objectpath` で指定されたデータファイルのパスと同じになります。その場合、COPY は JSONPaths ファイルをデータファイルとして読み込み、エラーを返します。たとえば、データファイルがオブジェクトパス `s3://amzn-s3-demo-bucket/my_data.json` を使用し、JSONPaths ファイルが `s3://amzn-s3-demo-bucket/my_data.jsonpaths` であるとします。この場合、COPY は `my_data.jsonpaths` をデータファイルとしてロードしようとします。

キー名が `"jsonpaths"` 以外の文字列である場合、COPY コマンドはエラーを返しませんが、jsonpaths_file を無視して `'auto'` 引数を代わりに使用します。

以下のいずれかの状況に当てはまる場合、COPY コマンドは失敗します。

- JSON が正しい形式ではない。
- 複数の JSON オブジェクトがある。
- オブジェクトの外に空白以外の文字が存在する。
- 配列エレメントが空の文字列であるか、文字列ではない。

MAXERROR は JSONPaths には適用されません。

ENCRYPTED オプションを指定している場合でも、JSONPaths ファイルの暗号化は行わないでください。

詳細については、「JSON 形式からの COPY」を参照してください。

JSONPath 式

JSONPaths ファイルは JSONPath 式を使用してターゲット列にデータフィールドをマッピングします。各 JSONPath 式は、Amazon Redshift のターゲットテーブル内の 1 列に対応しています。JSONPath 配列要素の順序は、ターゲットテーブル内の列の順序または列リストが使用される場合は列リスト内の列の順序と一致していなければなりません。

例に示すように、フィールド名と値のどちらにも二重引用符が必要です。引用符の文字は、傾きの付いた "高機能な" 引用符ではなくシンプルな引用符 (0x22) にする必要があります。

JSONPath 式によって参照されるオブジェクト要素が JSON データにない場合、COPY は NULL 値をロードしようとします。参照されるオブジェクトが正しい形式ではない場合、COPY はロードエラーを返します。

JSONPath 式により参照される配列要素が JSON データまたは Avro データに見つからない場合、COPY は次のエラー `Invalid JSONPath format: Not an array or index out of range.` で失敗します。ソースデータに存在しない配列要素を JSONPaths からすべて削除し、ソースデータの配列が正しい形式であることを確認してください。

JSONPath 式では、ブラケット表記またはドット表記のいずれかを使用できますが、両方の表記を混ぜて使用することはできません。次の例は、ブラケット表記を使用した JSONPath 式を示しています。

```
{
    "jsonpaths": [
        "$['venuename']",
        "$['venuecity']",
        "$['venuestate']",
        "$['venueseats']"
    ]
}
```

次の例は、ドット表記を使用した JSONPath 式を示しています。

```
{
```

```
    "jsonpaths": [
        "$.venuename",
        "$.venuecity",
        "$.venuestate",
        "$.venueseats"
    ]
}
```

Amazon Redshift COPY 構文のコンテキストでは、JSONPath 式は、JSON または Avro の階層データ構造内の 1 つの名前要素に対する明示的なパスを指定する必要があります。Amazon Redshift では、あいまいなパスや複数の名前要素に解決される可能性がある、ワイルドカード文字やフィルター式などの JSONPath 要素をサポートしていません。

詳細については、「JSON 形式からの COPY」を参照してください。

Avro データに対する JSONPaths の使用

次の例で、複数のレベルがある Avro スキーマを示します。

```
{
    "name": "person",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "guid", "type": "string"},
        {"name": "isActive", "type": "boolean"},
        {"name": "age", "type": "int"},
        {"name": "name", "type": "string"},
        {"name": "address", "type": "string"},
        {"name": "latitude", "type": "double"},
        {"name": "longitude", "type": "double"},
        {
            "name": "tags",
            "type": {
                        "type" : "array",
                        "name" : "inner_tags",
                        "items" : "string"
                    }
        },
        {
            "name": "friends",
            "type": {
                        "type" : "array",
```

```
                          "name" : "inner_friends",
                          "items" : {
                                      "name" : "friends_record",
                                      "type" : "record",
                                      "fields" : [
                                                   {"name" : "id", "type" : "int"},
                                                   {"name" : "name", "type" : "string"}
                                                 ]
                                    }
                      }
        },
        {"name": "randomArrayItem", "type": "string"}
    ]
}
```

次の例で、AvroPath 式を使用して前のスキーマを参照する JSONPaths ファイルを示します。

```
{
    "jsonpaths": [
        "$.id",
        "$.guid",
        "$.address",
        "$.friends[0].id"
    ]
}
```

JSONPaths の例には、以下の要素が含まれています。

jsonpaths

AvroPath 式を含む JSON オブジェクトの名前です。

[…]

かっこ内にパス要素を含む JSON 配列を囲みます。

$

ドル記号は、`"fields"`配列である Avro スキーマのルート要素を表します。

"$.id",

AvroPath 式のターゲットです。このインスタンスでは、ターゲットは `"fields"` 配列の `"id"` という名前の要素です。式はカンマで区切ります。

"$.friends[0].id"

かっこは配列インデックスを示します。JSONPath 式はゼロベースのインデックスを使用します。従って、この式は `"friends"` 配列の第 1 要素を `"id"` と言う名前で参照します。

Avro スキーマの構文では、内部フィールドを使用してレコード構造および配列データ型を定義する必要があります。内部フィールドは AvroPath 式には無視されます。たとえば、フィールド `"friends"` は `"inner_friends"` という名前の配列を定義し、は `"friends_record"` という名前のレコードを定義します。フィールド `"id"` を参照する AvroPath 式は、追加のフィールドを無視してターゲットフィールドを直接参照できます。次の AvroPath 式は、`"friends"`配列内に属する 2 つのフィールドを参照します。

```
"$.friends[0].id"
"$.friends[0].name"
```

列データ形式のパラメータ

標準データ形式に加えて、COPY は、Amazon S3 から COPY の以下の列データ形式をサポートしています。列形式の COPY は、特定の制限でサポートされています。詳細については、「列データ形式の COPY」を参照してください。

ORC

最適化された行列 (ORC) ファイル形式を使用するファイルからデータをロードします。

PARQUET

Parquet ファイル形式を使用するファイルからデータをロードします。

ファイル圧縮パラメータ

次のパラメータを指定して、圧縮されたデータファイルからロードできます。

ファイル圧縮パラメータ

BZIP2

入力ファイルが圧縮された bzip2 形式 (.bz2 ファイル) であることを指定する値です。COPY 操作では、圧縮されたそれぞれのファイルを読み取り、ロード時にデータを解凍します。

GZIP

入力ファイルが圧縮された gzip 形式 (.gz 形式) であることを指定する値です。COPY 操作では、圧縮されたそれぞれのファイルを読み取り、ロード時にデータを解凍します。

LZOP

入力ファイルが圧縮された lzop 形式 (.lzo ファイル) であることを指定する値です。COPY 操作では、圧縮されたそれぞれのファイルを読み取り、ロード時にデータを解凍します。

> **Note**
>
> COPY は、lzop --filter オプションを使用して圧縮されたファイルをサポートしていません。

ZSTD

入力ファイルが圧縮された Zstandard 形式 (.zst ファイル) であることを指定する値です。COPY 操作では、圧縮されたそれぞれのファイルを読み取り、ロード時にデータを解凍します。

> **Note**
>
> ZSTD は、Amazon S3 から COPY を使用する場合のみサポートされます。

データ変換パラメータ

テーブルをロードする際に、COPY は暗黙的にソースデータの文字列をターゲット列のデータ型に変換しようとします。デフォルトの動作とは異なる変換を指定する必要がある場合、またはデフォルトの変換がエラーになった場合、次のパラメータを指定してデータ変換を管理できます。これらのパラメータの構文に関する詳細については、「COPY 構文」を参照してください。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- BLANKSASNULL
- DATEFORMAT
- EMPTYASNULL

- ENCODING
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- IGNOREBLANKLINES
- IGNOREHEADER
- NULL AS
- REMOVEQUOTES
- ROUNDEC
- TIMEFORMAT
- TRIMBLANKS
- TRUNCATECOLUMNS

データ変換パラメータ

ACCEPTANYDATE

`00/00/00 00:00:00` などの無効な形式を含め、任意の日付形式をエラーなしにロードできるようにします。このパラメータは TIMESTAMP 列および DATE 列にのみ適用されます。ACCEPTANYDATE は常に DATEFORMAT パラメータと共に使用します。データの日付形式が DATEFORMAT の仕様と一致しない場合、Amazon Redshift はそのフィールドに NULL 値を挿入します。

ACCEPTINVCHARS [AS] ['replacement_char']

データに無効な UTF-8 文字がある場合でも、VARCHAR 列へのデータのロードを有効にします。ACCEPTINVCHARS を指定した場合、COPY は replacement_char で指定されている文字列から構成される同じ長さの文字列で、無効な各 UTF-8 文字を置き換えます。たとえば、置換文字が '^' である場合、無効な 3 バイト文字は '^^^' で置き換えられます。

置換文字には NULL 以外の任意の ASCII 文字を使用できます。デフォルトは疑問符 (?) です。無効な UTF-8 文字の詳細については、「マルチバイト文字のロードエラー」を参照してください。

COPY は無効な UTF-8 文字を含んだ行の数を返し、対象行ごとに STL_REPLACEMENTS システムテーブルにエントリを追加します (各ノードスライスで最大 100 行まで)。さらに多くの無効な UTF-8 文字も置き換えられますが、それらの置換イベントは記録されません。

ACCEPTINVCHARS を指定しなかった場合、無効な UTF-8 文字があるごとに、COPY はエラーを返します。

ACCEPTINVCHARS は VARCHAR 列に対してのみ有効です。

BLANKSASNULL

NULL など、空白文字のみから構成される空のフィールドをロードします。このオプションは CHAR と VARCHAR の列にのみ適用されます。INT など、他のデータ型の空のフィールドは常に NULL でロードされます。例えば、3 つの連続するスペース文字を含む (それ以外の文字はない) 文字列は NULL としてロードされます。このオプションなしのデフォルト動作では、スペース文字をそのままロードします。

DATEFORMAT [AS] {'dateformat_string' | 'auto' }

DATEFORMAT を指定しない場合、デフォルト形式は `'YYYY-MM-DD'` です。たとえば、有効な代替形式は `'MM-DD-YYYY'` です。

COPY コマンドが日付値または時刻値の形式を認識しない場合、または日付値または時刻値で異なる形式が使用されている場合は、`'auto'`引数を DATEFORMAT または TIMEFORMAT パラメータとともに使用します。`'auto'` 引数は、DATEFORMAT および TIMEFORMAT 文字列を使用する場合にサポートされない形式を認識します。`'auto'` キーワードでは大文字小文字を区別します。詳細については、「DATEFORMAT と TIMEFORMAT で自動認識を使用する」を参照してください。

日付形式には時間情報 (時、分、秒) を含めることができますが、この情報は無視されます。AS キーワードはオプションです。詳細については、「 DATEFORMAT と TIMEFORMAT の文字列」を参照してください。

EMPTYASNULL

Amazon Redshift で CHAR と VARCHAR の空のフィールドを NULL としてロードすることを指定します。INT など、他のデータ型の空のフィールドは常に NULL でロードされます。データに 2 つの区切り記号が連続し、区切り記号の間に文字がない場合、空のフィールドになります。EMPTYASNULL と NULL AS '' (空の文字列) は同じ動作を生成します。

ENCODING [AS] file_encoding

ロードデータのエンコードタイプを指定します。COPY コマンドは、ロード時にデータを指定されたエンコードから UTF-8 に変換します。

file_encoding の有効な値は次のとおりです。

- `UTF8`

- `UTF16`
- `UTF16LE`
- `UTF16BE`
- `ISO88591`

デフォルト: `UTF8`。

ソースファイル名には UTF-8 エンコードを使用する必要があります。

ロードデータに別のエンコードが指定されている場合でも、以下のファイルには UTF-8 エンコードを使用する必要があります。

- マニフェストファイル
- JSONPaths ファイル

次のパラメータを使用して指定される引数文字列には UTF-8 を使用する必要があります。

- FIXEDWIDTH 'fixedwidth_spec'
- ACCEPTINVCHARS 'replacement_char'
- DATEFORMAT 'dateformat_string'
- TIMEFORMAT 'timeformat_string'
- NULL AS 'null_string'

固定幅データファイルは UTF-8 エンコーディングを使用する必要があります。フィールド幅はバイト数ではなく、文字数をベースにしています。

すべてのロードデータには、指定されたエンコードを使用する必要があります。COPY が別のエンコードを検出した場合、ファイルがスキップされてエラーが返されます。

`UTF16` を指定した場合、データにはバイトオーダーマーク (BOM) が必要です。UTF-16 データがリトルエンディアン (LE) であるかビッグエンディアン (BE) であるかがわかっている場合、BOM があるかどうかに関係なく `UTF16LE` または `UTF16BE` を使用できます。

ISO-8859-1 エンコーディングを使用するには、`ISO88591` を指定します。詳細については、Wikipedia の「[ISO/IEC 8859-1](https://en.wikipedia.org/wiki/ISO/IEC_8859-1)」を参照してください。

ESCAPE

このパラメータを指定した場合、入力データのバックスラッシュ文字 (\) はエスケープ文字として扱われます。バックスラッシュ文字の直後に続く文字は、通常は特定の目的に使用される文字

である場合でも、現在の列の値の一部としてテーブルにロードされます。例えば、区切り文字、引用符、埋め込まれた改行文字、またはエスケープ文字自体のいずれかが正当な列の値として含まれている場合、このパラメータを使用してその文字をエスケープできます。

REMOVEQUOTES パラメータと組み合わせて ESCAPE パラメータを使用すると、他の場合には削除される引用符 (' または ") をエスケープし保持することができます。デフォルトの null 文字列 `\N` はそのまま使用できますが、入力データで `\\N` としてエスケープすることもできます。NULL AS パラメータで代替 null 文字列を指定しない限り、`\N`と `\\N` は同じ結果になります。

> **Note**
>
> 制御文字 `0x00` (NUL) はエスケープできません。入力データから削除するか変換してください。この文字はレコードの終わり (EOR) マーカーとして扱われ、レコードの残りの部分は切り捨てられます。

FIXEDWIDTH ロードに対して ESCAPE パラメータを使用することはできません。また、エスケープ文字自体を指定することはできません。エスケープ文字は常にバックスラッシュ文字です。また、入力データの適切な場所にエスケープ文字が含まれていることを確認する必要があります。

次に、ESCAPE パラメータを指定する場合の入力データおよびその結果、ロードされるデータの例を示します。行 4 の結果は、REMOVEQUOTES パラメータも指定されていることを想定しています。入力データはパイプで区切られたフィールド 2 つで構成されます。

```
1|The quick brown fox\[newline]
jumped over the lazy dog.
2| A\\B\\C
3| A \| B \| C
4| 'A Midsummer Night\'s Dream'
```

データは列 2 に次にようにロードされます。

```
The quick brown fox
jumped over the lazy dog.
A\B\C
A|B|C
A Midsummer Night's Dream
```

> **Note**
>
> ロード用の入力データにエスケープ文字を適用する作業はユーザーが担当します。ただし、ESCAPE パラメータを使用して以前アンロードされたデータを再ロードした場合は例外となります。この場合、データにはすでに必要なエスケープ文字が含まれています。

ESCAPE パラメータでは 8 進数、16 進数、Unicode、またはその他のエスケープシーケンス表記を解釈しません。例えば、ソースデータに 8 進数のラインフィード値 (`\012`) があり、ESCAPE パラメータを使用してこのデータをロードしようとすると、Amazon Redshift は値 `012` をテーブルにロードします。この値は、エスケープされているラインフィードとしては解釈されません。

Microsoft Windows プラットフォームからのデータの改行文字をエスケープするには、2 つのエスケープ文字の使用が必要となることがあります。1 つはキャリッジリターン用、もう 1 つはラインフィード用です。または、ファイルのロード前にキャリッジリターンを削除することができます (例えば、dos2unix utility を使用)。

EXPLICIT_IDS

自動生成される値をテーブルのソースデータファイルの明示的値でオーバーライドするには、IDENTITY 列を持つテーブルに EXPLICIT_IDS を使用します。コマンドに列リストが含まれる場合、そのリストにはこのパラメータを使用する IDENTITY 列が含まれている必要があります。EXPLICIT_IDS 値のデータ形式は、CREATE TABLE 定義で指定された IDENTITY 形式に一致する必要があります。

EXPLICIT_IDS オプションを使用してテーブルに対して COPY コマンドを実行する際、Amazon Redshift はテーブル内の IDENTITY 列の一意性をチェックしません。

列がGENERATED BY DEFAULT AS IDENTITYで定義されている場合は、コピーできます。値は、指定した値で生成または更新されます。EXPLICIT_IDS オプションは必須ではありません。COPY は IDENTITY のハイウォーターマークを更新しません。

EXPLICIT_IDS を使用する COPY コマンドの例については、「IDENTITY 列の明示的値を使用して VENUE をロードする」を参照してください。

FILLRECORD

一部のレコードの最後で連続する列が欠落している場合に、データをロードできるようにします。欠落している列は NULL としてロードされます。テキスト形式と CSV 形式では、VARCHAR 列が欠落している場合、NULL の代わりに長さ 0 の文字列がロードされます。テ

キストおよび CSV から VARCHAR 列に NULL をロードするには、EMPTYASNULL キーワードを指定します。NULL の置き換えは、列定義で NULL が使用できる場合にのみ可能です。

たとえば、テーブル定義に 4 つの null が許容された CHAR 列があり、レコードに値 `apple, orange, banana, mango` がある場合、COPY コマンドは値 `apple, orange` のみを含むレコードをロードし、記入できます。欠落している CHAR 値は NULL 値としてロードされます。

IGNOREBLANKLINES

データファイルでラインフィードのみ含む空白行を無視し、ロードしません。

IGNOREHEADER [AS] number_rows

指定された number_rows をファイルヘッダーとして扱い、ロードしません。並列ロードですべてのファイルのファイルヘッダーをスキップするには、IGNOREHEADER を使用します。

NULL AS 'null_string'

null_string に一致するフィールドを NULL としてロードします。ここで null_string は任意の文字列です。データに null ターミネータ (NUL (UTF-8 0000) またはバイナリゼロ (0x000) と呼ばれることもあります) が含まれる場合、COPY はそれを他の文字として扱います。例えば、'1' || NUL || '2' を含むレコードは長さ 3 バイトの文字列としてコピーされます。フィールドに NUL のみが含まれている場合は、NULL AS を使用して、`'\0'` または `'\000'` (例えば、`NULL AS '\0'` または `NULL AS '\000'`) を指定することにより、null ターミネータを NULL に置き換えることができます。フィールドに NUL で終わる文字列が含まれており、NULL AS を指定した場合、文字列の最後に NUL が挿入されます。null_string 値に '\ n' (改行) を使用しないでください。Amazon Redshift は、行区切り文字として使用するために '\n' を予約します。デフォルトの null_string は `'\N`' です。

> **Note**
>
> NOT NULL として定義された列に Null のロードを試みた場合、COPY コマンドは失敗します。

REMOVEQUOTES

入力データの文字列を囲む引用符を削除します。区切り記号を含む引用符内のすべての文字は保持されます。文字列に開始の一重または二重引用符があるが、対応する終了引用符がない場合、COPY コマンドはその行をロードできず、エラーを返します。次の表は、引用符を含む文字列とその結果、ロードされる値の簡単な例を示しています。

入力文字列	REMOVEQUOTES オプションでロードされる値
"区切り記号はパイプ (|) 文字です"	区切り記号はパイプ (|) 文字です
'黒'	黒
"白"	白
青'	青'
青'	値はロードされません: エラー状態
"青	値はロードされません: エラー状態
'''黒'''	''黒''
''	<空白>

ROUNDEC

入力値の小数点以下の桁数が列の小数点以下の桁数よりも多い場合に数値を四捨五入します。COPY のデフォルト動作では、列の小数点以下の桁数に合わせて必要に応じて値が切り捨てられます。たとえば、`20.259`という値が DECIMAL(8,2) 列にロードされた場合、COPY はデフォルトでこの値を `20.25` に切り捨てます。ROUNDEC が指定されている場合、COPY はこの値を `20.26` に四捨五入します。INSERT コマンドでは、列の小数点以下の桁数に合わせて必要に応じて値が四捨五入されます。そのため、COPY コマンドを ROUNDEC パラメータとともに使用した場合の動作は、INSERT コマンドを実行した場合の動作と同じになります。

TIMEFORMAT [AS] {'timeformat_string' | 'auto' | 'epochsecs' | 'epochmillisecs' }

時間形式を指定します。TIMEFORMAT が指定されていない場合、デフォルトの形式は、TIMESTAMP 列では `YYYY-MM-DD HH:MI:SS`、TIMESTAMPTZ 列では `YYYY-MM-DD HH:MI:SSOF` です。`OF` は協定世界時 (UTC) からのオフセットです。timeformat_string には、タイムゾーン指定子を含めることができません。デフォルトの形式と異なる形式の TIMESTAMPTZ データをロードするには、「auto」を指定します。詳細については、「DATEFORMAT と TIMEFORMAT で自動認識を使用する」を参照してください。timeformat_string の詳細については、「 DATEFORMAT と TIMEFORMAT の文字列」を参照してください。

'auto' 引数は、DATEFORMAT および TIMEFORMAT 文字列を使用する場合にサポートされない形式を認識します。COPY コマンドが日付値または時刻値の形式を認識しない場合、または日付値および時刻値でそれぞれ異なる形式が使用されている場合は、'auto'引数を DATEFORMAT または TIMEFORMAT パラメータとともに使用します。詳細については、「DATEFORMAT と TIMEFORMAT で自動認識を使用する」を参照してください。

ソースデータがエポック時間 (1970 年 1 月 1 日 00:00:00 UTC からの秒数またはミリ秒数) で表されている場合、'epochsecs'または 'epochmillisecs' を指定します。

'auto'、'epochsecs'、'epochmillisecs'の各キーワードでは大文字小文字を区別します。

AS キーワードはオプションです。

TRIMBLANKS

VARCHAR 文字列から末尾の空白文字を削除します。このパラメータは VARCHAR データ型の列にのみ適用されます。

TRUNCATECOLUMNS

列の仕様に合うよう、該当する文字数で列のデータを切り捨てます。データ型が VARCHAR または CHAR の列、およびサイズが 4 MB 以下の行にのみ適用されます。

データのロード操作

次のパラメータを指定して、トラブルシューティングの際のロード操作のデフォルトの動作を管理したり、ロード時間を短縮します。

- COMPROWS
- COMPUPDATE
- IGNOREALLERRORS
- MAXERROR
- NOLOAD
- STATUPDATE

パラメータ

COMPROWS numrows

圧縮分析のサンプルサイズとして使用される行数を指定します。分析は各データスライスの行に対して実行されます。たとえば、`COMPROWS 1000000`(1,000,000) を指定し、システムに合計 4 つのスライスが含まれている場合、スライスごとに 250,000 行のみが読み取られ、分析されます。

COMPROWS を指定しない場合、サンプルサイズはデフォルトでスライスごとに 100,000 になります。COMPROWS の値がスライスごとに 100,000 行のデフォルト値より小さい場合、自動的にデフォルト値にアップグレードされます。ただし、ロードされるデータの量が有意のサンプルとしては不十分な場合、自動圧縮は実行されません。

COMPROWS 数が入力ファイルの行数より大きい場合でも、COPY コマンドは続行し、利用可能なすべての行で圧縮分析を実行します。この引数の許容範囲は 1000～2147483647 (2,147,483,647) の数値です。

COMPUPDATE [PRESET | { ON | TRUE } | { OFF | FALSE }],

COPY 実行中に圧縮エンコードを自動的に適用するかどうかを制御します。

COMPUPDATE が PRESET の場合、COPY コマンドを実行すると、ターゲットテーブルが空の場合、列に RAW 以外のエンコードが既に指定されていても、各列に圧縮エンコードが選択されます。現在指定されている列のエンコードは置き換えることができます。各列のエンコードは、列のデータタイプに基づきます。サンプリングされているデータはありません。Amazon Redshift では、次のように圧縮エンコードが自動的に割り当てられます。

- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、または DOUBLE PRECISION データ型として定義されている列には、RAW 圧縮が割り当てられます。
- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIMESTAMP、または TIMESTAMPTZ として定義された列には AZ64 圧縮が割り当てられます。
- CHAR または VARCHAR として定義された列には、LZO 圧縮が割り当てられます。

COMPUPDATE を削除して COPY コマンドを実行すると、ターゲットテーブルが空で、どの列にもエンコード (RAW は除く) を指定していない場合にのみ、各列に圧縮エンコードが選択されます。各列のエンコードは、Amazon Redshift によって決定されます。サンプリングされているデータはありません。

COMPUPDATE が ON (または TRUE) の場合、または COMPUPDATE がオプションなしで指定されている場合は、テーブルの列に RAW 以外のエンコードがすでに指定されていても、テーブルが空であれば COPY によって自動圧縮が適用されます。現在指定されている列のエンコードは置き換えることができます。列ごとのエンコードは、サンプルデータの分析によって異なります。詳細については、「自動圧縮ありでテーブルをロードする」を参照してください。

COMPUPDATE OFF (または FALSE) の場合、自動圧縮は無効になります。列のエンコードを変更することはできません。

圧縮を分析するシステムテーブルの詳細については、「STL_ANALYZE_COMPRESSION」を参照してください。

IGNOREALLERRORS

このオプションを指定すると、ロードオペレーション中に発生したすべてのエラーを無視します。

MAXERROR オプションを指定している場合は、IGNOREALLERRORS オプションを指定することはできません。ORC や Parquet の列形式に対しては、IGNOREALLERRORS オプションを指定できません。

MAXERROR [AS] error_count

ロードのエラー数が error_count 以上である場合、ロードは失敗します。ロードのエラーがそれより少ない場合、処理は続行され、ロードできなかった行数を示す INFO メッセージが返されます。データの形式エラーやその他の不整合のために一部の行をテーブルにロードできないときにロードを継続するには、このパラメータを使用します。

最初のエラーが発生したときにロードを失敗させる場合、この値を `0` または `1` に設定します。AS キーワードはオプションです。MAXERROR のデフォルト値は `0`、そしてその限度は `100000` です。

Amazon Redshift の並列処理のため、報告される実際のエラー数が指定された MAXERROR より大きくなることがあります。Amazon Redshift クラスターのノードで MAXERROR を超えたことが検出された場合、各ノードは発生したすべてのエラーを報告します。

NOLOAD

データを実際にロードせずにデータファイルの有効性をチェックします。実際にデータロードを実行せずに、エラーなしでデータファイルがロードされることを確認するには、NOLOAD パラメータを使用します。NOLOAD パラメータと共に COPY を実行すると、ファイルを解析するだけであるため、データのロードよりはるかに高速になります。

STATUPDATE [{ ON | TRUE } | { OFF | FALSE }]

COPY コマンドが成功したとき最後に行う自動計算とオプティマイザ統計の更新を制御します。デフォルトでは、STATUPDATE パラメータを使用しない場合、テーブルが最初は空ならば、統計は自動的に更新されます。

データを空ではないテーブルに入れるとテーブルのサイズが大きく変化する場合は、常に ANALYZE コマンドを実行するか STATUPDATE ON 引数を使用して統計を更新することをお勧めします。

STATUPDATE ON (または TRUE) の場合、テーブルが最初に空であるかどうかに関係なく、統計は自動的に更新されます。STATUPDATE を使用する場合、現在のユーザーはテーブル所有者またはスーパーユーザーであることが必要です。STATUPDATE を指定しない場合、INSERT 権限のみ必要です。

STATUPDATE OFF (または FALSE) を使用すると、統計は更新されません。

詳細については、「テーブルを分析する」を参照してください。

パラメータリスト (アルファベット順)

次の一覧は、アルファベット順にソートされた COPY コマンドパラメータのそれぞれの説明へのリンクです。

- ACCEPTANYDATE
- ACCEPTINVCHARS
- ACCESS_KEY_ID and SECRET_ACCESS_KEY
- AVRO
- BLANKSASNULL
- BZIP2
- COMPROWS
- COMPUPDATE
- CREDENTIALS
- CSV
- DATEFORMAT
- DELIMITER

- EMPTYASNULL
- ENCODING
- ENCRYPTED
- ESCAPE
- EXPLICIT_IDS
- FILLRECORD
- FIXEDWIDTH
- FORMAT
- FROM
- GZIP
- IAM_ROLE
- IGNOREALLERRORS
- IGNOREBLANKLINES
- IGNOREHEADER
- JSON
- LZOP
- MANIFEST
- MASTER_SYMMETRIC_KEY
- MAXERROR
- NOLOAD
- NULL AS
- READRATIO
- REGION
- REMOVEQUOTES
- ROUNDEC
- SESSION_TOKEN
- SHAPEFILE
- SSH
- STATUPDATE
- TIMEFORMAT

- SESSION_TOKEN
- TRIMBLANKS
- TRUNCATECOLUMNS
- ZSTD

使用に関する注意事項

トピック

他の AWS リソースにアクセスするアクセス許可

クラスターと他の AWS リソース (Amazon S3、Amazon DynamoDB、Amazon EMR、または Amazon EC2 など) との間でデータを移動する場合、クラスターには、リソースにアクセスして必要なアクションを実行するための許可が必要です。例えば、Amazon S3 からデータをロードする場合、COPY はバケットへの LIST アクセスとバケットオブジェクトへの GET アクセスが必要です。最小限のアクセス権限については、「COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可」を参照してください。

リソースにアクセスする認可を取得するには、クラスターが認証される必要があります。次の認証方法のいずれかを選択できます。

- ロールベースアクセスコントロール– ロールベースのアクセスコントロールの場合、クラスターが認証と認可に使用する AWS Identity and Access Management (IAM) ロールを指定します。AWS

認証情報および機密データを保護するには、ロールベースの認証を使用することを強くお勧めします。

- キーベースのアクセスコントロール – キーベースのアクセスコントロールの場合は、ユーザーの AWS アクセス認証情報 (アクセスキー ID とシークレットアクセスキー) をプレーンテキストとして指定します。

ロールベースアクセスコントロール

ロールベースのアクセスコントロールを使用して、クラスターはユーザーに代わって一時的に、IAM ロールを引き受けます。その後、そのロールに付与された承認内容に基づいて、クラスターは必要な AWS のリソースにアクセスできるようになります。

IAM ロールの作成は、ロールが AWS アイデンティティであり、AWS で何を実行でき、何を実行できないかを決定するアクセス許可ポリシーを持つという点で、ユーザーへのアクセス許可の付与と似ています。ただし、1 人のユーザーに一意に関連付けられるのではなく、ロールは必要に応じてすべてのエンティティが引き受けることができます。また、ロールにはいずれの認証情報 (パスワードやアクセスキー) も関連付けられません。代わりに、ロールがクラスターに関連付けられた場合は、アクセスキーが動的に作成され、クラスターに提供されます。

AWS 認証情報を保護することに加えて、AWSのリソースおよび機密ユーザーデータへのアクセスに対して、より安全で、きめの細かいコントロールを提供するロールベースのアクセスコントロールの使用をお勧めします。

ロールベースの認証には次の利点があります。

- AWS 標準の IAM ツールを使用して、IAM ロールを定義し、そのロールを複数のクラスターと関連付けられます。ロールのためにアクセスポリシーを変更すると、変更はロールを使用するすべてのクラスターに自動的に適用されます。
- 特定の AWS リソースおよびアクションへのアクセス許可を、特定のクラスターとデータベースユーザーに付与するための、きめ細かな IAM ポリシーを定義できます。
- クラスターは、実行時に一時的なセッション認証情報を取得し、必要に応じて操作が完了するまで認証情報を更新します。キーに基づく一時的認証情報を使用する場合、完了する前に一時的認証情報の期限が切れると、操作は失敗します。
- アクセスキー ID とシークレットアクセスキー ID は SQL コードには格納、送信されません。

ロールベースのアクセスコントロールを使用するには、Amazon Redshift サービスロールタイプを使用して IAM ロールを作成してから、クラスターにロールをアタッチする必要があります。ロール

は、少なくとも COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可 に示されたアクセス権限が必要です。IAM ロールを作成してクラスターにアタッチするステップについては、「Amazon Redshift 管理ガイド」の「ユーザーに代わって Amazon Redshift が他の AWS サービスにアクセスすることを認可する」を参照してください。

クラスターにロールを追加するか、Amazon Redshift マネジメントコンソール、CLI、または API を使用してクラスターに関連付けられるロールを表示できます。詳細については、「Amazon Redshift 管理ガイド」の「IAM ロールとクラスターの関連付け」を参照してください。

IAM ロールを作成する場合、IAM はロールの Amazon リソースネーム (ARN) を返します。IAM ロールを指定するには、IAM_ROLEパラメータまたは CREDENTIALS パラメータでロールの ARN を指定します。

例えば、以下のロールがクラスターにアタッチされるとします。

```
"IamRoleArn": "arn:aws:iam::0123456789012:role/MyRedshiftRole"
```

次の COPY コマンドの例では、Amazon S3 への認証とアクセスのために前の例の IAM_ROLE パラメータと ARN を使用します。

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

次の COPY コマンドの例は、CREDENTIALS パラメータを使用して IAM ロールを指定しています。

```
copy customer from 's3://amzn-s3-demo-bucket/mydata'
credentials
'aws_iam_role=arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

さらに、スーパーユーザーは、データベースユーザーおよびグループに ASSUMEROLE 権限を付与して、COPY オペレーションのロールへのアクセスを提供できます。詳細については、GRANTを参照してください。

キーベースのアクセスコントロール

キーベースのアクセスコントロールを使用して、データが含まれている AWS リソースへのアクセスを許可された、IAM ユーザーのアクセスキー ID とシークレットアクセスキーを提供します。ACCESS_KEY_ID and SECRET_ACCESS_KEY パラメータを一緒に使用するか、CREDENTIALSパラメータを使用できます。

> **Note**
>
> プレーンテキストのアクセスキー ID とシークレットアクセスキーを指定する代わりに、認証のために IAM ロールを使用することを強くお勧めします。キーベースのアクセスコントロールを選択する場合は、AWSアカウント (ルート) 認証情報を使用しないでください。常に IAM ユーザーを作成し、そのユーザーのアクセスキー ID とシークレットアクセスキーを指定します。IAM ユーザーを作成する手順については、「AWS アカウントでの IAM ユーザーの作成」を参照してください。

ACCESS_KEY_ID と SECRET_ACCESS_KEY を使用して認証するには、次に示すように、承認されたユーザーのアクセスキー ID および完全なシークレットアクセスキーで、*<access-key-id>* と *<secret-access-key>* を置き換えます。

```
ACCESS_KEY_ID '<access-key-id>'
SECRET_ACCESS_KEY '<secret-access-key>';
```

CREDENTIALS パラメータを使用して認証するには、次に示すように、承認されたユーザーのアクセスキー ID および完全なシークレットアクセスキーで、*<access-key-id>* と *<secret-access-key>* を置き換えます。

```
CREDENTIALS
'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>';
```

IAM ユーザーは、少なくとも COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可に示された許可が必要です。

一時的な認証情報

キーに基づくアクセスコントロールを使用する場合、一時的なセキュリティ認証情報を使用して、データへのユーザーのアクセスを制限できます。ロールベースの認証は、自動的に一時的な認証情報を使用します。

> **Note**
>
> 一時的な認証情報を作成してアクセスキー ID とシークレットアクセスキーをプレーンテキストで提供するのではなく、「role-based access control」を使用することを強くお勧めします。ロールベースのアクセスコントロールは、自動的に一時的な認証情報を使用します。

一時的セキュリティ認証情報はセキュリティを強化します。使用期限が短く、期限が切れた後は再利用できないためです。トークンを使用して生成されるアクセスキー ID とシークレットアクセスキーはトークンなしに使用できません。これらの一時的セキュリティ認証情報を持つユーザーは認証情報の有効期限内のみリソースにアクセスできます。

ユーザーにリソースへの一時的アクセスを許可するには AWS Security Token Service (AWS STS) API 操作を呼び出します。AWS STS API 操作は、セキュリティトークン、アクセスキー ID、およびシークレットアクセスキーから構成される一時的セキュリティ認証情報を返します。一時的セキュリティ認証情報は、リソースへの一時的アクセスを必要とするユーザーに発行します。これらのユーザーは既存の IAM ユーザーであるか、非 AWS ユーザーです。一時的なセキュリティ認証情報の作成に関する詳細については、IAM ユーザーガイドの[一時的なセキュリティ認証情報の使用]を参照してください。

[ACCESS_KEY_ID and SECRET_ACCESS_KEY] パラメータとともに [SESSION_TOKEN] パラメータを使用するか、[CREDENTIALS]パラメータを使用できます。また、トークンと共に提供されているアクセスキー ID とシークレットアクセスキーを指定する必要があります。

ACCESS_KEY_ID、SECRET_ACCESS_KEY、および SESSION_TOKEN を使用して認証するには、次に示すように、*<temporary-access-key-id>*、*<temporary-secret-access-key>*、および *<temporary-token>* を置き換えます。

```
ACCESS_KEY_ID '<temporary-access-key-id>'
SECRET_ACCESS_KEY '<temporary-secret-access-key>'
SESSION_TOKEN '<temporary-token>';
```

CREDENTIALS を使用して認証するには、次のように `session_token=<temporary-token>` を認証情報文字列に含めます。

```
CREDENTIALS
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>';
```

次の例は、一時的セキュリティ認証情報を使用する COPY コマンドを示しています。

```
copy table-name
from 's3://objectpath'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>';
```

次の例では、一時的認証情報とファイル暗号化を使用して LISTING テーブルをロードします。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
access_key_id '<temporary-access-key-id>'
secret_access_key '<temporary-secret-access-key>'
session_token '<temporary-token>'
master_symmetric_key '<root-key>'
encrypted;
```

次の例では、CREDENTIALS パラメータを一時的認証情報およびファイル暗号化とともに使用して LISTING テーブルをロードします。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
credentials
'aws_access_key_id=<temporary-access-key-id>;aws_secret_access_key=<temporary-secret-access-key>;session_token=<temporary-token>;master_symmetric_key=<root-key>'
encrypted;
```

> **Important**
>
> 一時的セキュリティ認証情報は、COPY および UNLOAD 操作の期間全体で有効にする必要があります。一時的セキュリティ認証情報の期限が操作中に切れた場合、コマンドは失敗し、処理はロールバックされます。例えば、一時的セキュリティ認証情報の期限が 15 分後に切れるときに COPY 操作に 1 時間かかる場合、COPY 操作は完了前に失敗します。ロールベースのアクセスを使用する場合、一時的なセキュリティ認証情報は操作が完了するまで自動的に更新されます。

COPY、UNLOAD、CREATE LIBRARY のための IAM のアクセス許可

CREDENTIALS パラメータによって参照される IAM ロールまたはユーザーには、少なくとも、次のアクセス許可が必要です。

- Amazon S3 から COPY の場合、Amazon S3 バケットを LIST するアクセス許可であり、ロードされている Amazon S3 オブジェクトと、マニフェストファイル (使用する場合) を GET するアクセス許可。

- Amazon S3、Amazon EMR、および JSON 形式のデータのリモートホスト (SSH) から COPY を実行する場合は、Amazon S3 の JSONPaths ファイル (使用する場合) に対して LIST および GET を実行するアクセス許可。
- DynamoDB から COPY を実行する場合は、ロードされた DynamoDB テーブルに対して SCAN および DESCRIBE を実行するアクセス許可。
- Amazon EMR クラスターから COPY を実行する場合、`ListInstances` アクションを Amazon EMR クラスターで実行するための許可。
- Amazon S3 への UNLOAD の場合、データファイルのアンロード先 Amazon S3 バケットに対する GET、LIST、および PUT 許可。
- Amazon S3 からの CREATE LIBRARY の場合、Amazon S3 バケットを一覧表示し、インポートされる Amazon S3 オブジェクトを取得する許可。

> **Note**
>
> COPY、UNLOAD、CREATE LIBRARY コマンドの実行時に、`S3ServiceException: Access Denied`というエラーメッセージが返される場合、クラスターには Amazon S3 への適切なアクセス許可がありません。

IAM ポリシーを管理するには、クラスター、ユーザー、またはユーザーが属するグループにアタッチされている IAM ロールに IAM ポリシーをアタッチします。例えば、`AmazonS3ReadOnlyAccess` マネージドポリシーは、Amazon S3 リソースへの LIST および GET 許可を付与します。IAM ポリシーの詳細については、IAM ユーザーガイドの IAM ポリシーの管理を参照してください。

Amazon S3 アクセスポイントのエイリアスで COPY を使用する

COPY は、Amazon S3 アクセスポイントのエイリアスをサポートしています。詳細については、Amazon Simple Storage Service ユーザーガイドのアクセスポイントにバケットスタイルのエイリアスを使用するを参照してください。

Amazon S3 からマルチバイトのデータをロードする

データに ASCII 以外のマルチバイト文字 (漢字やキリル文字) が含まれる場合、データを VARCHAR 列にロードする必要があります。VARCHAR データ型は 4 バイトの UTF-8 文字をサポートしますが、CHAR データ型はシングルバイトの ASCII 文字のみを受け取ります。5 バイト以上の文字を Amazon Redshift テーブルにロードすることはできません。詳細については、「マルチバイト文字」を参照してください。

GEOMETRY もしくは GEOGRAPHY データ型の列のロード

CSV ファイルなどの文字区切りのテキストファイルに含まれるデータから、GEOMETRY あるいは GEOGRAPHY 列に対して COPY が可能です。データは、Well-known Binary (WKB もしくは EWKB のいずれか) 形式、または Well-known Text (WKT もしくは EWKT) 形式の 16 進表記であり、COPY コマンドへの単一の入力行の最大サイズ内に収まる必要があります。詳細については、「COPY」を参照してください。

シェープファイルからロードする方法については、「シェープファイルを Amazon Redshift にロードする」を参照してください。

GEOMETRY および GEOGRAPHY データ型の詳細については、「Amazon Redshift での空間データのクエリ」を参照してください。

HLLSKETCH データ型のロード

HLL スケッチは、Amazon Redshift でサポートされているスパース形式またはデンス形式でのみコピーできます。HyperLogLog スケッチで COPY コマンドを使用するには、デンスの HyperLogLog スケッチには Base64 形式を使用し、スパースの HyperLogLog スケッチには JSON 形式を使用します。詳細については、「HyperLogLog 関数」を参照してください。

次の例では、CREATE TABLE および COPY を使用して、CSV ファイルからテーブルにデータをインポートします。まず、この例では、CREATE TABLE を使用してテーブル t1 を作成します。

```
CREATE TABLE t1 (sketch hllsketch, a bigint);
```

次に、COPY を使用して CSV ファイルからテーブル t1 にデータをインポートします。

```
COPY t1 FROM s3://amzn-s3-demo-bucket/unload/' IAM_ROLE
 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' CSV;
```

VARBYTE データ型の列のロード

CSV、Parquet、ORC 形式のファイルからデータをロードできます。CSV の場合、データは、VARBYTE データの 16 進数表現によりファイルからロードされます。FIXEDWIDTH オプションを指定して VARBYTE データをロードすることはできません。COPY の ADDQUOTES または REMOVEQUOTES オプションは、サポートされていません。VARBYTE 列をパーティション列として使用することはできません。

複数ファイル読み取り時のエラー

COPY コマンドはアトミックでトランザクショナルです。つまり、COPY コマンドが複数のファイルからデータを読み取る場合でも、プロセス全体は 1 つのトランザクションとして扱われます。COPY でファイル読み込みにエラーが発生した場合、プロセスがタイムアウトになるまで (statement_timeout を参照)、または長時間 (15～30 分間) Amazon S3 からデータをダウンロードできないときは各ファイルを 1 回のみロードするようにして、自動的にファイル読み込みを再試行します。COPY コマンドが失敗した場合、トランザクション全体がキャンセルされ、変更はすべてロールバックされます。ロードエラー処理の詳細については、「データロードのトラブルシューティング」を参照してください。

COPY コマンドが正常に開始されると、クライアントによる切断などでセッションが終了しても、アプリケーションは停止しません。ただし、COPY コマンドが BEGIN … END セッションブロック内にあり、セッションが終了したためにこのブロックが完了していない場合、COPY を含むすべてのトランザクションがロールバックされます。トランザクションの詳細については、「BEGIN」を参照してください。

JSON 形式からの COPY

JSON のデータ構造は、一連のオブジェクトまたは配列により構成されています。JSON オブジェクトの先頭と末尾には中括弧が付き、順序が設定されていない一連の名前と値のペアが含まれます。各名前と値はコロンで区切られ、ペアはカンマで区切られます。名前は二重引用符で囲まれた文字列です。引用符は、傾きの付いた「高機能な」引用符ではなくシンプルな引用符 (0x22) にする必要があります。

JSON 配列の先頭と末尾には角括弧が付き、順序が設定された一連のカンマ区切りの値が含まれます。値には、二重引用符で囲まれた文字列、数値、ブール値の true または false、Null、JSON オブジェクト、配列を指定できます。

JSON のオブジェクトと配列を入れ子にして、階層データ構造を実現できます。次の例は、2 つの有効なオブジェクトを持つ JSON データ構造を示しています。

```
{
    "id": 1006410,
    "title": "Amazon Redshift Database Developer Guide"
}
{
    "id": 100540,
    "name": "Amazon Simple Storage Service User Guide"
}
```

2 つの JSON 配列と同じデータを次に示します。

```
[
    1006410,
    "Amazon Redshift Database Developer Guide"
]
[
    100540,
    "Amazon Simple Storage Service User Guide"
]
```

JSON の COPY オプション

JSON 形式のデータで COPY を使用する場合は、次のオプションを指定できます。

- `'auto'` – COPY は JSON ファイルからフィールドを自動的にロードします。
- `'auto ignorecase'` – COPY は、フィールド名の大文字と小文字を区別せずに、JSON ファイルからフィールドを自動的にロードします。
- `s3://jsonpaths_file` – COPY は JSONPaths ファイルを使用して JSON ソースデータを解析します。JSONPaths ファイルは、JSONPath 式の配列とペアになった `"jsonpaths"` という名前の単一の JSON オブジェクトを格納するテキストファイルです。名前が `"jsonpaths"` 以外の文字列である場合、COPY は JSONPaths ファイルの代わりに `'auto'` 引数を使用します。

`'auto'`、`'auto ignorecase'`、または JSONPaths ファイルを使用し、JSON オブジェクトまたは配列のいずれかを使用してデータをロードする方法を示す例については、JSON からのコピーの例を参照してください。

JSONPath オプション

Amazon Redshift COPY 構文では、JSONPath 式は、角括弧表記またはドット表記のいずれかを使用して、JSON の階層データ構造内の 1 つの名前要素に対する明示的なパスを指定します。Amazon Redshift では、あいまいなパスや複数の名前要素に解決される可能性がある、ワイルドカード文字やフィルター式などの JSONPath 要素をサポートしていません。その結果、Amazon Redshift は複雑な複数レベルのデータ構造を解析することはできません。

次は、ブラケット表記を使用した JSONPath 式を含む JSONPaths ファイルの例です。ドル記号 ($) はルートレベル構造を現します。

```
{
    "jsonpaths": [
```

```
        "$['id']",
        "$['store']['book']['title']",
  "$['location'][0]"
     ]
}
```

前の例で、$['location'][0]は配列内の最初の要素を参照します。JSON はゼロベースの配列インデックス付けを使用します。配列インデックスは正の整数 (0 以上) である必要があります。

次の例は、前出の JSONPaths ファイルをドット表記で表したものです。

```
{
    "jsonpaths": [
        "$.id",
        "$.store.book.title",
  "$.location[0]"
     ]
}
```

jsonpaths 配列でブラケット表記とドット表記を混在させることはできません。ブラケットは、配列要素を参照するためにブラケット表記とドット表記の両方で使用できます。

ドット表記を使用する場合、JSONPath の式は以下の文字を含む必要があります。

- 1 つの一重引用符 (')
- ピリオドまたはドット (.)
- 配列要素を参照するために使用されていない場合はブラケット ([ ])

JSONPath 式によって参照される名前と値のペアの値がオブジェクトまたは配列の場合は、中括弧または角括弧を含むオブジェクトまたは配列全体が文字列としてロードされます。例えば、JSON データに次のオブジェクトが含まれているとします。

```
{
    "id": 0,
    "guid": "84512477-fa49-456b-b407-581d0d851c3c",
    "isActive": true,
    "tags": [
        "nisi",
        "culpa",
        "ad",
        "amet",
```

```
            "voluptate",
            "reprehenderit",
            "veniam"
        ],
        "friends": [
            {
                "id": 0,
                "name": "Martha Rivera"
            },
            {
                "id": 1,
                "name": "Renaldo"
            }
        ]
}
```

この場合、JSONPath 式 `$['tags']` は次の値を返します。

```
"["nisi","culpa","ad","amet","voluptate","reprehenderit","veniam"]"
```

この場合、JSONPath 式 `$['friends'][1]` は次の値を返します。

```
"{"id": 1,"name": "Renaldo"}"
```

`jsonpaths` 配列の各 JSONPath 式は、Amazon Redshift のターゲットテーブル内の 1 列に対応しています。`jsonpaths` 配列要素の順序は、ターゲットテーブル内の列の順序または列リストが使用される場合は列リスト内の列の順序と一致していなければなりません。

`'auto'` 引数または JSONPaths ファイルを使用し、JSON オブジェクトまたは配列のいずれかを使用してデータをロードする方法を示す例については、「JSON からのコピーの例」を参照してください。

複数の JSON ファイルをコピーする方法については、マニフェストを使用し、データファイルを指定する を参照してください。

JSON のエスケープ文字

COPY は改行文字として \n を、タブ文字として \t をロードします。バックスラッシュをロードするには、バックスラッシュをバックスラッシュでエスケープします (\\)。

たとえば、バケット `escape.json` 内の `s3://amzn-s3-demo-bucket/json/` という名前のファイルに、次の JSON があるとします。

```
{
  "backslash": "This is a backslash: \\",
  "newline": "This sentence\n is on two lines.",
  "tab": "This sentence \t contains a tab."
}
```

ESCAPES テーブルを作成し JSON をロードするには、次のコマンドを実行します。

```
create table escapes (backslash varchar(25), newline varchar(35), tab varchar(35));

copy escapes from 's3://amzn-s3-demo-bucket/json/escape.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as json 'auto';
```

ESCAPES テーブルにクエリを実行し、結果を表示します。

```
select * from escapes;

       backslash        |      newline       |               tab
------------------------+-------------------+----------------------------------
 This is a backslash: \ | This sentence      | This sentence    contains a tab.
                        :  is on two lines.
(1 row)
```

数値の精度の喪失

JSON 形式のデータファイルから数値データ型として定義された列に数値をロードするときに、精度が失われる可能性があります。一部の浮動小数点値は、コンピュータシステムで正確に表されません。そのため、JSON ファイルからコピーするデータは、想定したとおりに丸められない可能性があります。精度の喪失を避けるため、次のいずれかの代替策を使用することをお勧めします。

- 二重引用文字で値を囲んで、数値を文字列として表します。
- ROUNDEC を使用して、数値を切り捨てるのではなく丸めます。
- JSON または Avro ファイルを使用する代わりに、CSV、文字区切り形式、または固定幅形式のテキストファイルを使用します。

列データ形式の COPY

COPY では、次の列形式で Amazon S3 からデータをロードできます。

- ORC
- Parquet

列データ形式からの COPY の使用例については、「COPY の例」を参照してください。

COPY では、列形式のデータがサポートされますが、以下の考慮事項があります。

- Amazon S3 バケットは、Amazon Redshift データベースと同じ AWS リージョンに存在する必要があります。
- VPC エンドポイントを介して Amazon S3 データにアクセスするには、「Amazon Redshift 管理ガイド」の「拡張 VPC のルーティングで Amazon Redshift Spectrum を使用する」の説明に沿って、IAM ポリシーと IAM ロールを使用してアクセスを設定します。
- COPY では、圧縮エンコードは自動的に適用されません。
- 以下の COPY パラメータのみサポートされています。
  - ORC または Parquet ファイルからコピーする場合は ACCEPTINVCHARS。
  - FILLRECORD
  - FROM
  - IAM_ROLE
  - CREDENTIALS
  - STATUPDATE
  - MANIFEST
  - EXPLICIT_IDS
- ロード中に COPY でエラーが発生すると、コマンドは失敗します。ACCEPTANYDATE および MAXERROR は、列データ型ではサポートされていません。
- エラーメッセージは、SQL クライアントに送信されます。一部のエラーは、STL_LOAD_ERRORS と STL_ERROR に記録されます。
- COPY は列データファイルで発生した列と同じ順序でターゲットテーブルの列に値を挿入します。ターゲットテーブルの列数とデータファイルの列数が一致する必要があります。
- COPY オペレーションに指定したファイルに以下のいずれかの拡張子が含まれている場合、データを圧縮解除するためにパラメータを追加する必要はありません。
  - `.gz`
  - `.snappy`
  - `.bz2`

- Parquet および ORC ファイル形式からの COPY では、Redshift Spectrum とバケットアクセスが使用されます。これらの形式で COPY を使用するには、Amazon S3 の署名付き URL の使用をブロックする IAM ポリシーがないことを確認してください。Amazon Redshift によって生成された署名付き URL は 1 時間有効です。これにより、Amazon Redshift は Amazon S3 バケットからすべてのファイルをロードするのに十分な時間を確保できます。列指向形式から COPY でスキャンしたファイルごとに、一意の署名付き URL が生成されます。`s3:signatureAge` アクションを含むバケットポリシーの場合は、値を少なくとも 3,600,000 ミリ秒に設定してください。詳細については、拡張された VPC のルーティングで Amazon Redshift Spectrum を使用するを参照してください。

DATEFORMAT と TIMEFORMAT の文字列

COPY コマンドは、DATEFORMAT オプションと TIMEFORMAT オプションを使用して、ソースデータの日付と時刻の値を解析します。DATEFORMAT と TIMEFORMAT はフォーマットされた文字列であり、ソースデータの日付と時刻の値の形式と一致する必要があります。例えば、日付値が `Jan-01-1999` のソースデータをロードする COPY コマンドには、次の DATEFORMAT 文字列を含める必要があります。

```
COPY ...
            DATEFORMAT AS 'MON-DD-YYYY'
```

COPY データ変換の管理の詳細については、「データ変換パラメータ」を参照してください。

DATEFORMAT と TIMEFORMAT 文字列には、日時区切り記号 ('-'、'/'、':' など)、および次の「日付部分」と「時間部分」を含めることができます。

> **Note**
>
> 日付値または時刻値の形式を次の dateparts および timeparts と一致させることができない場合、または互いに異なる形式を使用する日付値および時刻値がある場合は、`'auto'` 引数を DATEFORMAT または TIMEFORMAT パラメータと共に使用します。`'auto'` 引数は、DATEFORMAT または TIMEFORMAT 文字列を使用する場合にサポートされない形式を認識します。詳細については、「DATEFORMAT と TIMEFORMAT で自動認識を使用する」を参照してください。

日付部分または時刻部分	意味
YY	年 (世紀なし)
YYYY	年 (世紀あり)
MM	月 (数字)
MON	月 (名前。省略名またはフルネーム)
DD	日 (数字)
HH または HH24	時 (24 時間制) Note SQL 関数の DATETIME の形式の文字列では、HH は HH12 と同じです。ただし、COPY の DATEFORMAT と TIMEFORMAT の文字列では、HH は HH24 と同じです。
HH12	時 (12 時間制)
MI	分
SS	Seconds
AM または PM	午前午後の指標 (12 時間制用)

デフォルトの日付形式は YYYY-MM-DD です。タイムゾーン (TIMESTAMP) 形式なしのデフォルトのタイプスタンプは YYYY-MM-DD HH:MI:SS です。タイムゾーン付きのデフォルトのタイムスタンプ (TIMESTAMPTZ) 形式は、YYYY-MM-DD HH:MI:SSOF です。ここで、OF は UTC からのオフセットです (例えば、–8:00。timeformat_string のタイムゾーン指定子 (TZ、tz または OF) を含めることはできません。秒 (SS) フィールドは、マイクロ秒レベルの詳細に至るまでの分数秒もサポートします。デフォルト形式と異なる形式の TIMESTAMPTZ データをロードするには、「auto」を指定します。

次に、ソースデータで検出できるサンプルの日付または時刻と、それらに対応する DATEFORMAT または TIMEFORMAT 文字列を示します。

ソースデータの日付または時刻の例	DATEFORMAT または TIMEFORMAT 構文
03/31/2003	DATEFORMAT AS 'MM/DD/YYYY'
March 31, 2003	DATEFORMAT AS 'MON DD, YYYY'
03.31.2003 18:45:05 03.31.2003 18:45:05.123456	TIMEFORMAT AS 'MM.DD.YYYY HH:MI:SS'

例

TIMEFORMAT の使用例については、「タイムスタンプまたは日付スタンプのロード」を参照してください。

DATEFORMAT と TIMEFORMAT で自動認識を使用する

DATEFORMAT または TIMEFORMAT パラメータの引数として `'auto'` を指定すると、Amazon Redshift ではソースデータの日付形式または時間形式を自動的に認識して変換します。例を以下に示します。

```
copy favoritemovies from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
dateformat 'auto';
```

DATEFORMAT と TIMEFORMAT に `'auto'` 引数を使用すると、COPY は「 DATEFORMAT と TIMEFORMAT の文字列」の表に示された日付と時間の形式を認識して変換します。`'auto'` 引数は、DATEFORMAT および TIMEFORMAT 文字列を使用する場合にサポートされない次の形式も認識します。

形式	有効な入力文字列の例
ISO 8601	2019-02-11T05:09:12.195Z

形式	有効な入力文字列の例
ユリウス日	J2451187
BC	Jan-08-95 BC
YYYYMMDD HHMISS	19960108 040809
YYMMDD HHMISS	960108 040809
YYYY.DDD	1996.008
YYYY-MM-DD HH:MI:SS.SSS	1996-01-08 04:05:06.789
DD Mon HH:MI:SS YYYY TZ	17 Dec 07:37:16 1997 PST
MM/DD/YYYY HH:MI:SS.SS TZ	12/17/1997 07:37:16.00 PST
YYYY-MM-DD HH:MI:SS+/-TZ	1997-12-17 07:37:16-08
DD.MM.YYYY HH:MI:SS TZ	12.17.1997 07:37:16.00 PST

自動認識は epochsecs および epochmillisecs はサポートしていません。

日付またはタイムスタンプの値が自動的に変換されるかどうかをテストするには、CAST 関数を使用して文字列を日付またはタイムスタンプの値に変換します。たとえば、次のコマンドはタイムスタンプ値 `'J2345678 04:05:06.789'` をテストします。

```
create table formattest (test char(21));
insert into formattest values('J2345678 04:05:06.789');
select test, cast(test as timestamp) as timestamp, cast(test as date) as date from
 formattest;

        test          |      timestamp       | date
----------------------+----------------------+------------
```

```
J2345678 04:05:06.789   1710-02-23 04:05:06 1710-02-23
```

DATE 列のソースデータに時間情報が含まれる場合、時間コンポーネントは切り捨てられます。TIMESTAMP 列のソースデータで時間情報が省略されている場合、時間コンポーネントには 00:00:00 が使用されます。

COPY の例

> **Note**
>
> 次の例では読みやすくするため、改行しています。credentials-args 文字列には改行やスペースを含めないでください。

トピック

- NOLOAD オプションを使用する COPY コマンド
- マルチバイト区切り文字と ENCODING オプションを含む COPY コマンド

DynamoDB テーブルから FAVORITEMOVIES をロードする

AWS SDK には、Movies という名前の DynamoDB テーブルを作成する簡単な例が含まれています。(この例については、DynamoDB の使用開始を参照)。次の例では、DynamoDB テーブルから Amazon Redshift MOVIES テーブルにデータをロードします。Amazon Redshift テーブルはすでにデータベースに存在する必要があります。

```
copy favoritemovies from 'dynamodb://Movies'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
readratio 50;
```

Amazon S3 バケットから LISTING をロードする

次の例では、Amazon S3 バケットから LISTING をロードします。COPY コマンドは `/data/listing/` フォルダ内のすべてのファイルをロードします。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listing/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

Amazon EMR クラスターから LISTING をロードする

次の例は、タブ区切りデータを含んだ SALES テーブルを Amazon EMR クラスター上の LZOP 圧縮ファイルからロードします。COPY は、`myoutput/`フォルダ内の `part-` で始まるすべてのファイルをロードします。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/part-*'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '\t' lzop;
```

次の例は、Amazon EMR クラスター上の JSON 形式データを SALES テーブルにロードします。COPY は、`myoutput/json/`フォルダ内のすべてのファイルをロードします。

```
copy sales
from 'emr://j-SAMPLE2B500FC/myoutput/json/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
JSON 's3://amzn-s3-demo-bucket/jsonpaths.txt';
```

マニフェストを使用し、データファイルを指定する

マニフェストを使用すると、COPY コマンドで必要なすべてのファイル、しかも必要なファイルのみを Amazon S3 からロードできます。異なるバケットの複数のファイル、または同じプレフィックスを共有しない複数のファイルをロードする必要がある場合も、マニフェストを使用できます。

たとえば、3 つのファイル `custdata1.txt`、`custdata2.txt`、`custdata3.txt`をロードする必要があるとします。次のコマンドを使用し、プレフィックスを指定することで、`amzn-s3-demo-bucket`内で `custdata` で始まるすべてのファイルをロードすることができます。

```
copy category
from 's3://amzn-s3-demo-bucket/custdata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

エラーのために 2 つのファイルしか存在しない場合、COPY はこれら 2 つのファイルのみロードして正常に終了しますが、データロードは未完了になります。バケット内に同じプレフィックスを使用する不要なファイル (`custdata.backup` などのファイル) がある場合、COPY はそのファイルもロードするため、不要なデータがロードされることになります。

必要なファイルがすべてロードされ、不要なデータがロードされないようにするには、マニフェストファイルを使用できます。マニフェストは、COPY コマンドで処理されるファイルをリストする、JSON 形式のテキストファイルです。例えば、次のマニフェストは前の例の 3 つのファイルをロードします。

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.1",
         "mandatory":true
      },
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.2",
         "mandatory":true
      },
      {
         "url":"s3://amzn-s3-demo-bucket/custdata.3",
         "mandatory":true
      }
   ]
```

```
}
```

オプションの mandatory フラグは、ファイルが存在しない場合に COPY を終了することを示します。デフォルト: false。mandatory 設定と関係なく、どのファイルも見つからない場合、COPY は終了します。この例で、ファイルが見つからない場合、COPY はエラーを返します。custdata.backup など、キープレフィックスのみ指定した場合に選択された可能性がある不要なファイルは、マニフェストにないため、無視されます。

以下の例に示すように、Parquet または ORC 形式のデータファイルからロードする場合、metaフィールドは必須です。

```
{
   "entries":[
      {
         "url":"s3://amzn-s3-demo-bucket1/orc/2013-10-04-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      },
      {
         "url":"s3://amzn-s3-demo-bucket2/orc/2013-10-05-custdata",
         "mandatory":true,
         "meta":{
            "content_length":99
         }
      }
   ]
}
```

次の例では、cust.manifestという名前のマニフェストを使用します。

```
copy customer
from 's3://amzn-s3-demo-bucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc
manifest;
```

マニフェストを使用し、異なるバケットからファイルをロードしたり、同じプレフィックスを共有しないファイルをロードしたりできます。次の例は、日付スタンプで始まる名前を持つファイルのデータをロードする JSON を示しています。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/2013-10-04-custdata.txt","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket/2013-10-05-custdata.txt","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket/2013-10-06-custdata.txt","mandatory":true},
    {"url":"s3://amzn-s3-demo-bucket/2013-10-07-custdata.txt","mandatory":true}
  ]
}
```

バケットがクラスターと同じ AWS リージョンにある限り、マニフェストは異なるバケットにあるファイルをリストできます。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata1.txt","mandatory":false},
    {"url":"s3://amzn-s3-demo-bucket2/custdata1.txt","mandatory":false},
    {"url":"s3://amzn-s3-demo-bucket2/custdata2.txt","mandatory":false}
  ]
}
```

パイプ区切りファイル (デフォルトの区切り記号) から LISTING をロードする

次の例は、オプションが指定されておらず、入力ファイルにデフォルトの区切り記号であるパイプ文字 (|) が含まれる簡単な場合を示しています。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

Parquet 形式の列指向データを使用した LISTING のロード

次の例では、parquet と呼ばれる Amazon S3 のフォルダからデータをロードします。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings/parquet/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as parquet;
```

ORC 形式の列指向データを使用した LISTING のロード

次の例では、`orc`と呼ばれる Amazon S3 のフォルダからデータをロードします。

```
copy listing
from 's3://amzn-s3-demo-bucket/data/listings/orc/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as orc;
```

オプションを使用した EVENT のロード

次の例では、パイプ区切りデータを EVENT テーブルにロードし、次のルールを適用します。

- 引用符のペアを使用して文字列を囲んでいる場合、引用符は削除されます。
- 空の文字列と空白を含む文字列は NULL 値としてロードされます。
- 5 件を超えるエラーが返されると、ロードは失敗します。
- タイムスタンプ値は指定された形式に準拠する必要があります。たとえば、`2008-09-26 05:43:12`は有効なタイムスタンプです。

```
copy event
from 's3://amzn-s3-demo-bucket/data/allevents_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
removequotes
emptyasnull
blanksasnull
maxerror 5
delimiter '|'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

固定幅のデータファイルから VENUE をロードする

```
copy venue
from 's3://amzn-s3-demo-bucket/data/venue_fw.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
fixedwidth 'venueid:3,venuename:25,venuecity:12,venuestate:2,venueseats:6';
```

前述の例では、次のサンプルデータと同じ形式のデータファイルを想定しています。次の例では、すべての列が仕様に示された幅と同じになるよう、スペースがプレースホルダーの役割を果たします。

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
```

```
5  Gillette Stadium         Foxborough  MA68756
```

CSV ファイルから CATEGORY をロードする

次の表に示す値とともに CATEGORY をロードしたいとします。

catid	catgroup	catname	catdesc
12	Shows	Musicals	ミュージカル劇
13	Shows	Plays	すべての「ミュージカル以外」の劇
14	Shows	Opera	すべてのオペラ、ライト、「ロック」オペラ
15	Concerts	Classical	すべてのシンフォニー、コンチェルト、合唱団のコンサート

次の例は、テキストファイルの内容をカンマで区切ったフィールド値とともに示しています。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,All "non-musical" theatre
14,Shows,Opera,All opera, light, and "rock" opera
15,Concerts,Classical,All symphony, concerto, and choir concerts
```

カンマ区切り入力を指定する DELIMITER パラメータを使用してファイルをロードした場合、一部の入力フィールドにカンマが含まれているため、COPY コマンドは失敗します。この問題を避けるには、CSV パラメータを使用し、カンマを含むフィールドを引用符で囲みます。引用符で囲んだ文字列内に引用符がある場合、引用符を 2 つにしてエスケープする必要があります。デフォルトの引用符は二重引用符です。したがって、二重引用符を追加して各二重引用符をエスケープする必要があります。新しい入力ファイルは次のようになります。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,"All ""non-musical"" theatre"
14,Shows,Opera,"All opera, light, and ""rock"" opera"
15,Concerts,Classical,"All symphony, concerto, and choir concerts"
```

ファイル名が `category_csv.txt` であるとした場合、次の COPY コマンドを使用してファイルをロードできます。

```
copy category
from 's3://amzn-s3-demo-bucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv;
```

または、入力内の二重引用符をエスケープする必要をなくすため、QUOTE AS パラメータを使用して異なる引用文字を指定できます。例えば、`category_csv.txt` の次のバージョンでは引用文字として "%" を使用しています。

```
12,Shows,Musicals,Musical theatre
13,Shows,Plays,%All "non-musical" theatre%
14,Shows,Opera,%All opera, light, and "rock" opera%
15,Concerts,Classical,%All symphony, concerto, and choir concerts%
```

次の COPY コマンドでは QUOTE AS を使用して `category_csv.txt` をロードします。

```
copy category
from 's3://amzn-s3-demo-bucket/data/category_csv.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
csv quote as '%';
```

IDENTITY 列の明示的値を使用して VENUE をロードする

次の例では、VENUE テーブルの作成時に少なくとも 1 列 (`venueid` 列など) は IDENTITY 列とするよう指定されたと想定しています。このコマンドは IDENTITY 列の自動生成値のデフォルトの IDENTITY 動作をオーバーライドし、代わりに venue.txt ファイルから明示的値をロードします。Amazon Redshift は、EXLICIT_IDS オプションを使用するときに、重複する IDENTITY 値がテーブルにロードされているかどうかを確認しません。

```
copy venue
from 's3://amzn-s3-demo-bucket/data/venue.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
explicit_ids;
```

パイプ区切りの GZIP ファイルから TIME をロードする

次の例では、パイプ区切りの GZIP ファイルから TIME テーブルをロードします。

```
copy time
```

```
from 's3://amzn-s3-demo-bucket/data/timerows.gz'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
gzip
delimiter '|';
```

タイムスタンプまたは日付スタンプのロード

次の例では、書式設定したタイムスタンプ付きのデータをロードします。

> **Note**
>
> TIMEFORMAT `HH:MI:SS` では、`SS`を超える小数点以下の秒数もマイクロ秒レベルまでサポートします。この例で使用されるファイル `time.txt` には `2009-01-12 14:15:57.119568` という 1 行が含まれています。

```
copy timestamp1
from 's3://amzn-s3-demo-bucket/data/time.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
timeformat 'YYYY-MM-DD HH:MI:SS';
```

このコピーの結果は次のとおりです。

```
select * from timestamp1;
c1
----------------------------
2009-01-12 14:15:57.119568
(1 row)
```

デフォルト値を使用してファイルのデータをロードする

次の例では、TICKIT データベースの VENUE テーブルのバリエーションを使用します。次のステートメントで定義される VENUE_NEW テーブルを考えてみます。

```
create table venue_new(
venueid smallint not null,
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

次の例に示すように、VENUESEATS 列に値がない venue_noseats.txt データファイルを考えてみます。

```
1|Toyota Park|Bridgeview|IL|
2|Columbus Crew Stadium|Columbus|OH|
3|RFK Stadium|Washington|DC|
4|CommunityAmerica Ballpark|Kansas City|KS|
5|Gillette Stadium|Foxborough|MA|
6|New York Giants Stadium|East Rutherford|NJ|
7|BMO Field|Toronto|ON|
8|The Home Depot Center|Carson|CA|
9|Dick's Sporting Goods Park|Commerce City|CO|
10|Pizza Hut Park|Frisco|TX|
```

次の COPY ステートメントはファイルからテーブルを正しくロードし、省略された列に DEFAULT 値 ('1000') を適用します。

```
copy venue_new(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_noseats.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

ロードされたテーブルを表示します。

```
select * from venue_new order by venueid;
venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |       1000
2 | Columbus Crew Stadium      | Columbus        | OH         |       1000
3 | RFK Stadium                | Washington      | DC         |       1000
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |       1000
5 | Gillette Stadium           | Foxborough      | MA         |       1000
6 | New York Giants Stadium    | East Rutherford | NJ         |       1000
7 | BMO Field                  | Toronto         | ON         |       1000
8 | The Home Depot Center      | Carson          | CA         |       1000
9 | Dick's Sporting Goods Park | Commerce City   | CO         |       1000
10 | Pizza Hut Park             | Frisco          | TX         |       1000
(10 rows)
```

次の例の場合、ファイルに VENUESEATS データが含まれていないことを想定すると共に、VENUENAME データも含まれていないことを想定します。

```
1||Bridgeview|IL|
2||Columbus|OH|
3||Washington|DC|
4||Kansas City|KS|
5||Foxborough|MA|
6||East Rutherford|NJ|
7||Toronto|ON|
8||Carson|CA|
9||Commerce City|CO|
10||Frisco|TX|
```

同じテーブル定義を使用すると、次の COPY ステートメントは失敗します。これは、VENUENAME に対して DEFAULT 値が指定されておらず、VENUENAME が NOT NULL 列であるためです。

```
copy venue(venueid, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

次に、IDENTITY 列を使用する VENUE テーブルのバリエーションを考えてみます。

```
create table venue_identity(
venueid int identity(1,1),
venuename varchar(100) not null,
venuecity varchar(30),
venuestate char(2),
venueseats integer not null default '1000');
```

前の例と同じように、VENUESEATS 列にはソースファイルに対応する値がないと想定します。次の COPY ステートメントは、IDENTITY データ値を自動生成する代わりに事前定義済みの IDENTITY データ値を含め、テーブルを正しくロードします。

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

次のステートメントは失敗します。これは、IDENTITY 列が含まれていない (列リストから VENUEID が抜けています) が、EXPLICIT_IDS パラメータが含まれているためです。

```
copy venue(venuename, venuecity, venuestate)
```

```
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' explicit_ids;
```

次のステートメントは失敗します。これは EXPLICIT_IDS パラメータが含まれていないためです。

```
copy venue(venueid, venuename, venuecity, venuestate)
from 's3://amzn-s3-demo-bucket/data/venue_pipe.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|';
```

ESCAPE データを使用したデータのコピー

次の例は、区切り文字 (この場合、パイプ文字) に一致する文字をロードする方法を示しています。入力ファイルで、ロードするすべてのパイプ文字 (|) がバックスラッシュ文字 (\) でエスケープされていることを確認します。次に、ESCAPE パラメータを使用してファイルをロードします。

```
$ more redshiftinfo.txt
1|public\|event\|dwuser
2|public\|sales\|dwuser

create table redshiftinfo(infoid int,tableinfo varchar(50));

copy redshiftinfo from 's3://amzn-s3-demo-bucket/data/redshiftinfo.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter '|' escape;

select * from redshiftinfo order by 1;
infoid |       tableinfo
-------+--------------------
1      | public|event|dwuser
2      | public|sales|dwuser
(2 rows)
```

ESCAPE パラメータを使用しないと、`Extra column(s) found`エラーが発生して、この COPY コマンドは失敗します。

> **Important**
>
> ESCAPE パラメータを指定した COPY を使用してデータをロードした場合、対応する出力ファイルを生成するには、UNLOAD コマンドにも ESCAPE パラメータを指定する必要が

あります。同様に、ESCAPE パラメータを使って UNLOAD を実行すると、同じデータを COPY する場合に ESCAPE を使用する必要があります。

JSON からのコピーの例

以下の例で、次のデータを含む CATEGORY テーブルをロードします。

CATID	CATGROUP	CATNAME	CATDESC
1	スポーツ	MLB	Major League Baseball
2	スポーツ	NHL	National Hockey League
3	スポーツ	NFL	National Football League
4	スポーツ	NBA	National Basketball Association
5	Concerts	Classical	すべてのシンフォニー、コンチェルト、合唱団のコンサート

トピック

- 'auto' オプションを使用した JSON データからのロード
- 'auto ignorecase' オプションを使用した JSON データからのロード
- JSONPaths ファイルを使用した JSON データからのロード
- JSONPaths ファイルを使用した JSON 配列からのロード

'auto' オプションを使用した JSON データからのロード

`'auto'` オプションを使用して JSON データからロードするには、JSON データが一連のオブジェクトで構成されている必要があります。キー名が列名と一致している必要がありますが、順序は関係ありません。`category_object_auto.json` という名前のファイルの内容を次に示します。

```
{
    "catdesc": "Major League Baseball",
    "catid": 1,
    "catgroup": "Sports",
    "catname": "MLB"
```

```
}
{
    "catgroup": "Sports",
    "catid": 2,
    "catname": "NHL",
    "catdesc": "National Hockey League"
}{
    "catid": 3,
    "catname": "NFL",
    "catgroup": "Sports",
    "catdesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "catid": 4,
    "catgroup": "Sports",
    "catname": "NBA",
    "catdesc": "National Basketball Association"
}
{
    "catid": 5,
    "catgroup": "Shows",
    "catname": "Musicals",
    "catdesc": "All symphony, concerto, and choir concerts"
}
```

前の例の JSON データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_auto.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto';
```

'auto ignorecase' オプションを使用した JSON データからのロード

`'auto ignorecase'` オプションを使用して JSON データからロードするには、JSON データが一連のオブジェクトで構成されている必要があります。キー名の大文字と小文字は列名と一致する必要がなく、順序は関係ありません。`category_object_auto-ignorecase.json` という名前のファイルの内容を次に示します。

```
{
    "CatDesc": "Major League Baseball",
```

```
    "CatID": 1,
    "CatGroup": "Sports",
    "CatName": "MLB"
}
{
    "CatGroup": "Sports",
    "CatID": 2,
    "CatName": "NHL",
    "CatDesc": "National Hockey League"
}{
    "CatID": 3,
    "CatName": "NFL",
    "CatGroup": "Sports",
    "CatDesc": "National Football League"
}
{
    "bogus": "Bogus Sports LLC",
    "CatID": 4,
    "CatGroup": "Sports",
    "CatName": "NBA",
    "CatDesc": "National Basketball Association"
}
{
    "CatID": 5,
    "CatGroup": "Shows",
    "CatName": "Musicals",
    "CatDesc": "All symphony, concerto, and choir concerts"
}
```

前の例の JSON データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_auto ignorecase.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 'auto ignorecase';
```

JSONPaths ファイルを使用した JSON データからのロード

JSON データオブジェクトが列名に直接対応していない場合、JSONPaths ファイルを使用して、JSON 要素を列にマッピングすることができます。JSON ソースデータの順序は重要ではありませんが、JSONPaths ファイルの式の順序は列の順序と一致している必要があります。`category_object_paths.json` という名前の次のようなデータファイルがあるとします。

```
{
    "one": 1,
    "two": "Sports",
    "three": "MLB",
    "four": "Major League Baseball"
}
{
    "three": "NHL",
    "four": "National Hockey League",
    "one": 2,
    "two": "Sports"
}
{
    "two": "Sports",
    "three": "NFL",
    "one": 3,
    "four": "National Football League"
}
{
    "one": 4,
    "two": "Sports",
    "three": "NBA",
    "four": "National Basketball Association"
}
{
    "one": 6,
    "two": "Shows",
    "three": "Musicals",
    "four": "All symphony, concerto, and choir concerts"
}
```

`category_jsonpath.json` という名前の次の JSONPaths ファイルで、ソースデータをテーブルの列にマッピングします。

```
{
    "jsonpaths": [
        "$['one']",
        "$['two']",
        "$['three']",
        "$['four']"
    ]
}
```

前の例の JSON データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_paths.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://amzn-s3-demo-bucket/category_jsonpath.json';
```

JSONPaths ファイルを使用した JSON 配列からのロード

一連の配列で構成される JSON データからロードするには、JSONPaths ファイルを使用して、配列の要素を列にマッピングする必要があります。`category_array_data.json` という名前の次のようなデータファイルがあるとします。

```
[1,"Sports","MLB","Major League Baseball"]
[2,"Sports","NHL","National Hockey League"]
[3,"Sports","NFL","National Football League"]
[4,"Sports","NBA","National Basketball Association"]
[5,"Concerts","Classical","All symphony, concerto, and choir concerts"]
```

`category_array_jsonpath.json` という名前の次の JSONPaths ファイルで、ソースデータをテーブルの列にマッピングします。

```
{
    "jsonpaths": [
        "$[0]",
        "$[1]",
        "$[2]",
        "$[3]"
    ]
}
```

前の例の JSON データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_array_data.json'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
json 's3://amzn-s3-demo-bucket/category_array_jsonpath.json';
```

Avro の例からのコピー

以下の例で、次のデータを含む CATEGORY テーブルをロードします。

CATID	CATGROUP	CATNAME	CATDESC
1	スポーツ	MLB	Major League Baseball
2	スポーツ	NHL	National Hockey League
3	スポーツ	NFL	National Football League
4	スポーツ	NBA	National Basketball Association
5	Concerts	Classical	すべてのシンフォニー、コンチェルト、合唱団のコンサート

トピック

- 'auto' オプションを使用した Avro データからのロード
- 'auto ignorecase' オプションを使用した Avro データからのロード
- JSONPaths ファイルを使用した Avro データからのロード

'auto' オプションを使用した Avro データからのロード

`'auto'` 引数を使用して Avro データからロードするには、Avro スキーマのフィールド名が列名と一致している必要があります。ただし、`'auto'` 引数を使用する場合は、順序は関係ありません。次で `category_auto.avro` という名前のファイルのスキーマを示します。

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "catid", "type": "int"},
        {"name": "catdesc", "type": "string"},
        {"name": "catname", "type": "string"},
        {"name": "catgroup", "type": "string"},
}
```

Avro ファイルのデータはバイナリ形式であるため、人には読み取り不可能です。次に、`category_auto.avro`ファイルのデータの JSON 形式を示します。

```
{
```

```
    "catid": 1,
    "catdesc": "Major League Baseball",
    "catname": "MLB",
    "catgroup": "Sports"
}
{
    "catid": 2,
    "catdesc": "National Hockey League",
    "catname": "NHL",
    "catgroup": "Sports"
}
{
    "catid": 3,
    "catdesc": "National Basketball Association",
    "catname": "NBA",
    "catgroup": "Sports"
}
{
    "catid": 4,
    "catdesc": "All symphony, concerto, and choir concerts",
    "catname": "Classical",
    "catgroup": "Concerts"
}
```

前の例の Avro データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_auto.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto';
```

'auto ignorecase' オプションを使用した Avro データからのロード

`'auto ignorecase'` 引数を使用して Avro データからロードするために、Avro スキーマのフィールド名の大文字と小文字が列名の大文字と小文字を一致させる必要はありません。ただし、`'auto ignorecase'` 引数を使用する場合は、順序は関係ありません。次で `category_auto-ignorecase.avro` という名前のファイルのスキーマを示します。

```
{
    "name": "category",
    "type": "record",
    "fields": [
```

```
        {"name": "CatID", "type": "int"},
        {"name": "CatDesc", "type": "string"},
        {"name": "CatName", "type": "string"},
        {"name": "CatGroup", "type": "string"},
}
```

Avro ファイルのデータはバイナリ形式であるため、人には読み取り不可能です。次に、`category_auto-ignorecase.avro`ファイルのデータの JSON 形式を示します。

```
{
   "CatID": 1,
   "CatDesc": "Major League Baseball",
   "CatName": "MLB",
   "CatGroup": "Sports"
}
{
   "CatID": 2,
   "CatDesc": "National Hockey League",
   "CatName": "NHL",
   "CatGroup": "Sports"
}
{
   "CatID": 3,
   "CatDesc": "National Basketball Association",
   "CatName": "NBA",
   "CatGroup": "Sports"
}
{
   "CatID": 4,
   "CatDesc": "All symphony, concerto, and choir concerts",
   "CatName": "Classical",
   "CatGroup": "Concerts"
}
```

前の例の Avro データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_auto-ignorecase.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format as avro 'auto ignorecase';
```

JSONPaths ファイルを使用した Avro データからのロード

Avro スキーマのフィールド名が列名に直接対応しない場合、JSONPaths ファイルを使用してスキーマの要素を列にマッピングできます。JSONPaths のファイル式の順序は、列の順序に一致している必要があります。

前の例と同じデータだが次のスキーマを持った `category_paths.avro` という名前のデータファイルがあるとします。

```
{
    "name": "category",
    "type": "record",
    "fields": [
        {"name": "id", "type": "int"},
        {"name": "desc", "type": "string"},
        {"name": "name", "type": "string"},
        {"name": "group", "type": "string"},
        {"name": "region", "type": "string"}
     ]
}
```

`category_path.avropath` という名前の次の JSONPaths ファイルで、ソースデータをテーブルの列にマッピングします。

```
{
    "jsonpaths": [
        "$['id']",
        "$['group']",
        "$['name']",
        "$['desc']"
    ]
}
```

前の例の Avro データファイルからロードするには、次の COPY コマンドを実行します。

```
copy category
from 's3://amzn-s3-demo-bucket/category_object_paths.avro'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
format avro 's3://amzn-s3-demo-bucket/category_path.avropath ';
```

ESCAPE オプションを指定する COPY 用のファイルの準備

次の例では、ESCAPE パラメータを指定した COPY コマンドでデータを Amazon Redshift テーブルにインポートする前に、データを準備して改行文字を "エスケープする" 方法について説明します。データを準備して改行文字を区切らないと、Amazon Redshift は COPY コマンドの実行時にロードエラーを返します。通常、改行文字はレコード区切り文字として使用されるためです。

例えば、ファイルまたは外部テーブルの列を Amazon Redshift テーブルにコピーするとします。そのファイルまたは列に XML 形式のコンテンツまたは同様なデータが含まれている場合、コンテンツの一部である改行文字 (\n) はすべてバックスラッシュ文字 (\) でエスケープする必要があります。

埋め込み改行文字を含むファイルまたはテーブルは、比較的簡単に一致パターンを提供します。テキストファイル > の次の例に示すように、ほとんどの場合、それぞれの埋め込み改行文字は ' ' 文字の後に続きます。場合によっては、その間に空白文字 (`nlTest1.txt` またはタブ) が入ります。

```
$ cat nlTest1.txt
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>|1000
<xml>
</xml>|2000
```

次の例では、テキスト処理ユーティリティを実行してソースファイルを事前処理し、必要な場所にエスケープ文字を挿入できます (| 文字は、Amazon Redshift テーブルにコピーされるときに列データを区切る区切り記号として使用されます。)

```
$ sed -e ':a;N;$!ba;s/>[[:space:]]*\n/>\\\n/g' nlTest1.txt > nlTest2.txt
```

同様に、Perl を使用しても同じような処理を行うことができます。

```
cat nlTest1.txt | perl -p -e 's/>\s*\n/>\\\n/g' > nlTest2.txt
```

`nlTest2.txt` ファイルから Amazon Redshift へのデータロードに対応するため、Amazon Redshift に 2 列のテーブルを作成しました。最初の列 c1 は文字の列です。この列は `nlTest2.txt` ファイルからの XML 形式のコンテンツを保持します。2 番目の列 c2 は同じファイルからロードされる整数値を保持します。

sed コマンドを実行した後、ESCAPE パラメータを使用して `nlTest2.txt` ファイルから Amazon Redshift テーブルにデータを正しくロードすることができます。

> **Note**
>
> COPY コマンドに ESCAPE パラメータを指定すると、バックスラッシュ文字を含むいくつかの特殊文字をエスケープします (改行文字を含む)。

```
copy t2 from 's3://amzn-s3-demo-bucket/data/nlTest2.txt'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
escape
delimiter as '|';

select * from t2 order by 2;

c1             |  c2
-------------+------
<xml start>
<newline characters provide>
<line breaks at the end of each>
<line in content>
</xml>
| 1000
<xml>
</xml>         | 2000
(2 rows)
```

外部データベースからエクスポートされるデータファイルも同様に準備できます。例えば、Oracle データベースの場合、Amazon Redshift にコピーするテーブルの各対象列に対して REPLACE 関数を使用できます。

```
SELECT c1, REPLACE(c2, \n',\\n' ) as c2 from my_table_with_xml
```

また、通常大量のデータを処理する多くのデータベースエクスポートツールや抽出、変換、ロード (ETL) ツールは、エスケープ文字と区切り文字を指定するオプションを備えています。

シェープファイルを Amazon Redshift にロードする

次の例は、COPY を使用して Esri シェープファイルをロードする方法を示しています。シェープファイルのロードについての詳細は、「シェープファイルを Amazon Redshift にロードする」を参照してください。

シェープファイルのロード

以下のステップは、COPY コマンドを使用して Amazon S3 から OpenStreetMap データを取り込む方法を示しています。この例では、ノルウェーのGeofabrik のダウンロードサイトから得たシェープファイルアーカイブが、AWSリージョンのプライベート Amazon S3 バケットにアップロードされていることを前提としています。.shp、.shx、および .dbf ファイルは、同じ Amazon S3 プレフィックスとファイル名を共有する必要があります。

簡素化されていないデータの取り込み

次のコマンドは、簡略化せずに最大ジオメトリのサイズに収まるテーブルを作成し、データを取り込みます。好みの GIS ソフトウエアで gis_osm_natural_free_1.shp を開き、このレイヤーの列を調べます。デフォルトでは、IDENTITY 列または GEOMETRY 列のいずれかが最初になります。GEOMETRY 列が最初である場合、次のようにテーブルを作成できます。

```
CREATE TABLE norway_natural (
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

または、IDENTITY 列が最初であるとき、次のようにテーブルを作成できます。

```
CREATE TABLE norway_natural_with_id (
   fid INT IDENTITY(1,1),
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

これで、COPY を使用してデータを取り込むことができます。

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
```

```
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural' completed, 83891 record(s) loaded successfully
```

または、次に示すように、データを取り込むことができます。

```
COPY norway_natural_with_id FROM 's3://bucket_name/shapefiles/norway/
gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO: Load into table 'norway_natural_with_id' completed, 83891 record(s) loaded
 successfully.
```

簡素化されたデータの取り込み

次のコマンドは、テーブルを作成し、最大ジオメトリのサイズに収まらないデータを簡略化せずに取り込もうとします。gis_osm_water_a_free_1.shp シェープファイルを調べて、次のように適切なテーブルを作成します。

```
CREATE TABLE norway_water (
   wkb_geometry GEOMETRY,
   osm_id BIGINT,
   code INT,
   fclass VARCHAR,
   name VARCHAR);
```

COPY コマンドを実行すると、エラーが発生します。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
ERROR:  Load into table 'norway_water' failed.  Check 'stl_load_errors' system table
 for details.
```

STL_LOAD_ERRORS をクエリすると、ジオメトリが大きすぎることがわかります。

```
SELECT line_number, btrim(colname), btrim(err_reason) FROM stl_load_errors WHERE query
 = pg_last_copy_id();
 line_number |    btrim     |                                 btrim
-------------+--------------
+-----------------------------------------------------------------------
```

```
     1184705 | wkb_geometry | Geometry size: 1513736 is larger than maximum supported
 size: 1048447
```

これを克服するために、SIMPLIFY AUTO パラメータが COPY コマンドに追加され、形状が簡略化されています。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989196 record(s) loaded successfully.
```

簡略化された行とジオメトリを表示するには、SVL_SPATIAL_SIMPLIFY をクエリします。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
    20 |     1664115 |                -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

自動的に計算された許容値よりも低い許容値で SIMPLIFY AUTO max_tolerance を使用すると、おそらく取り込みエラーが発生します。この場合、MAXERROR を使用してエラーを無視します。

```
COPY norway_water FROM 's3://bucket_name/shapefiles/norway/gis_osm_water_a_free_1.shp'
FORMAT SHAPEFILE
SIMPLIFY AUTO 1.1E-05
MAXERROR 2
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';

INFO:  Load into table 'norway_water' completed, 1989195 record(s) loaded successfully.
INFO:  Load into table 'norway_water' completed, 1 record(s) could not be loaded.
 Check 'stl_load_errors' system table for details.
```

SVL_SPATIAL_SIMPLIFY を再度クエリして、COPY がロードできなかったレコードを特定します。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+------------------+-------------+------------+------------
+-----------------
    29 |     1184704 |           1.1e-05 |      1513736 | f          |          0 |
         0
    29 |     1664115 |           1.1e-05 |      1233456 | t          |     794432 |
     1.1e-05
```

この例では、最初のレコードがうまく収まらなかったため、simplified 列に false が表示されています。2 番目のレコードは、指定された許容値内でロードされました。ただし、最終的なサイズは、最大許容値を指定せずに自動的に計算された許容値を使用するよりも大きくなります。

圧縮シェープファイルからのロード

Amazon Redshift COPY は、圧縮されたシェープファイルからのデータの取り込みをサポートしています。すべてのシェープファイルコンポーネントには、同じ Amazon S3 プレフィックスと同じ圧縮サフィックスが必要です。例として、前の例からデータをロードするとします。この場合、ファイル gis_osm_water_a_free_1.shp.gz、gis_osm_water_a_free_1.dbf.gz、および gis_osm_water_a_free_1.shx.gz は同じ Amazon S3 ディレクトリを共有する必要があります。COPY コマンドには GZIP オプションが必要です。FROM 句では、次に示すように、正しい圧縮ファイルを指定する必要があります。

```
COPY norway_natural FROM 's3://bucket_name/shapefiles/norway/compressed/
gis_osm_natural_free_1.shp.gz'
FORMAT SHAPEFILE
GZIP
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural' completed, 83891 record(s) loaded successfully.
```

列の順序が異なるテーブルへのデータのロード

最初の列が GEOMETRY でないテーブルがある場合は、列マッピングを使用して列をターゲットテーブルにマッピングできます。例えば、最初の列として osm_id を指定してテーブルを作成します。

```
CREATE TABLE norway_natural_order (
   osm_id BIGINT,
   wkb_geometry GEOMETRY,
   code INT,
```

```
    fclass VARCHAR,
    name VARCHAR);
```

次に、列マッピングを使用してシェープファイルを取り込みます。

```
COPY norway_natural_order(wkb_geometry, osm_id, code, fclass, name)
FROM 's3://bucket_name/shapefiles/norway/gis_osm_natural_free_1.shp'
FORMAT SHAPEFILE
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/MyRoleName';
INFO:  Load into table 'norway_natural_order' completed, 83891 record(s) loaded
 successfully.
```

Geogry 列を含めたデータのテーブルへのロード

GEOGRAPHY 列を含むテーブルがある場合は、まず、この列を GEOMETRY 列に取り込んだ上で、オブジェクトを GEOGRAPHY オブジェクトにキャストします。例えば、シェープファイルを GEOMETRY 列にコピーした後で、テーブルを変更して GEOGRAPHY データ型の列を追加します。

```
ALTER TABLE norway_natural ADD COLUMN wkb_geography GEOGRAPHY;
```

次に、ジオメトリをジオグラフィに変換します。

```
UPDATE norway_natural SET wkb_geography = wkb_geometry::geography;
```

オプションで、GEOMETRY列を削除できます。

```
ALTER TABLE norway_natural DROP COLUMN wkb_geometry;
```

NOLOAD オプションを使用する COPY コマンド

実際にデータをロードする前にデータファイルを検証するには、COPY コマンドで NOLOAD オプションを使用します。Amazon Redshift は入力ファイルを解析し、発生したエラーをすべて表示します。次の例では NOLOAD オプションを使用しており、実際にテーブルに読み込まれた行はありません。

```
COPY public.zipcode1
FROM 's3://amzn-s3-demo-bucket/mydata/zipcode.csv'
DELIMITER ';'
IGNOREHEADER 1 REGION 'us-east-1'
NOLOAD
```

```
CREDENTIALS 'aws_iam_role=arn:aws:iam::123456789012:role/myRedshiftRole';

Warnings:
Load into table 'zipcode1' completed, 0 record(s) loaded successfully.
```

マルチバイト区切り文字と ENCODING オプションを含む COPY コマンド

次の例では、マルチバイトデータを含む Amazon S3 ファイルから LATIN1 をロードします。COPY コマンドは、ISO-8859-1 としてエンコードされた入力ファイルのフィールドを区切る区切り文字として 8 進形式 (`\302\246\303\254`) を指定します。UTF-8 で同じ区切り文字を指定するには、`DELIMITER '¦ì'` を指定します。

```
COPY latin1
FROM 's3://amzn-s3-demo-bucket/multibyte/myfile'
IAM_ROLE 'arn:aws:iam::123456789012:role/myRedshiftRole'
DELIMITER '\302\246\303\254'
ENCODING ISO88591
```

CREATE DATABASE

新しいデータベースを作成します。

データベースを作成するには、スーパーユーザーであるか、CREATEDB 権限を持っている必要があります。ゼロ ETL 統合に関連付けられたデータベースを作成するには、スーパーユーザーであるか、CREATEDB 権限と CREATEUSER 権限の両方を持っている必要があります。

トランザクションブロック (BEGIN ... END) 内で CREATE DATABASE を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

構文

```
CREATE DATABASE database_name
[ { [ WITH ]
    [ OWNER [=] db_owner ]
    [ CONNECTION LIMIT { limit | UNLIMITED } ]
    [ COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE } ]
    [ ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT } ]
  }
```

```
  | { [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid }
  | { FROM { { ARN '<arn>' } { WITH DATA CATALOG SCHEMA '<schema>' | WITH NO DATA
 CATALOG SCHEMA } }
             | { INTEGRATION '<integration_id>' [ DATABASE '<source_database>' ] [SET
 {REFRESH_INTERVAL <interval>} ] } }
  | { IAM_ROLE  {default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' } }
]
```

パラメータ

database_name

新しいデータベースの名前。有効な名前の詳細については、「名前と識別子」を参照してください。

WiTH

オプションキーワード

OWNER

データベース所有者を指定します。

=

オプションの文字。

db_owner

データベース所有者のユーザー名。

CONNECTION LIMIT { limit | UNLIMITED }

ユーザーが同時に開けるデータベース接続の最大数。この制限はスーパーユーザーには適用されません。同時接続の最大数を許可するには、UNLIMITED キーワードを使用します。ユーザーごとの接続数の制限が適用される場合もあります。詳細については、「CREATE USER」を参照してください。デフォルトは UNLIMITED です。現在の接続を確認するには、STV_SESSIONSシステムビューに対してクエリを実行します。

> **Note**
>
> ユーザーとデータベースの両方の接続制限が適用される場合は、ユーザーが接続しようとしたときに、両方の制限内に未使用の接続スロットがなければなりません。

COLLATE { CASE_SENSITIVE | CASE_INSENSITIVE }

文字列の検索または比較が CASE_SENSITIVE か CASE_INSENSITIVE かを指定する句。デフォルトは CASE_SENSITIVE です。

ISOLATION LEVEL { SERIALIZABLE | SNAPSHOT }

データベースに対してクエリを実行するときに使用される分離レベルを指定する句。

- SERIALIZABLE 分離 – 同時実行トランザクションの完全な直列化機能を提供します。詳細については、「直列化可能分離」を参照してください。
- SNAPSHOT 分離 – 更新および削除の競合に対する保護機能を備えた分離レベルを提供します。これは、プロビジョニングされたクラスターまたはサーバーレス名前空間で作成されたデフォルトのデータベースです。

データベースが実行されている同時実行モデルを、次のように表示できます。

- STV_DB_ISOLATION_LEVEL カタログビューのクエリを実行します。詳細については、「STV_DB_ISOLATION_LEVEL」を参照してください。

```
SELECT * FROM stv_db_isolation_level;
```

- PG_DATABASE_INFO ビューに対してクエリを実行します。

```
SELECT datname, datconfig FROM pg_database_info;
```

データベースごとの分離レベルは、`concurrency_model` キーの隣に表示されます。1 の値は、SNAPSHOT を表します。2 の値は、SERIALIZABLE を表します。

Amazon Redshift データベースでは、SERIALIZABLE と SNAPSHOT の両方の分離は、直列化可能な分離レベルのタイプです。つまり、ダーティリード、ノンリピータブルリード、ファントムリードは、SQL 標準に従って防止されます。どちらの分離レベルでも、トランザクションが開始されたときに存在するデータのスナップショットに対してトランザクションが実行され、他のトランザクションがそのスナップショットを変更できないことが保証されます。ただし、SNAPSHOT 分離は、異なるテーブル行での書き込みスキューの挿入と更新を妨げないため、完全な直列化機能を提供しません。

次のシナリオは、SNAPSHOT 分離レベルを使用した書き込みスキューの更新を示しています。`Numbers` という名前のテーブルには、`0` と `1` の値を含む `digits` という名前の列が含まれています。各ユーザーの UPDATE ステートメントは、他のユーザーと重複しません。ただし、`0`

と 1 の値はスワップされます。実行する SQL は、このタイムラインに従い次の結果になります。

時間	ユーザー 1 アクション	ユーザー 2 アクション
1	BEGIN;	
2		BEGIN;
3	SELECT * FROM Numbers; ```digits ------ 0 1```	
4		SELECT * FROM Numbers; ```digits ------ 0 1```
5	UPDATE Numbers SET digits=0 WHERE digits=1;	

時間	ユーザー 1 アクション	ユーザー 2 アクション
6	SELECT * FROM Numbers ``` digits ------ 0 0 ```	
7	COMMIT	
8		Update Numbers SET digits=1 WHERE digits=0;
9		SELECT * FROM Numbers; ``` digits ------ 1 1 ```
10		COMMIT;

時間	ユーザー 1 アクション	ユーザー 2 アクション

11	SELECT * FROM Numbers
	`digits`
	`------`
	`1`
	`0`	

12		SELECT * FROM Numbers;
		`digits`
		`------`
		`1`
		`0`

直列化可能な分離を使用して同じシナリオを実行する場合、Amazon Redshift は直列化可能な分離違反によりユーザー 2 を終了し、エラー `1023` を返します。詳細については、「[直列化可能分離エラーを修正する方法]」を参照してください。この場合、ユーザー 1 だけが正常にコミットできます。すべてのワークロードが直列化可能な分離を要件として必要とするわけではありません。その場合、スナップショット分離はデータベースのターゲット分離レベルとして十分です。

FROM ARN '<ARN>'

データベースの作成に使用する AWS Glue データベース ARN。

{ DATA CATALOG SCHEMA '<schema>' | WITH NO DATA CATALOG SCHEMA }

> **Note**
>
> このパラメータは、CREATE DATABASE コマンドで FROM ARN パラメータも使用する場合にのみ適用されます。

AWS Glue Data Catalog でオブジェクトにアクセスしやすくするためにスキーマを使用してデータベースを作成するかどうかを指定します。

FROM INTEGRATION '<integration_id>' [DATABASE '<source_database>'] [SET {REFRESH_INTERVAL <interval>}]

ゼロ ETL 統合識別子を使用してデータベースを作成するかどうかを指定します。SVV_INTEGRATION システムビューから `integration_id` を取得できます。Aurora PostgreSQL のゼロ ETL 統合では、SVV_INTEGRATION から取得できる `source_database` 名前を指定する必要があります。SET REFRESH_INTERVAL 句は、ゼロ ETL ソースからターゲットデータベースにデータを更新するおおよその時間間隔を秒単位で設定します。この値は、ソースタイプが Aurora MySQL、Aurora PostgreSQL、または RDS for MySQL のゼロ ETL 統合では、0～432,000 秒 (5 日) に設定できます。Amazon DynamoDB ゼロ ETL 統合の場合、値は 900～432,000 秒 (15 分～5 日) に設定できます。ソースタイプが Aurora MySQL、Aurora PostgreSQL、または RDS for MySQL のゼロ ETL 統合では、デフォルトの `interval` はゼロ (0) 秒です。Amazon DynamoDB ゼロ ETL 統合の場合、デフォルトの `interval` は 900 秒 (15 分) です。

例については、「ゼロ ETL 統合の結果を受け取るデータベースを作成する」を参照してください。ゼロ ETL 統合を使用したデータベースの作成の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift でのデスティネーションデータベースの作成」を参照してください。

IAM_ROLE { default | 'SESSION' | 'arn:aws:iam::*<AWS #####-id>*:role/*<role-name>*' }

> **Note**
>
> このパラメータは、CREATE DATABASE コマンドで FROM ARN パラメータも使用する場合にのみ適用されます。

CREATE DATABASE コマンドの実行時にクラスターに関連付けられた IAM ロールを指定すると、Amazon Redshift はデータベースに対してクエリを実行するときにロールの認証情報を使用します。

`default` キーワードを指定することは、デフォルトとして設定されてクラスターに関連付けられている IAM ロールを使用することを意味します。

フェデレーション ID を使用して Amazon Redshift クラスターに接続し、このコマンドを使用して作成された外部スキーマからテーブルにアクセスする場合は、`'SESSION'` を使用します。フェデレーション ID の使用例については、フェデレーション ID の設定方法を説明している「フェデレーション ID を使用して、ローカルリソースと Amazon Redshift Spectrum の外部テーブルへの Amazon Redshift アクセスを管理する」を参照してください。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。少なくとも、IAM ロールには、Amazon S3 バケットで LIST オペレーションを実行してアクセスを受ける許可と、バケットに含まれる Amazon S3 オブジェクトで GET オペレーションを実行する許可が必要です。データベース用の AWS Glue Data Catalog を使用してデータベースを作成するときに IAM_ROLE を使用する方法の詳細については、「コンシューマーとして Lake Formation 管理のデータ共有を使用する」を参照してください。

以下に ARN が 1 つの場合の IAM_ROLE パラメータ文字列の構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

ロールを連鎖することで、クラスターは別のアカウントに属している可能性がある別の IAM ロールを引き受けることができます。最大10 個までのロールを連鎖できます。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

この IAM ロールに次のような IAM アクセス許可ポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
```

```
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

フェデレーションクエリで使用する IAM ロールを作成するステップについては、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。

> **Note**
>
> 連鎖したロールのリストには空白を含めないでください。

以下に連鎖された 3 つのロールの構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

データ共有で CREATE DATABASE を使用するための構文

次の構文で、同じ AWS アカウント内のデータ共有からデータベースを作成するために使用される、CREATE DATABASE コマンドについて確認できます。

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF [ ACCOUNT account_id ]
 NAMESPACE namespace_guid
```

次の構文で、AWSアカウント間のデータ共有からデータベースを作成するために使用される,CREATE DATABASE コマンドについて確認できます。

```
CREATE DATABASE database_name
[ [ WITH PERMISSIONS ] FROM DATASHARE datashare_name ] OF ACCOUNT account_id
 NAMESPACE namespace_guid
```

データ共有で CREATE DATABASE を使用するためのパラメータ

FROM DATASHARE

データ共有の場所を示すキーワード。

datashare_name

コンシューマーデータベースが作成されるデータ共有の名前。

WITH PERMISSIONS

データ共有から作成されたデータベースに、個々のデータベースオブジェクトにアクセスするためのオブジェクトレベルの許可が必要であることを指定します。この句を指定しないと、データベースに対する USAGE アクセス許可を付与されたユーザーまたはロールは、データベース内のすべてのデータベースオブジェクトに自動的にアクセスできるようになります。

NAMESPACE namespace_guid

データ共有が属しているプロデユーサ名前空間を指定する値。

ACCOUNT account_id

データ共有が属しているプロデユーサアカウントを指定する値。

データ共有のための CREATE DATABASE の使用上の注意

データベーススーパーユーザーとして、CREATE DATABASE を使用して AWS アカウント内のデータ共有からデータベースを作成する場合は、NAMESPACE オプションを指定します。ACCOUNT オプションは省略可能です。CREATE DATABASE を使用して AWS アカウント間のデータ共有からデータベースを作成する場合は、プロデューサーで ACCOUNT と NAMESPACE の両方を指定します。

コンシューマークラスター上の 1 つのデータ共有に対して、作成できるコンシューマデータベースは 1 つだけです。同じデータ共有を参照する、複数のコンシューマーデータベースを作成することはできません。

AWS Glue Data Catalog の CREATE DATABASE

AWS Glue データベース ARN を使用してデータベースを作成するには、CREATE DATABASE で ARN を指定します。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA;
```

オプションで、IAM_ROLE パラメータに値を指定することもできます。パラメータおよび許容される値の詳細については、「パラメータ」を参照してください。

IAM ロールを使用して ARN からデータベースを作成する方法を示す例は、以下のとおりです。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE <iam-role-arn>
```

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH NO DATA CATALOG SCHEMA
 IAM_ROLE default;
```

DATA CATALOG SCHEMA を使用してデータベースを作成することもできます。

```
CREATE DATABASE sampledb FROM ARN <glue-database-arn> WITH DATA CATALOG SCHEMA
 <sample_schema> IAM_ROLE default;
```

ゼロ ETL 統合の結果を受け取るデータベースを作成する

ゼロ ETL 統合 ID を使用してデータベースを作成するには、CREATE DATABASE コマンドで `integration_id` を指定します。

```
CREATE DATABASE destination_db_name FROM INTEGRATION 'integration_id';
```

例えば、まず SVV_INTEGRATION から統合 ID を取得します。

```
SELECT integration_id FROM SVV_INTEGRATION;
```

次に、取得した統合 ID のいずれかを使用して、ゼロ ETL 統合を受け取るデータベースを作成します。

```
CREATE DATABASE sampledb FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111';
```

ゼロ ETL 統合のソースデータベースが必要な場合は、例えば以下のように指定します。

```
CREATE DATABASE sampledb FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'
 DATABASE 'sourcedb';
```

データベースの更新間隔を設定することもできます。例えば、ゼロ ETL 統合のソースからのデータの更新間隔を 7,200 秒にするには以下のように設定します。

```
CREATE DATABASE myacct_mysql FROM INTEGRATION 'a1b2c3d4-5678-90ab-cdef-EXAMPLE11111'
 SET REFRESH_INTERVAL 7200;
```

SVV_INTEGRATION カタログビューに、integration_id、target_database、source、refresh_interval などのゼロ ETL 統合に関する情報がないかクエリします。

```
SELECT * FROM svv_integration;
```

CREATE DATABASE の制限

Amazon Redshift は、データベースに次の制限を適用します。

- クラスターごとのユーザー定義データベースは最大 60 個です。
- データベース名は最大 127 バイトです。
- データベース名を予約語にすることはできません。

データベースの照合

照合とは、データベースエンジンが SQL の文字型データを比較およびソートする方法を定義するための一連の規則です。大文字と小文字が区別されない照合は、最もよく使用されます。Amazon Redshift では、大文字と小文字を区別しない照合を使用して、他のデータウェアハウスシステムからの移行を容易にしています。大文字と小文字を区別しない照合のネイティブサポートにより、Amazon Redshift は、分散キー、ソートキー、範囲が制限されたスキャンなど、重要な調整法や最適化の方法を引き継ぐことができます。

COLLATE 句により、データベース内のすべての CHAR および VARCHAR 列に対する、デフォルトの照合手段を指定します。CASE_INSENSITIVE が指定されている場合、すべての CHAR または VARCHAR 列では、大文字と小文字を区別しない照合が使用されます。照合の詳細については、「照合順序」を参照してください。

大文字と小文字を区別しない列に挿入または取り込んだデータは、元の大文字と小文字を維持します。ただし、ソートやグループ化など、すべての比較ベースの文字列操作は、大文字と小文字を区別しません。LIKE 述語、および正規表現関数などのパターンマッチング操作でも、大文字と小文字を区別しません。

次の SQL オペレーションでは、適切な照合セマンティクスをサポートします。

- 比較演算子:=、<>、<、<=、>、>=
- LIKE 演算子
- ORDER BY 句
- GROUP BY 句
- MIN、MAX、LISTAGG など、文字列比較を使用する集計関数
- PARTITION BY 句や ORDER BY 句などのウィンドウ関数
- スカラー関数: greatest() 、least()、STRPOS()、REGEXP_COUNT()、REGEXP_REPLACE()、REGEXP_INSTR()、REG
- Distinct 句
- UNION、INTERSECT、および EXCEPT
- IN LIST

Amazon Redshift Spectrum および Aurora PostgreSQL のフェデレーティッドクエリを含む外部クエリの場合、VARCHAR または CHAR 列の照合は、現行のデータベースレベルの照合と同じです。

次の例では、Amazon Redshift Spectrum テーブルをクエリしています。

```
SELECT ci_varchar FROM spectrum.test_collation
WHERE ci_varchar = 'AMAZON';

ci_varchar
----------
amazon
Amazon
AMAZON
AmaZon
(4 rows)
```

データベースの照合を使用してテーブルを作成する方法については、「CREATE TABLE」を参照してください。

COLLATE 関数の詳細については、「COLLATE 関数」を参照してください。

データベース照合の制限

Amazon Redshift でデータベース照合を使用する場合、以下の制限事項があります。

- PG カタログテーブルや Amazon Redshift システムテーブルなど、すべてのシステムテーブルまたはビューでは、大文字と小文字が区別されます。
- コンシューマデータベースとプロデューサデータベースに異なるデータベースレベルの照合順序がある場合、Amazon Redshift はデータベース間クエリとクラスター間クエリをサポートしません。
- Amazon Redshift は、リーダーノードのみのクエリで大文字と小文字を区別しない照合をサポートしていません。

  以下に、サポートされていない大文字と小文字を区別しないクエリと、Amazon Redshift から送信されるエラーの例を示します。

```
SELECT collate(usename, 'case_insensitive') FROM pg_user;
ERROR:  Case insensitive collation is not supported in leader node only query.
```

- Amazon Redshift は、比較、関数、結合、設定オペレーションなど、大文字と小文字を区別する列と区別しない列間でのやり取りはサポートしていません。

  次に、大文字と小文字を区別する列と区別しない列の間で、やり取りした際に発生するエラーの例を示します。

```
CREATE TABLE test
  (ci_col varchar(10) COLLATE case_insensitive,
   cs_col varchar(10) COLLATE case_sensitive,
   cint int,
   cbigint bigint);
```

```
SELECT ci_col = cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT concat(ci_col, cs_col) FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT ci_col FROM test UNION SELECT cs_col FROM test;
ERROR:  Query with different collations is not supported yet.
```

```
SELECT * FROM test a, test b WHERE a.ci_col = b.cs_col;
ERROR:  Query with different collations is not supported yet.
```

```
Select Coalesce(ci_col, cs_col) from test;
ERROR:  Query with different collations is not supported yet.
```

```
Select case when cint > 0 then ci_col else cs_col end from test;
ERROR:  Query with different collations is not supported yet.
```

- Amazon Redshift は SUPER データ型の照合をサポートしていません。大文字と小文字を区別しないデータベースでの SUPER 列の作成、および SUPER 列と大文字文字を区別しない列間のやり取りはサポートされていません。

  次の例では、大文字と小文字を区別しないデータベースに、データ型に SUPER を使用するテーブルを作成しています。

  ```
  CREATE TABLE super_table (a super);
  ERROR: SUPER column is not supported in case insensitive database.
  ```

  次の例では、SUPER データと比較しながら、大文字と小文字を区別しない文字列を使用するデータをクエリしています。

  ```
  CREATE TABLE test_super_collation
  (s super, c varchar(10) COLLATE case_insensitive, i int);
  ```

  ```
  SELECT s = c FROM test_super_collation;
  ERROR:  Coercing from case insensitive string to SUPER is not supported.
  ```

これらのクエリを機能させるには、COLLATE 関数を使用して、ある列に対する照合を、別の列と一致するように変換します。詳細については、「COLLATE 関数」を参照してください。

例

データベースを作成する

次の例では、TICKIT という名前のデータベースを作成し、その所有権を DWUSER というユーザーに与えます。

```
create database tickit
with owner dwuser;
```

データベースに関する詳細を表示するために、PG_DATABASE_INFO カタログテーブルに対しクエリを実行しています。

```
select datname, datdba, datconnlimit
from pg_database_info
where datdba > 1;

 datname     | datdba | datconnlimit
-------------+--------+-------------
 admin       |    100 | UNLIMITED
 reports     |    100 | 100
 tickit      |    100 | 100
```

次の例では、スナップショット分離レベルを使用して **sampledb** という名前のデータベースを作成します。

```
CREATE DATABASE sampledb ISOLATION LEVEL SNAPSHOT;
```

次の例では、データ共有 salesshare からデータベース sales_db を作成しています。

```
CREATE DATABASE sales_db FROM DATASHARE salesshare OF NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

データベース照合の例

大文字と小文字を区別しないデータベースの作成

次の例では、データベース `sampledb` 、およびテーブル `T1` を作成し、そのテーブル `T1` にデータを挿入します。

```
create database sampledb collate case_insensitive;
```

SQL クライアントを使用して、先ほど作成した新しいデータベースに接続します。Amazon Redshift クエリエディタ v2 を使用している場合は、エディタで `sampledb` を選択します。RSQL を使用している場合は、次のようなコマンドを使用します。

```
\connect sampledb;
```

```
CREATE TABLE T1 (
  col1 Varchar(20) distkey sortkey
);
```

```
INSERT INTO T1 VALUES ('bob'), ('john'), ('Mary'), ('JOHN'), ('Bob');
```

その後、クエリが John を含む結果を検索します。

```
SELECT * FROM T1 WHERE col1 = 'John';

 col1
 ------
 john
 JOHN
(2 row)
```

大文字と小文字を区別しない規則での順序付け

次の例は、テーブル T1 での、大文字と小文字を区別しない順序付けを示しています。大文字と小文字を区別しない列では、Bob と bob あるいは John と john は同一に扱われるので、その順序は確定しません。

```
SELECT * FROM T1 ORDER BY 1;

 col1
 ------
 bob
 Bob
 JOHN
 john
 Mary
(5 rows)
```

同様に次の例では、GROUP BY 句に対する、大文字と小文字を区別しない順序付けを示しています。Bob と Bob は等し認識されるので、同じグループに属します。どちらが結果に表示されるかは非決定的です。

```
SELECT col1, count(*) FROM T1 GROUP BY 1;
```

```
 col1 | count
 -----+------
 Mary |  1
 bob  |  2
 JOHN |  2
(3 rows)
```

大文字と小文字を区別しない列でウィンドウ関数を使用してクエリを実行する

次の例では、大文字と小文字を区別しない列に対し、ウィンドウ関数によるクエリを実行します。

```
SELECT col1, rank() over (ORDER BY col1) FROM T1;

 col1 | rank
 -----+------
 bob  |   1
 Bob  |   1
 john |   3
 JOHN |   3
 Mary |   5
(5 rows)
```

DISTINCT キーワードを使用したクエリ

次の例に、DISTINCT キーワードを使用しながら、T1テーブルに対しクエリを実行します。

```
SELECT DISTINCT col1 FROM T1;

 col1
 ------
 bob
 Mary
 john
(3 rows)
```

UNION 句を使用したクエリ

次の例に、テーブル T1 および T2 に対し UNIONN 句を使用した場合の結果を示します。

```
CREATE TABLE T2 AS SELECT * FROM T1;
```

```
SELECT col1 FROM T1 UNION SELECT col1 FROM T2;

 col1
------
 john
 bob
 Mary
(3 rows)
```

CREATE DATASHARE

現在のデータベースに新しいデータ共有を作成します。このデータ共有の所有者は CREATE DATASHARE コマンドの発行者です。

Amazon Redshift は、各データ共有を単一の Amazon Redshift データベースに関連付けます。関連付けられたデータベースからデータ共有にのみオブジェクトを追加できます。同じ Amazon Redshift データベースに複数のデータ共有を作成できます。

データ共有の詳細については、Amazon Redshift でのデータの共有を参照してください。

データ共有に関する情報を表示するには、SHOW DATASHARES を使用します。

必要な権限

以下に、CREATE DATASHARE に必要な権限を示します。

- スーパーユーザー
- CREATE DATASHARE の権限を持つユーザー
- データベースの所有者

構文

```
CREATE DATASHARE datashare_name
[[SET] PUBLICACCESSIBLE [=] TRUE | FALSE ];
```

パラメータ

datashare_name

データ共有の名前。データ共有名は、クラスター名前空間内で一意である必要があります。

[[SET] PUBLICACCESSIBLE]

公開でアクセス可能なクラスターとデータ共有を共有できるかどうかを指定する句。

SET PUBLICACCESSIBLE のデフォルト値は FALSE です。

使用に関する注意事項

デフォルトでは、データ共有の所有者は共有のみを所有し、共有内のオブジェクトは所有しません。

スーパーユーザーとデータベース所有者のみが CREATE DATASHARE を使用して、ALTER 権限を他のユーザーまたはグループに委任できます。

例

次の例では、データ共有 salesshare を作成します。

```
CREATE DATASHARE salesshare;
```

次の例では、AWS Data Exchangeが管理するデータ共有 demoshare を作成します。

```
CREATE DATASHARE demoshare SET PUBLICACCESSIBLE TRUE, MANAGEDBY ADX;
```

CREATE EXTERNAL FUNCTION

Amazon Redshift 用の AWS Lambda に基づいてスカラーユーザー定義関数 (UDF) を作成します。Lambda のユーザー定義関数の詳細については、スカラー Lambda UDFを参照してください。

必要な権限

以下に、CREATE EXTERNAL FUNCTION に必要な権限を示します。

- スーパーユーザー
- CREATE [もしくは REPLACE] EXTERNAL FUNCTION の権限を持つユーザー

構文

```
CREATE [ OR REPLACE ] EXTERNAL FUNCTION external_fn_name ( [data_type] [, ...] )
RETURNS data_type
```

```
{ VOLATILE | STABLE }
LAMBDA 'lambda_fn_name'
IAM_ROLE { default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>'
RETRY_TIMEOUT milliseconds
MAX_BATCH_ROWS count
MAX_BATCH_SIZE size [ KB | MB ];
```

パラメータ

OR REPLACE

その関数の名前、入力引数のデータ型、あるいは署名が既存の関数と同じである場合に既存の関数を置き換えることを指定する句。同一のデータタイプセットを定義する新しい関数によってのみ既存の関数を置き換えることができます。関数の置き換えは、スーパーユーザーのみが行うことができます。

すでに存在するの関数と同じ名前と入力引数のデータタイプで、異なる署名の関数を定義する場合は、新しい関数を作成することになります。つまり、関数名はオーバーロードされます。詳細については、「関数名の多重定義」を参照してください。

external_fn_name

外部関数の名前。スキーマ名を指定すると (myschema.myfunction など)、指定したスキーマを使用して関数が作成されます。指定しない場合、現在のスキーマに関数が作成されます。有効な名前の詳細については、「名前と識別子」を参照してください。

すべての UDF 名の前に f_ を付けることをお勧めします。Amazon Redshift は、UDF 名の f_ プレフィックスを予約します。f_ プレフィックスを使用することで、UDF 名が現在または将来の Amazon Redshift の組み込み SQL 関数名と競合しないようにすることができます。詳細については、「UDF 名の競合の回避」を参照してください。

data_type

入力引数のデータタイプ。詳細については、「データ型」を参照してください。

RETURNS data_type

関数によって返される値のデータ型。RETURNS データ型には、Amazon Redshift のすべての標準データ型を使用できます。詳細については、「Python UDF データ型」を参照してください。

VOLATILE | STABLE

関数の変動率についてのクエリオプティマイザを報告します。

関数の最適な変動率の分類を厳正に設定してラベルを付けることで、最高の最適化が得られます。厳正度の順に、低度の厳正度から変動率を分類すると、以下のようになります。

- VOLATILE
- STABLE

VOLATILE

同じ引数が入っている場合、単一のステートメントに含まれる行であっても、関数は連続の呼び出しに異なる結果を返すことがあります。クエリオプティマイザは、変動的な関数の動作について想定することはできません。変動的な関数を使用するクエリは、入力ごとに関数を再評価する必要があります。

STABLE

引数が同じである場合、単一のステートメント内で処理される連続的な呼び出しに対して関数が同じ結果を返すことが保証されます。異なるステートメントから呼び出された場合、関数が異なる結果を返すことがあります。このカテゴリにより、オプティマイザは 1 つのステートメント内で関数が呼び出される回数を減らすことができます。

選択した厳密さが関数に対して有効でない場合、オプティマイザは、この厳密さに基づく一部の呼び出しをスキップするリスクがあることに注意してください。そのため、結果セットが不正確になる場合があります。

IMMUTABLE 句は、現在 Lambda UDF ではサポートされていません。

LAMBDA 'lambda_fn_name'

Amazon Redshift が呼び出す関数の名前。

AWS Lambda 関数を作成する手順については、AWS Lambdaデベロッパーガイドの「コンソールで Lambda 関数を作成する」を参照してください。

Lambda 関数に必要なアクセス許可の詳細については、AWS Lambdaデベロッパーガイドの「AWS Lambda アクセス許可」を参照してください。

IAM_ROLE { default | ‘arn:aws:iam::*<AWS #####-id>*:role/*<role-name>*’

デフォルトキーワードを使用して、CREATE EXTERNAL FUNCTION コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。CREATE EXTERNAL FUNCTION コマンドは、この IAM ロールを介して Lambda 関数を呼

び出すことが許可されています。クラスターに Lambda 関数を呼び出す権限を持つ既存の IAM ロールがある場合は、ロールの ARN に置き換えることができます。詳細については、「Lambda UDF の認可パラメータの設定」を参照してください。

以下に、IAM_ROLE パラメータの構文を示します。

```
IAM_ROLE 'arn:aws:iam::aws-account-id:role/role-name'
```

RETRY_TIMEOUT milliseconds

再試行バックオフの遅延に対して Amazon Redshift が使用する合計時間 (ミリ秒)。

失敗したクエリをすぐに再試行する代わりに、Amazon Redshift はバックオフを実行し、再試行の間に一定の時間待機します。その後、Amazon Redshift は、すべての遅延の合計が指定した RETRY_TIMEOUT 値以上になるまで、失敗したクエリを再実行するためのリクエストを再試行します。デフォルト値は 20,000 ミリ秒です。

Lambda 関数が呼び出されると、Amazon Redshift は、`TooManyRequestsException`、`EC2ThrottledException`、`ServiceException` などのエラーを受け取ったクエリを再試行します。

RETRY_TIMEOUT パラメータを 0 ミリ秒に設定して、Lambda UDF の再試行を防ぐことができます。

MAX_BATCH_ROWS count

Amazon Redshift が 1 回の Lambda 呼び出しに対して 1 回のバッチリクエストで送信する最大行数。

このパラメータの最小値は 1 です。最大値は INT_MAX または 2,147,483,647 です。

このパラメータはオプションです。デフォルト値は INT _MAX または 2,147,483,647 です。

MAX_BATCH_SIZE size [KB | MB]

Amazon Redshift が 1 回の Lambda 呼び出しに対して 1 回のバッチリクエストで送信するデータペイロードの最大サイズ。

このパラメータの最小値は 1 KB です。最大値は 5 MB です。

このパラメータのデフォルト値は 5 MB です。

KB と MB はオプションです。測定単位を設定しない場合、Amazon Redshift はデフォルトで KB を使用します。

使用に関する注意事項

Lambda UDF を作成するときは、次の点を考慮してください。

- 入力引数に対する Lambda 関数呼び出しの順序は、固定も保証もされていません。クラスター設定によっては、クエリを実行するインスタンス間で順序が異なる場合があります。
- 関数が入力引数ごとに 1 回だけ適用されるという保証はありません。Amazon Redshift と AWS Lambda とのやり取りにより、同じ入力に対して呼び出しが繰り返される可能性があります。

例

以下は、スカラー Lambda ユーザー定義関数 (UDF) の使用例を示します。

Node.js Lambda 関数を使用したスカラー Lambda UDF の例

次の例では、入力引数として 2 つの整数を受け取る `exfunc_sum` という外部関数を作成します。この関数は、合計を整数出力として返します。呼び出される Lambda 関数の名前は `lambda_sum` です。この Lambda 関数に使用される言語は Node.js 12.x です。IAM ロールを必ず指定してください。この例では、IAM ロールとして `'arn:aws:iam::123456789012:user/johndoe'` を使用しています。

```
CREATE EXTERNAL FUNCTION exfunc_sum(INT,INT)
RETURNS INT
VOLATILE
LAMBDA 'lambda_sum'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 関数はリクエストペイロードを受け取り、各行を繰り返して処理します。単一の行ですべての値が加算されて、その行の合計が計算され、応答配列に保存されます。結果配列の行数は、リクエストペイロードで受信した行数と似ています。

JSON 応答ペイロードは、外部関数によって認識されるために、['results' (結果)] フィールドに結果データを持っている必要があります。Lambda 関数に送信されるリクエストの引数フィールドには、データペイロードが含まれます。バッチリクエストの場合、データペイロードに複数の行が存在する可能性があります。次の Lambda 関数は、リクエストデータペイロードのすべての行を繰り返し処理します。また、単一の行内のすべての値を個別に繰り返します。

```
exports.handler = async (event) => {
    // The 'arguments' field in the request sent to the Lambda function contains the
 data payload.
```

```
    var t1 = event['arguments'];

    // 'len(t1)' represents the number of rows in the request payload.
    // The number of results in the response payload should be the same as the number
 of rows received.
    const resp = new Array(t1.length);

    // Iterating over all the rows in the request payload.
    for (const [i, x] of t1.entries())
    {
        var sum = 0;
        // Iterating over all the values in a single row.
        for (const y of x) {
            sum = sum + y;
        }
        resp[i] = sum;
    }
    // The 'results' field should contain the results of the lambda call.
    const response = {
        results: resp
    };
    return JSON.stringify(response);
};
```

次の例では、リテラル値を使用して外部関数を呼び出します。

```
select exfunc_sum(1,2);
exfunc_sum
------------
 3
(1 row)
```

次の例では、整数データ型の 2 つの列 c1 と c2 を持つ t_sum というテーブルを作成し、2 行のデータを挿入します。次に、このテーブルの列名を渡すことにより、外部の関数が呼び出されます。2 つのテーブル行は、リクエストペイロードのバッチリクエストで単一の Lambda 呼び出しとして送信されます。

```
CREATE TABLE t_sum(c1 int, c2 int);
INSERT INTO t_sum VALUES (4,5), (6,7);
SELECT exfunc_sum(c1,c2) FROM t_sum;
 exfunc_sum
---------------
```

```
 9
 13
(2 rows)
```

RETRY_TIMEOUT 属性を使用したスカラー Lambda UDF の例

次のセクションでは、Lambda UDF で RETRY_TIMEOUT 属性を使用する方法の例を見つけることができます。

AWS Lambda 関数には、関数ごとに設定する同時実行数についての制限があります。同時実行数の制限に関する詳細については、「AWS Lambda デベロッパーガイド」の「Lambda の予約済み同時実行数の管理」と、AWS コンピューティングブログの記事「AWS Lambda 関数の同時実行数の管理」を参照してください。

Lambda UDF によって処理されるリクエストの数が同時実行制限を超えると、新しいリクエストは TooManyRequestsException エラーを受け取ります。Lambda UDF は、Lambda 関数に送信されるリクエスト間のすべての遅延の合計が、設定した RETRY_TIMEOUT 値以上になるまで、このエラーを再試行します。デフォルトの RETRY_TIMEOUT 値は 20,000 ミリ秒です。

次の例では、exfunc_sleep_3 という名前の Lambda 関数を使用します。この関数は、リクエストペイロードを取り込み、各行を繰り返し処理し、入力を大文字に変換します。その後、3 秒間スリープし、結果を返します。この Lambda 関数に使用される言語は Python 3.8 です。

結果配列の行数は、リクエストペイロードで受信した行数と似ています。JSON 応答ペイロードは、外部関数によって認識されるために、results フィールドに結果データを持っている必要があります。Lambda 関数に送信されるリクエストの arguments フィールドには、データペイロードが含まれています。バッチリクエストの場合、データペイロードに複数の行が表示されることがあります。

この関数の同時実行制限は、RETRY_TIMEOUT 属性の使用法を示すために、予約済み同時実行で特に 1 に設定されています。属性が 1 に設定されている場合、Lambda 関数は一度に 1 つのリクエストしか処理できません。

```
import json
import time
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)
```

```
    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        # Iterating over all the values in a single row.
        for j, y in enumerate(x):
            resp[i] = y.upper()

    time.sleep(3)
    ret = dict()
    ret['results'] = resp
    ret_json = json.dumps(ret)
    return ret_json
```

次に、RETRY_TIMEOUT 属性の追加例を 2 つ示します。それぞれ単一の Lambda UDF を呼び出します。Lambda UDF を呼び出す間、各例は同じ SQL クエリを実行して、2 つの同時実行データベースセッションから同時に Lambda UDF を呼び出します。Lambda UDF を呼び出す最初のクエリが UDF によって処理されている場合、2 番目のクエリは `TooManyRequestsException` エラーを受け取ります。この結果は、UDF で予約された同時実行を特別に 1 に設定したために発生します。Lambda 関数の予約済み同時実行を設定する方法については、予約済み同時実行数の設定を参照してください。

次の最初の例では、Lambda UDF の RETRY_TIMEOUT 属性を 0 ミリ秒に設定します。Lambda リクエストが Lambda 関数から例外を受け取った場合、Amazon Redshift は再試行を行いません。この結果は、RETRY_TIMEOUT 属性が 0 に設定されているために発生します。

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 0;
```

RETRY_TIMEOUT を 0 に設定すると、別々のデータベースセッションから次の 2 つのクエリを実行して、異なる結果を確認できます。

Lambda UDF を使用する最初の SQL クエリが正常に実行されます。

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

別のデータベースセッションから同時に実行される 2 番目のクエリは、`TooManyRequestsException` エラーを受け取ります。

```
select exfunc_upper('Varchar');
ERROR:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
DETAIL:
-----------------------------------------------
error:  Rate Exceeded.; Exception: TooManyRequestsException; ShouldRetry: 1
code:      32103
context:query:     0
location:  exfunc_client.cpp:102
process:   padbmaster [pid=26384]
-----------------------------------------------
```

次の 2 番目の例では、Lambda UDF の RETRY_TIMEOUT 属性を 3,000 ミリ秒に設定します。2 番目のクエリが同時に実行された場合でも、Lambda UDF は遅延の合計が 3,000 ミリ秒になるまで再試行します。したがって、両方のクエリが正常に実行されます。

```
CREATE OR REPLACE EXTERNAL FUNCTION exfunc_upper(varchar)
RETURNS varchar
VOLATILE
LAMBDA 'exfunc_sleep_3'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test'
RETRY_TIMEOUT 3000;
```

RETRY_TIMEOUT を 3,000 ミリ秒に設定すると、別々のデータベースセッションから次の 2 つのクエリを実行して、同じ結果を確認できます。

Lambda UDF を実行する最初の SQL クエリが正常に実行されます。

```
select exfunc_upper('Varchar');
 exfunc_upper
 --------------
 VARCHAR
(1 row)
```

2 番目のクエリは同時に実行され、Lambda UDF は遅延の合計が 3,000 ミリ秒になるまで再試行します。

```
select exfunc_upper('Varchar');
```

```
 exfunc_upper
--------------
 VARCHAR
(1 row)
```

Python Lambda 関数を使用したスカラー Lambda UDF の例

次の例では、`exfunc_multiplication` という名前の外部関数を作成します。この関数は、数値を乗算して整数を返します。この例では、Lambda 応答に success フィールドと `error_msg` フィールドが組み込まれています。乗算結果に整数オーバーフローがあり、`error_msg` メッセージが `Integer multiplication overflow` に設定されている場合、成功フィールドは false に設定されます。`exfunc_multiplication` 関数は、入力引数として 3 つの整数を取り、その合計を整数出力として返します。

呼び出される Lambda 関数の名前は `lambda_multiplication` です。この Lambda 関数に使用される言語は Python 3.8 です。IAM ロールを必ず指定してください。

```
CREATE EXTERNAL FUNCTION exfunc_multiplication(int, int, int)
RETURNS INT
VOLATILE
LAMBDA 'lambda_multiplication'
IAM_ROLE 'arn:aws:iam::123456789012:role/Redshift-Exfunc-Test';
```

Lambda 関数はリクエストペイロードを受け取り、各行を繰り返して処理します。1 つの行ですべての値が乗算され、その行の結果が計算されます。この結果は、応答リストに保存されます。この例では、デフォルトで true に設定されているブールの成功値を使用します。行の乗算結果に整数オーバーフローがある場合、成功値は false に設定されます。その後、反復ループが中断されます。

応答ペイロードの作成中に、成功値が false の場合、次の Lambda 関数がペイロードに `error_msg` フィールドを追加します。また、エラーメッセージを `Integer multiplication overflow` に設定します。成功値が true の場合、結果データが結果フィールドに追加されます。結果配列の行数は、存在する場合、リクエストペイロードで受信した行数と似ています。

Lambda 関数に送信されるリクエストの引数フィールドには、データペイロードが含まれます。バッチリクエストの場合、データペイロードに複数の行が存在する可能性があります。次の Lambda 関数は、リクエストデータペイロードのすべての行を繰り返し処理し、1 つの行内のすべての値を個別に繰り返し処理します。

```
import json
```

```
def lambda_handler(event, context):
    t1 = event['arguments']
    # 'len(t1)' represents the number of rows in the request payload.
    # The number of results in the response payload should be the same as the number of
 rows received.
    resp = [None]*len(t1)

    # By default success is set to 'True'.
    success = True
    # Iterating over all rows in the request payload.
    for i, x in enumerate(t1):
        mul = 1
        # Iterating over all the values in a single row.
        for j, y in enumerate(x):
            mul = mul*y

        # Check integer overflow.
        if (mul >= 9223372036854775807 or mul <= -9223372036854775808):
            success = False
            break
        else:
            resp[i] = mul
    ret = dict()
    ret['success'] = success
    if not success:
        ret['error_msg'] = "Integer multiplication overflow"
    else:
        ret['results'] = resp
    ret_json = json.dumps(ret)

    return ret_json
```

次の例では、リテラル値を使用して外部関数を呼び出します。

```
SELECT exfunc_multiplication(8, 9, 2);
  exfunc_multiplication
---------------------------
          144
(1 row)
```

次の例では、整数データ型の 3 つの列 c1、c2、および c3 を持つ t_multi という名前のテーブルを作成します。外部関数は、このテーブルの列名を渡すことによって呼び出されます。データは、エラーがどのように伝播されるかを示す整数オーバーフローを引き起こすような方法で挿入されます。

```
CREATE TABLE t_multi (c1 int, c2 int, c3 int);
INSERT INTO t_multi VALUES (2147483647, 2147483647, 4);
SELECT exfunc_multiplication(c1, c2, c3) FROM t_multi;
DETAIL:
  -----------------------------------------------
  error:  Integer multiplication overflow
  code:      32004context:
  context:
  query:     38
  location:  exfunc_data.cpp:276
  process:   query2_16_38 [pid=30494]
  -----------------------------------------------
```

CREATE EXTERNAL MODEL

トピック

CREATE EXTERNAL MODEL の前提条件

CREATE EXTERNAL MODEL ステートメントを使用する前に、Amazon Redshift ML を使用するためのクラスターの設定 の前提条件を満たしてください。前提条件の概要は次のとおりです。

- AWS マネジメントコンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Amazon Redshift クラスターを作成します。
- クラスターの作成中に AWS Identity and Access Management (IAM) ポリシーをアタッチします。
- Amazon Redshift と Amazon Bedrock が、他のサービスとやり取りするロールを引き受けることを許可するには、IAM ロールに適切な信頼ポリシーを追加します。

IAM ロール、信頼ポリシー、およびその他の前提条件の詳細については、「Amazon Redshift ML を使用するためのクラスターの設定」を参照してください。

必要な権限

CREATE EXTERNAL MODEL に必要な権限を以下に示します。

- スーパーユーザー
- CREATE EXTERNAL MODEL の権限を持つユーザー
- CREATE EXTERNAL MODEL の権限を持つロール

コスト管理

Amazon Redshift 機械学習は既存のクラスターリソースを使用して予測モデルを作成するため、追加料金は発生しません。ただし、選択したモデルに基づいて Amazon Bedrock を使用する場合は AWS 料金が発生します。詳細については、「Amazon Redshift 機械学習を使用するためのコスト」を参照してください。

CREATE EXTERNAL MODEL 構文

以下は、CREATE EXTERNAL MODEL ステートメントの完全な構文です。

```
CREATE EXTERNAL MODEL model_name
FUNCTION function_name
IAM_ROLE {default/'arn:aws:iam::<account-id>:role/<role-name>'}
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID model_id
   [, PROMPT 'prompt prefix']
   [, SUFFIX 'prompt suffix']
   [, REQUEST_TYPE {RAW|UNIFIED}]
   [, RESPONSE_TYPE {VARCHAR|SUPER}]
);
```

`CREATE EXTERNAL MODEL` コマンドは、コンテンツの生成に使用する推論関数を作成します。

以下は、`RAW` の `REQUEST_TYPE` を使用して `CREATE EXTERNAL MODEL` が作成する推論関数の構文です。

```
SELECT inference_function_name(request_super)
[FROM table];
```

以下は、UNIFIED の REQUEST_TYPE を使用して CREATE EXTERNAL MODEL が作成する推論関数の構文です。

```
SELECT inference_function_name(input_text, [, inference_config [,
 additional_model_request_fields]])
[FROM table];
```

推論関数の使用方法については、「Amazon Redshift ML と Amazon Bedrock の統合用の外部モデルの使用」を参照してください。

CREATE EXTERNAL MODEL のパラメータと設定

このセクションでは、 CREATE EXTERNAL MODEL コマンドのパラメータと設定について説明します。

トピック

- CREATE EXTERNAL MODEL のパラメータ
- CREATE EXTERNAL MODEL の設定

CREATE EXTERNAL MODEL のパラメータ

model_name

外部モデルの名前。スキーマ内のモデル名は一意でなければなりません。

FUNCTION function_name (data_type [,...])

CREATE EXTERNAL MODEL が作成する推論関数の名前。推論関数を使用して Amazon Bedrock にリクエストを送信し、ML 生成テキストを取得できます。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

Amazon Redshift が Amazon Bedrock へのアクセスに使用する IAM ロール。IAM ロールに関する詳細は、「Amazon Redshift ML と Amazon Bedrock の統合のための IAM ロールの作成または更新」を参照してください。

MODEL_TYPE BEDROCK

モデルタイプを指定します。唯一の有効な値は BEDROCK です。

SETTINGS (MODEL_ID model_id [,...])

外部モデルの設定を指定します。詳細については、以下のセクションを参照してください。

CREATE EXTERNAL MODEL の設定

MODEL_ID model_id

外部モデルの識別子 (例: `anthropic.claude-v2`)。Amazon Bedrock モデルの ID については、「Amazon Bedrock model IDs」を参照してください。

PROMPT 'prompt prefix'

Amazon Redshift がすべての推論リクエストの先頭に追加する静的プロンプトを指定します。`UNIFIED` の `REQUEST_TYPE` でのみサポートされます。

SUFFIX 'prompt suffix'

Amazon Redshift がすべての推論リクエストの末尾に追加する静的プロンプトを指定します。`UNIFIED` の `REQUEST_TYPE` でのみサポートされます。

REQUEST_TYPE { RAW | UNIFIED }

Amazon Bedrock に送信されるリクエストの形式を指定します。有効な値には次のようなものがあります。

- RAW: 推論関数は入力を単一の SUPER 値として受け取り、常に SUPER 値を返します。SUPER 値の形式は、選択した Amazon Bedrock モデルに固有です。SUPER は、複数のアルゴリズムを組み合わせて、より優れた単一の予測を生成する予測モデルです。
- UNIFIED: 推論関数は統合 API を使用します。すべてのモデルに Amazon Bedrock との統一された一貫したインターフェイスがあります。これは、メッセージをサポートするすべてのモデルで機能します。この値はデフォルト値です。

  詳細については、「Amazon Bedrock API ドキュメント」の「Converse API documentation」を参照してください。

RESPONSE_TYPE { VARCHAR | SUPER }

レスポンスの形式を指定します。`REQUEST_TYPE` が `RAW` の場合、`RESPONSE_TYPE` は必須であり、有効な値は `SUPER` のみです。他のすべての `REQUEST TYPE` 値では、デフォルト値は `VARCHAR` で、`RESPONSE_TYPE` はオプションです。有効な値には次のようなものがあります。

- VARCHAR: Amazon Redshift は、モデルによって生成されたテキストレスポンスのみを返します。
- SUPER: Amazon Redshift は、モデルによって生成されたレスポンス JSON 全体を SUPER として返します。これには、テキストレスポンス、停止理由、モデル入出力トークンの使用状況

などの情報が含まれます。SUPER は、複数のアルゴリズムを組み合わせて、より優れた単一の予測を生成する予測モデルです。

CREATE EXTERNAL MODEL 推論関数パラメータ

このセクションでは、`CREATE EXTERNAL MODEL` コマンドが作成する推論関数の有効なパラメータについて説明します。

`RAW` の **`REQUEST_TYPE`** 用の CREATE EXTERNAL MODEL 推論関数パラメータ

`RAW` の `REQUEST_TYPE` で作成された推論関数には 1 つの SUPER 入力引数があり、常に SUPER データ型を返します。入力 SUPER の構文は、Amazon Bedrock から選択された特定のモデルのリクエストの構文に従います。

`UNIFIED` の **`REQUEST_TYPE`** 用の CREATE EXTERNAL MODEL 推論関数パラメータ

input_text

Amazon Redshift が Amazon Bedrock に送信するテキスト。

inference_config

Amazon Redshift が Amazon Bedrock に送信するオプションのパラメータを含む SUPER 値。これには以下が含まれます。

- maxTokens
- stopSequences
- temperature
- topP

これらのパラメータはすべてオプションであり、すべて大文字と小文字が区別されます。これらのパラメータの詳細については、「Amazon Bedrock API リファレンス」の「InferenceConfiguration」を参照してください。

CREATE EXTERNAL SCHEMA

現在のデータベースに新しい外部スキーマを作成します。この外部スキーマを使用して、Amazon RDS for PostgreSQL または Amazon Aurora PostgreSQL 互換エディションデータベースに接続できます。また、AWS Glueや Athena などの外部データカタログ内のデータベース、あるいは Amazon

EMR などの Apache Hive メタストアにあるデータベースを参照する、外部スキーマを作成することもできます。

このスキーマの所有者は CREATE EXTERNAL SCHEMA コマンドの発行者です。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用して所有者を変更します。スキーマに他のユーザーやグループへのアクセス権を付与するには、GRANTコマンドを使用します。

外部テーブルのアクセス権限に対して、GRANT または REVOKE コマンドを使用することはできません。代わりに、外部スキーマに対するアクセス権限の付与または取り消しを実行します。

> **Note**
>
> 現在、Amazon Athena データカタログに Redshift Spectrum の外部テーブルがある場合は、AWS Glue Data Catalogに Athena データカタログを移行することが可能です。Redshift Spectrum で AWS Glue データカタログを使用するには、AWS Identity and Access Management(IAM) ポリシーの変更が必要になる場合があります。詳細については、Athena ユーザーガイドの「AWS Glue データカタログへのアップグレード」を参照してください。

外部スキーマの詳細を表示するには、SVV_EXTERNAL_SCHEMASシステムビューにクエリを実行します。

構文

次の構文は、外部データカタログを使用してデータを参照するために使用する CREATE EXTERNAL SCHEMA コマンドを示しています。詳細については、「Amazon Redshift Spectrum」を参照してください。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM [ [ DATA CATALOG ] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK |
 REDSHIFT ]
[ DATABASE 'database_name' ]
[ SCHEMA 'schema_name' ]
[ REGION 'aws-region' ]
[ IAM_ROLE [ default | 'SESSION' | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' ] ]
[ AUTHENTICATION [ none | iam | mtls] ]
[ AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret- arn' ]
[ URI ['hive_metastore_uri' [ PORT port_number ] | 'hostname' [ PORT port_number ] |
 'msk bootstrap URL'] ]
[ CLUSTER_ARN 'arn:aws:kafka:<region>:<AWS #####-id>:cluster/msk/<cluster uuid>' ]
[ CATALOG_ROLE [ 'SESSION' | 'catalog-role-arn-string' ] ]
```

```
[ CREATE EXTERNAL DATABASE IF NOT EXISTS ]
[ CATALOG_ID 'Amazon Web Services account ID containing Glue or Lake Formation
 database' ]
```

次の構文は、RDS POSTGRES または Aurora PostgreSQL への横串検索を使用してデータを参照するために使用する CREATE EXTERNAL SCHEMA コマンドを示しています。作成した外部スキーマで、Kinesis Data Streams などのストリーミングソースを参照することもできます。詳細については、「Amazon Redshift での横串検索を使用したデータのクエリの実行」を参照してください。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM POSTGRES
DATABASE 'federated_database_name' [SCHEMA 'schema_name']
URI 'hostname' [ PORT port_number ]
IAM_ROLE [ default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' ]
SECRET_ARN 'ssm-secret-arn'
```

次の構文は、RDS MySQL または Aurora MySQL への横串検索を使用してデータを参照するために使用する CREATE EXTERNAL SCHEMA コマンドを示しています。詳細については、「Amazon Redshift での横串検索を使用したデータのクエリの実行」を参照してください。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] local_schema_name
FROM MYSQL
DATABASE 'federated_database_name'
URI 'hostname' [ PORT port_number ]
IAM_ROLE [ default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' ]
SECRET_ARN 'ssm-secret-arn'
```

次に、Kinesis のストリームデータの参照に使用する、CREATE EXTERNAL SCHEMA コマンドでの構文記述を示します。詳細については、「マテリアライズドビューへのストリーミング取り込み」を参照してください。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM KINESIS
IAM_ROLE [ default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' ]
```

次の構文は、Amazon Managed Streaming for Apache Kafka クラスターとそのトピックを参照して取り込むために使用する CREATE EXTERNAL SCHEMA コマンドについて説明します。接続するには、ブローカー URI を指定します。詳細については、「マテリアライズドビューへのストリーミング取り込み」を参照してください。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] schema_name
FROM MSK
[ IAM_ROLE [ default | 'arn:aws:iam::<AWS #####-id>:role/<role-name>' ] ]
URI 'msk bootstrap URL'
AUTHENTICATION [ none | iam | mtls ]
[ AUTHENTICATION_ARN 'acm-certificate-arn' | SECRET_ARN 'ssm-secret- arn' ];
```

次の構文は、クロスデータベースクエリを使用してデータを参照するために使用する CREATE EXTERNAL SCHEMA コマンドを示しています。

```
CREATE EXTERNAL SCHEMA local_schema_name
FROM  REDSHIFT
DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'
```

パラメータ

IF NOT EXISTS

指定されたスキーマが既に存在する場合、コマンドはエラーで終了するのではなく、何も変更しないで、スキーマが存在するというメッセージを返すことを示す句。この句は、CREATE EXTERNAL SCHEMA で既存のスキーマを作成しようとしてもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

local_schema_name

新しい外部スキーマの名前。有効な名前の詳細については、「名前と識別子」を参照してください。

FROM [DATA CATALOG] | HIVE METASTORE | POSTGRES | MYSQL | KINESIS | MSK | REDSHIFT

外部データベースの場所を示すキーワード。

DATA CATALOG は、外部データベースが Athena データカタログまたは AWS Glue Data Catalog 内で定義されていることを示します。

外部データベースが別の AWS リージョンにある外部データカタログで定義されている場合は、REGION パラメータが必要です。DATA CATALOG はデフォルトです。

HIVE METASTORE は、外部データベースが Apache Hive メタストアで定義されていることを示します。HIVE METASTORE が指定されている場合は、URI が必要です。

POSTGRES は、外部データベースが RDS PostgreSQL または Aurora PostgreSQL で定義されていることを示します。

MYSQL は、外部データベースが RDS MySQL または Aurora MySQL で定義されていることを示します。

KINESIS は、データソースが Kinesis Data Streams からのストリームであることを示します。

MSK は、データソースが Amazon MSK でプロビジョニングされたクラスターまたはサーバーレスクラスターであることを示します。

FROM REDSHIFT

データベースが Amazon Redshift にあることを示すキーワード。

DATABASE 'redshift_database_name' SCHEMA 'redshift_schema_name'

Amazon Redshift データベースの名前。

redshift_schema_name は、Amazon Redshift のスキーマを示します。デフォルトの redshift_schema_name は `public` です。

DATABASE 'federated_database_name'

サポートされている PostgreSQL または MySQL データベースエンジンの外部データベースの名前を示すキーワード。

[SCHEMA 'schema_name']

schema_name は、サポートされている PostgreSQL データベースエンジンのスキーマを示します。デフォルトの schema_name は `public` です。

サポートされている MySQL データベースエンジンへの横串検索を設定する場合、SCHEMA を指定することはできません。

REGION 'aws-region'

外部データベースが Athena データカタログまたは AWS Glue Data Catalog 内で定義されている場合の、データベースが存在する AWS リージョン。このパラメータは、データベースが外部データカタログで定義されている場合に必要です。

URI ['hive_metastore_uri' [PORT port_number] | 'hostname' [PORT port_number] | 'msk bootstrap URL']

サポートされている PostgreSQL または MySQL データベースエンジンのホスト名 URI と port_number。hostname は、レプリカセットのヘッドノードです。エンドポイントは、Amazon

Redshift クラスターから到達可能 (ルーティング可能) である必要があります。PostgreSQL のデフォルトの port_number は 5432 です。MySQL のデフォルトの port_number は 3306 です。

> **Note**
>
> サポートされている PostgreSQL または MySQL データベースエンジンは、Amazon Redshift クラスターと同じ VPC 内にあり、Amazon Redshift と RDS の url-rsPostgreSQL または Aurora PostgreSQL をリンクするセキュリティグループが必要です。さらに、拡張 VPC ルーティングを使用して、クロス VPC ユースケースを設定できます。詳細については、「RedShift マネージド VPC エンドポイント」を参照してください。

Hive メタストア URI の指定

データベースが Hive メタストアにある場合は、URI を指定し、オプションでメタストアのポート番号を指定します。デフォルトのポート番号は 9083 です。

URI にはプロトコル仕様 ("http://") が含まれていません。有効な URI の例: `uri '172.10.10.10'`。

ストリーミング取り込み用のブローカー URI の指定

ブートストラップブローカー URI を含めると、Amazon MSK クラスターに接続してストリーミングデータを受信できます。詳細と例については、「Amazon Managed Streaming for Apache Kafka からのストリーミング取り込みを開始する」を参照してください。

IAM_ROLE [default | 'SESSION' | 'arn:aws:iam::*<AWS #####-id>*:role/*<role-name>*']

デフォルトキーワードを使用して、CREATE EXTERNAL SCHEMA コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。

フェデレーション ID を使用して Amazon Redshift クラスターに接続し、このコマンドを使用して作成された外部スキーマからテーブルにアクセスする場合は、`'SESSION'` を使用します。詳細については、フェデレーション ID の設定方法を説明している「フェデレーション ID を使用してローカルリソースへの Amazon Redshift アクセスおよび Amazon Redshift Spectrum 外部テーブルを管理する」を参照してください。ARN の代わりに `'SESSION'` を使用するこの設定は、`DATA CATALOG` を使用してスキーマを作成した場合にのみ使用できることに注意してください。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。少なくとも、IAM ロールには、Amazon S3 バケットで LIST オペレーションを実行してアクセスを受ける許可と、バケットに含まれる Amazon S3 オブジェクトで GET オペレーションを実行する許可が必要です。

以下に ARN が 1 つの場合の IAM_ROLE パラメータ文字列の構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

ロールを連鎖することで、クラスターは別のアカウントに属している可能性がある別の IAM ロールを引き受けることができます。最大 10 個までのロールを連鎖できます。ロールの連鎖の例については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

この IAM ロールに次のような IAM アクセス許可ポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

フェデレーションクエリで使用する IAM ロールを作成するステップについては、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。

> **Note**
>
> 連鎖したロールのリストには空白を含めないでください。

以下に連鎖された 3 つのロールの構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

SECRET_ARN 'ssm-secret-arn'

AWS Secrets Manager を使用して作成された、(サポート対象の) PostgreSQL または MySQL データベースエンジンシークレットの、Amazon リソースネーム (ARN)。シークレットの ARN を作成および取得する方法については、AWS Secrets Managerユーザーガイドの「シークレットを作成する」および「Retrieving the Secret Value Secret」を参照してください。

CATALOG_ROLE ['SESSION' | catalog-role-arn-string]

データカタログの認証と認可のためにフェデレーション ID を使用して Amazon Redshift クラスターに接続する場合は、`'SESSION'` を使用します。フェデレーション ID のステップを完了するための詳細については、「フェデレーション ID を使用してローカルリソースへの Amazon Redshift アクセスおよび Amazon Redshift Spectrum 外部テーブルを管理する」を参照してください。この `'SESSION'` ロールは、DATA CATALOG でスキーマを作成した場合にのみ使用できることに注意してください。

クラスターによってデータカタログの認証と認可に使用される IAM ロールの Amazon リソースネーム (ARN) を使用します。

CATALOG_ROLE は Amazon Redshift を指定せず、指定された IAM_ROLE を使用します。カタログロールには、AWS Glueあるいは Athena のデータカタログへのアクセス許可が必要です。詳細については、「Amazon Redshift Spectrum 用の IAM ポリシー」を参照してください。

以下に ARN が 1 つの場合の CATALOG_ROLE パラメータ文字列の構文を示します。

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role>'
```

ロールを連鎖することで、クラスターは別のアカウントに属している可能性がある別の IAM ロールを引き受けることができます。最大10 個までのロールを連鎖できます。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

> **Note**
>
> 連鎖したロールのリストは、空白を含むことができません。

以下に連鎖された 3 つのロールの構文を示します。

```
CATALOG_ROLE 'arn:aws:iam::<aws-account-id>:role/<catalog-role-1-
name>,arn:aws:iam::<aws-account-id>:role/<catalog-role-2-name>,arn:aws:iam::<aws-
account-id>:role/<catalog-role-3-name>'
```

CREATE EXTERNAL DATABASE IF NOT EXISTS

指定の外部データベースが存在しない場合に、DATABASE 引数で指定された名前で外部データベースを作成する句。指定の外部データベースが存在する場合、コマンドは変更を加えません。この場合、コマンドは、エラーで終了せず、外部データが存在することを示すメッセージを返します。

> **Note**
>
> CREATE EXTERNAL DATABASE IF NOT EXISTS を HIVE METASTORE と併用することはできません。
> AWS Lake Formation が有効になっているデータカタログで、CREATE EXTERNAL DATABASE IF NOT EXISTS を使用するには、データカタログに対し `CREATE_DATABASE` を実行する許可が必要です。

CATALOG_ID 'Glue または Lake Formation データベースに含まれるアマゾン ウェブ サービスのアカウント ID '

データカタログデータベースが保存されているアカウント ID。

`CATALOG_ID` は、次のいずれかを設定することで、データカタログの認証と認可にフェデレーション ID を使用して Amazon Redshift クラスターまたは Amazon Redshift Serverless に接続する予定の場合に限り指定できます。

- CATALOG_ROLE ～ 'SESSION'
- IAM_ROLE、'SESSION'、'CATALOG_ROLE' をデフォルトに設定する

フェデレーション ID のステップを完了するための詳細については、「フェデレーション ID を使用して、ローカルリソースと Amazon Redshift Spectrum の外部テーブルへの Amazon Redshift アクセスを管理する」を参照してください。

AUTHENTICATION

ストリーミング取り込み用に定義された認証タイプ。認証タイプによるストリーミング取り込みは、Amazon Managed Streaming for Apache Kafka と連携します。AUTHENTICATION タイプには以下のものがあります。

- none — 必要な認証がないことを指定します。これは、MSK での認証されていないアクセスに対応します。
- iam — IAM 認証を指定します。これを選択するときは、IAM ロールに IAM 認証のアクセス許可があることを確認します。外部スキーマの定義の詳細については、「Amazon Managed Streaming for Apache Kafka (Amazon MSK) からのストリーミング取り込みを開始する」を参照してください。
- mtls – クライアントとサーバー間の認証を行うことで、相互 Transport Layer Security が安全な通信を提供することを指定します。この場合、クライアントは Redshift で、サーバーは Amazon MSK です。mTLS によるストリーミング取り込みの設定の詳細については、「Amazon MSK からの Redshift ストリーミング取り込みにおける mTLS による認証」を参照してください。

AUTHENTICATION_ARN

Amazon Redshift が Amazon MSK による mtls 認証に使用する AWS Certificate Manager 証明書の ARN。ARN は、発行された証明書を選択する際に ACM コンソールで利用できます。

CLUSTER_ARN

ストリーミング取り込みの場合、CLUSTER_ARN は、ストリーミング元の Amazon Managed Streaming for Apache Kafka クラスターのクラスター識別子です。CLUSTER_ARN を使用する場合は、kafka:GetBootstrapBrokers アクセス許可を含む IAM ロールポリシーが必要です。このオプションは、下位互換性のために用意されています。現在、ブートストラップブローカー URI オプションを使用して Amazon Managed Streaming for Apache Kafka クラスターに接続することをお勧めします。詳細については、「ストリーミングの取り込み」を参照してください。

使用に関する注意事項

Athena データカタログを使用する場合の制限については、「AWS 全般のリファレンス」の「Athena の制限」を参照してください。

AWS Glue Data Catalog を使用する場合の制限については、「AWS 全般のリファレンス」の「AWS Glue の制限」を参照してください。

これらの制限は、Hive メタストアには適用されません。

スキーマの数は 1 つのデータベースにつき最大 9,900 個です。詳細については、Amazon Redshift 管理ガイドの「クォータと制限」を参照してください。

スキーマの登録を解除するには、DROP SCHEMAコマンドを使用します。

外部スキーマの詳細を表示するには、システムビューにクエリを実行します。

- SVV_EXTERNAL_SCHEMAS
- SVV_EXTERNAL_TABLES
- SVV_EXTERNAL_COLUMNS

例

次の例では、米国西部 (オレゴン) リージョンの `sampledb` という名前のデータカタログのデータベースを使用して外部スキーマを作成します。この例を Athena または AWS Glue データカタログで使用します。

```
create external schema spectrum_schema
from data catalog
database 'sampledb'
region 'us-west-2'
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

次の例は、外部スキーマを作成し、`spectrum_db`という名前で新しい外部データベースを作成します。

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
```

```
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole'
create external database if not exists;
```

次の例では、hive_dbという名前の Hive メタストアデータベースを使って外部スキーマを作成します。

```
create external schema hive_schema
from hive metastore
database 'hive_db'
uri '172.10.10.10' port 99
iam_role 'arn:aws:iam::123456789012:role/MySpectrumRole';
```

次の連鎖ロールの例では、Amazon S3 へのアクセスにロール mys3Role を使用し、データカタログへのアクセスには myAthenaRole を使用します。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

```
create external schema spectrum_schema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myRedshiftRole,arn:aws:iam::123456789012:role/
myS3Role'
catalog_role 'arn:aws:iam::123456789012:role/myAthenaRole'
create external database if not exists;
```

次の例では、Aurora PostgreSQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM POSTGRES
DATABASE 'my_aurora_db' SCHEMA 'my_aurora_schema'
URI 'endpoint to aurora hostname' PORT 5432
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

次の例では、外部スキーマを作成し、コンシューマークラスターにインポートされた sales_db を参照しています。

```
CREATE EXTERNAL SCHEMA sales_schema FROM REDSHIFT DATABASE 'sales_db' SCHEMA 'public';
```

次の例では、Aurora MySQL データベースを参照する外部スキーマを作成します。

```
CREATE EXTERNAL SCHEMA [IF NOT EXISTS] myRedshiftSchema
FROM MYSQL
DATABASE 'my_aurora_db'
URI 'endpoint to aurora hostname'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyAuroraRole'
SECRET_ARN 'arn:aws:secretsmanager:us-east-2:123456789012:secret:development/
MyTestDatabase-AbCdEf'
```

CREATE EXTERNAL TABLE

指定のスキーマに新しい外部テーブルを作成します。外部テーブルはすべて、外部スキーマで作成されている必要があります。外部スキーマと外部テーブルは検索パスをサポートしていません。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

Amazon Redshift では、CREATE EXTERNAL TABLE コマンドを使用して作成された外部テーブルに加えて、AWS Glueまたは AWS Lake Formation カタログ、あるいは Apache Hive メタストアで定義された外部テーブルの参照が可能です。CREATE EXTERNAL SCHEMA コマンドを使用して、外部カタログで定義された外部データベースを登録し、外部テーブルを Amazon Redshift で使用できるようにします。外部テーブルが AWS Glue または AWS Lake Formation カタログあるいは Hive メタストアに存在する場合は、CREATE EXTERNAL TABLE を使用してテーブルを作成する必要はありません。外部テーブルを表示するには、SVV_EXTERNAL_TABLESシステムビューに対してクエリを実行します。

CREATE EXTERNAL TABLE AS コマンドを実行することで、クエリからの列定義に基づいて外部テーブルを作成し、そのクエリの結果を Amazon S3 に書き込むことができます。結果は、Apache Parquet または区切りテキスト形式です。外部テーブルにパーティションキーがある場合、Amazon Redshift はそれらのパーティションキーに従って新しいファイルをパーティション分割し、新しいパーティションを外部カタログに自動的に登録します。CREATE EXTERNAL TABLE AS の詳細については、「使用に関する注意事項」を参照してください。

他の Amazon Redshift テーブルで使用したものと同じ SELECT 構文で、外部テーブルにクエリを実行できます。INSERT 構文を使用して、Amazon S3 の外部テーブルの場所に新しいファイルを書き込むこともできます。詳細については、「INSERT (外部テーブル)」を参照してください。

外部テーブルでビューを作成するには、CREATE VIEWステートメントに WITH NO SCHEMA BINDING 句を含めます。

トランザクション内 (BEGIN… END) で CREATE EXTERNAL TABLE を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

必要な権限

外部テーブルを作成するには、外部スキーマの所有者またはスーパーユーザーである必要があります。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用して所有者を変更します。外部テーブルへのアクセスは、外部スキーマへのアクセスによってコントロールされます。外部テーブルに対してアクセス権限の GRANT または REVOKE を実行することはできません。代わりに、外部スキーマに対して USAGE の付与または削除を実行します。

使用に関する注意事項 には、外部テーブルに対する特定のアクセス許可に関する追加情報があります。

構文

```
CREATE EXTERNAL TABLE
external_schema.table_name
(column_name data_type [, …] )
[ PARTITIONED BY (col_name data_type [, … ] )]
[ { ROW FORMAT DELIMITED row_format |
  ROW FORMAT SERDE 'serde_name'
  [ WITH SERDEPROPERTIES ( 'property_name' = 'property_value' [, ...] ) ] } ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file' }
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
```

以下に示しているのは、CREATE EXTERNAL TABLE AS の構文です。

```
CREATE EXTERNAL TABLE
external_schema.table_name
[ PARTITIONED BY (col_name [, … ] ) ]
[ ROW FORMAT DELIMITED row_format ]
STORED AS file_format
LOCATION { 's3://bucket/folder/' }
[ TABLE PROPERTIES ( 'property_name'='property_value' [, ...] ) ]
 AS
 { select_statement }
```

パラメータ

external_schema.table_name

作成され、外部スキーマ名で修飾されるテーブルの名前。外部テーブルは、外部スキーマで作成されている必要があります。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

テーブル名の最大長は 127 バイトです。それより長い名前は 127 バイトまで切り詰められます。最大 4 バイトまで UTF-8 マルチバイト文字を使用できます。Amazon Redshift では、ユーザー定義の一時テーブルと、クエリ処理またはシステムメンテナンス中に Amazon Redshift によって作成された一時テーブルを含め、クラスターごとに 9,900 テーブルの制限を適用します。必要に応じて、データベース名でテーブル名を修飾することができます。次の例では、データベース名は `spectrum_db`、外部スキーマ名は `spectrum_schema`、テーブル名は `test` です。

```
create external table spectrum_db.spectrum_schema.test (c1 int)
stored as parquet
location 's3://amzn-s3-demo-bucket/myfolder/';
```

指定のデータベースまたはスキーマが存在せず、テーブルが作成されていない場合、このステートメントはエラーを返します。システムデータベース `template0`、`template1`、`padb_harvest`、または `sys:internal` にテーブルまたはビューを作成することはできません。

テーブル名は、指定のスキーマで一意の名前にする必要があります。

有効な名前の詳細については、「名前と識別子」を参照してください。

(column_name data_type)

作成される各列の名前とデータタイプ。

列名の最大長は 127 バイトです。それより長い名前は 127 バイトまで切り詰められます。最大 4 バイトまで UTF-8 マルチバイト文字を使用できます。列名 `"$path"` または `"$size"` は指定できません。有効な名前の詳細については、「名前と識別子」を参照してください。

デフォルトでは、Amazon Redshift は疑似列 (`$path` および `$size`) を使用して外部テーブルを作成します。セッションの疑似列の作成を無効にするには、`spectrum_enable_pseudo_columns`設定パラメータを `false` に設定します。詳細については、「疑似列 」を参照してください。

擬似列が有効な場合、1 つのテーブルで定義できる列の最大数は 1,598 です。擬似列が有効でない場合、1 つのテーブルで定義できる列の最大数は 1,600 です。

「横長のテーブル」を作成する場合は、ロードとクエリ処理の結果が即時に表示されるように、列リストが行幅の限度を超えないことを確認します。詳細については、「使用に関する注意事項」を参照してください。

CREATE EXTERNAL TABLE AS コマンドの場合、列はクエリから取得されるため、列リストは必要ありません。

data_type

次の データ型 がサポートされています。

- SMALLINT (INT2)
- INTEGER (INT、INT4)
- BIGINT (INT8)
- DECIMAL (NUMERIC)
- REAL (FLOAT4)
- DOUBLE PRECISION (FLOAT8)
- BOOLEAN (BOOL)
- CHAR (CHARACTER)
- VARCHAR (CHARACTER VARYING)
- VARBYTE (CHARACTER VARYING) – Parquet および ORC データファイルで、パーティション化されていないテーブルでのみ使用できます。
- DATE – テキスト、Parquet、または ORC データファイルでのみ使用できます。またはパーティション列としてのみ使用できます。
- TIMESTAMP

DATE では、以下に示す形式を使用できます。数字を使用して表される月の値では、次の形式がサポートされています。

- `mm-dd-yyyy` は、例えば、`05-01-2017` です。これがデフォルトです。
- `yyyy-mm-dd`、ただし年は 2 桁以上で表します。例えば、`2017-05-01` と指定します。

3 文字の略語を使用して表される月の値では、次の形式がサポートされています。

- `mmm-dd-yyyy` は、例えば、`may-01-2017` です。これがデフォルトです。

- dd-mmm-yyyy、ただし年は 2 桁以上で表します。例えば、01-may-2017 と指定します。
- yyyy-mmm-dd、ただし年は 3 桁以上で表します。例えば、2017-may-01 と指定します。

年の値が一貫して 100 未満の場合、年は次の方法で計算されます。

- 年の値が 70 より小さい場合、年は 2000 を足した数として計算されます。例えば、mm-dd-yyyy形式での 05-01-17 という日付は 05-01-2017 に変換されます。
- 年の値が 100 未満で 69 より大きい場合、年は 1900 を足した数として計算されます。例えば、mm-dd-yyyy形式での 05-01-89 という日付は 05-01-1989 に変換されます。
- 2 桁で表される年の値については、先頭にゼロを追加した 4 桁として年を表します。

テキストファイルのタイムスタンプ値は、yyyy-mm-dd HH:mm:ss.SSSSSS形式であることが必要です。次のタイムスタンプ値は、2017-05-01 11:30:59.000000となっています。

VARCHAR 列の長さは、文字単位ではなくバイト単位で定義されます。例えば、VARCHAR(12) 列には、シングルバイト文字なら 12 個、2 バイト文字なら 6 個含めることができます。外部テーブルのクエリを実行すると、エラーが返されずに、定義された列サイズに合うように結果が切り捨てられます。詳細については、「ストレージと範囲」を参照してください。

最高のパフォーマンスを得るために、データに合う最小の列サイズを指定することをお勧めします。列の値の最大サイズ (バイト単位) を調べるには、OCTET_LENGTH 関数を使用します。次の例では、E メール列の値の最大サイズを返します。

```
select max(octet_length(email)) from users;

max
---
 62
```

PARTITIONED BY (col_name data_type [, …])

1 つ以上のパーティション列でパーティション化されたテーブルを定義する句。指定の組み合わせごとに独立したデータディレクトリが使用されます。これにより、一部の状況でクエリパフォーマンスが向上します。パーティション化された列はテーブルデータ内には存在しません。テーブル列と同じ col_name の値を使用すると、エラーになります。

パーティションテーブルを作成した後、ALTER TABLE… ADD PARTITION ステートメントを使用してテーブルを変更し、新しいパーティションを外部カタログに登録します。パーティションを追加する際は、パーティションデータを含む で Amazon S3 サブフォルダの位置を定義します。

たとえば、テーブル `spectrum.lineitem_part` が `PARTITIONED BY (l_shipdate date)` で定義されている場合は、以下の ALTER TABLE コマンドを実行してパーティションを追加します。

```
ALTER TABLE spectrum.lineitem_part ADD PARTITION (l_shipdate='1992-01-29')
LOCATION 's3://spectrum-public/lineitem_partition/l_shipdate=1992-01-29';
```

CREATE EXTERNAL TABLE AS を使用する場合、ALTER TABLE...ADD PARTITION を実行する必要はありません。Amazon Redshift は、新しいパーティションを外部カタログに自動的に登録します。また、Amazon Redshift は、テーブルに定義された 1 つまたは複数のパーティションキーに基づいて、対応するデータを Amazon S3 のパーティションに自動的に書き込みます。

パーティションを表示するには、SVV_EXTERNAL_PARTITIONSシステムビューにクエリを実行します。

> **Note**
>
> CREATE EXTERNAL TABLE AS コマンドの場合、この列はクエリから取得されるため、パーティション列のデータ型を指定する必要はありません。

ROW FORMAT DELIMITED rowformat

仮想データの型式を指定する句。以下に、rowformat で想定される値を示します。

- LINES TERMINATED BY 'delimiter'
- FIELDS TERMINATED BY 'delimiter'

'区切り記号'に 1 つの ASCII 文字を指定します。印刷不能な ASCII 文字は、'*ddd*' の形式 (*d* は、0～7 で表され最大 '\177' までを取る 8 進数) を使用して指定できます。次の例では、BEL (ベル) 文字を 8 進数で指定しています。

```
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\007'
```

ROW FORMAT を省略した場合のデフォルト形式は DELIMITED FIELDS TERMINATED BY '\A' (ヘッダーの開始) と LINES TERMINATED BY '\n' (改行) です。

ROW FORMAT SERDE 'serde_name', [WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

基盤となるデータの SERDE 形式を指定する句。

'serde_name'

SerDe 名。以下の形式を指定できます。

- org.apache.hadoop.hive.serde2.RegexSerDe
- com.amazonaws.glue.serde.GrokSerDe
- org.apache.hadoop.hive.serde2.OpenCSVSerde

  このパラメータは、OpenCSVSerde において、次の SerDe プロパティをサポートします。

  ```
  'wholeFile' = 'true'
  ```

  `wholeFile` プロパティに `true` を設定することで、OpenCSV リクエストの引用符で囲まれた文字列内の改行文字 (\ n) を、正しく解析することができます。

- org.openx.data.jsonserde.JsonSerDe
  - JSON SERDE は Ion ファイルもサポートしています。
  - JSON は正しい形式になっている必要があります。
  - Ion および JSON 形式のタイムスタンプには、ISO8601 形式を使用する必要があります。
  - このパラメータでは、JsonSerDe のための、次の SerDe プロパティがサポートされています。

    ```
    'strip.outer.array'='true'
    ```

    配列内に複数の JSON レコードが含まれているかのように、大括弧で囲まれた 1 つの非常に大きな配列 ([…]) を含む Ion/JSON ファイルを処理します。

- com.amazon.ionhiveserde.IonHiveSerDe

  Amazon ION 形式では、データ型に加えて、テキスト形式とバイナリ形式を使用できます。ION 形式のデータを参照する外部テーブルの場合、外部テーブルの各列を、対応する ION 形式データの各要素にマッピングします。詳細については、「Amazon Ion」を参照してください。また、入力形式と出力形式を指定する必要もあります。

WITH SERDEPROPERTIES ('property_name' = 'property_value' [, ...])]

オプションで、プロパティ名と値をコンマで区切って指定します。

ROW FORMAT を省略した場合のデフォルト形式は DELIMITED FIELDS TERMINATED BY '\A' (ヘッダーの開始) と LINES TERMINATED BY '\n' (改行) です。

STORED AS file_format

外部データファイルのファイル形式。

有効な形式は次のとおりです。

- PARQUET
- RCFILE (LazyBinaryColumnarSerDe ではなく ColumnarSerD eのみを使用するデータ用)
- SEQUENCEFILE
- TEXTFILE (JSON ファイルを含むテキストファイル)。
- ORC
- AVRO
- INPUTFORMAT 'input_format_classname' OUTPUTFORMAT 'output_format_classname'

CREATE EXTERNAL TABLE AS コマンドは、TEXTFILE と PARQUET の 2 つのファイル形式のみをサポートしています。

INPUTFORMAT と OUTPUTFORMAT では、以下の例に示すように、クラス名を指定します。

```
'org.apache.hadoop.mapred.TextInputFormat'
```

LOCATION { 's3://bucket/folder/' | 's3://bucket/manifest_file'}

データファイルを含む Amazon S3 バケットかフォルダ、または Amazon S3 オブジェクトパスのリストを含むマニフェストファイルへのパス。バケットは、Amazon Redshift クラスターと同じ AWS リージョン内に置かれている必要があります。サポートされている AWS リージョンの一覧は、「Amazon Redshift Spectrum の制限事項」でご確認ください。

パスがバケットかフォルダを指定している場合 (`'s3://amzn-s3-demo-bucket/custdata/'` など)、Redshift Spectrum は指定されたバケットかフォルダ、およびサブフォルダのファイルをスキャンします。Redshift Spectrum は、隠しファイルと、ファイル名がピリオドやアンダースコアで始まるファイルを無視します。

パスがマニフェストファイルを指定している場合、`'s3://bucket/manifest_file'`引数は 1 つのファイル (たとえば `'s3://amzn-s3-demo-bucket/manifest.txt'` など) を明示的に参照する必要があります。キープレフィックスを参照することはできません。

マニフェストは、JSON 形式のテキストファイルであり、Amazon S3 からロードする各ファイルの URL とファイルサイズ (バイト単位) を示します。URL にはバケット名およびファイルの完全オブジェクトパスが含まれます。マニフェストで指定するファイルの場所は異なるバケットでも

かまいませんが、すべてのバケットは Amazon Redshift クラスターと同じ AWS リージョンに置かれている必要があります。ファイルが 2 回リストされている場合、ファイルは 2 回ロードされます。次の例は、3 つのファイルをロードするマニフェストの JSON を示しています。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1", "meta": { "content_length":
 5956875 } },
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2", "meta": { "content_length":
 5997091 } },
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1", "meta": { "content_length":
 5978675 } }
  ]
}
```

特定のファイルを含めることを必須にすることができます。これを行うには、マニフェストのファイルレベルで `mandatory` オプションを含めます。欠落している必須ファイルを使用して外部テーブルをクエリすると、SELECT ステートメントは失敗します。外部テーブルの定義に含まれるすべてのファイルが存在することを確認します。これらがすべて存在しない場合は、最初の必須ファイルが見つかりませんというエラーが表示されます。次の例では、`mandatory`オプションを `true` に設定したマニフェストの JSON を説明します。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket1/custdata.1", "mandatory":true, "meta":
 { "content_length": 5956875 } },
    {"url":"s3://amzn-s3-demo-bucket1/custdata.2", "mandatory":false, "meta":
 { "content_length": 5997091 } },
    {"url":"s3://amzn-s3-demo-bucket2/custdata.1", "meta": { "content_length":
 5978675 } }
  ]
}
```

UNLOAD を使用して作成したファイルを参照する場合は、UNLOADで MANIFEST パラメータを使用して作成したマニフェストを使用できます。マニフェストファイルは、「Amazon S3 からの COPY」でのマニフェストファイルと互換性がありますが、使用するキーが異なります。使用しないキーは無視されます。

TABLE PROPERTIES ('property_name'='property_value' [, ...])

テーブルプロパティのテーブル定義を設定する句。

> **Note**
>
> テーブルのプロパティでは、大文字と小文字が区別されます。

'compression_type'='value'

ファイル名に拡張子が含まれていない場合に使用する圧縮タイプを設定するプロパティ。ファイル拡張子があるときに、このプロパティを設定すると、拡張子は無視されて、プロパティで設定された値が使用されます。圧縮タイプの有効値は次のとおりです。

- bzip2
- gzip
- なし
- snappy

'data_cleansing_enabled'='true / false’

このプロパティは、テーブルのデータ処理が有効かどうかを設定します。'data_cleansing_enabled' が true に設定されている場合は、テーブルのデータ処理が有効です。'data_cleansing_enabled' が false に設定されている場合、テーブルのデータ処理は無効です。以下は、このプロパティが制御するテーブルレベルでのデータ処理プロパティのリストです。

- column_count_mismatch_handling
- invalid_char_handling
- overflow_handling
- replacement_char
- surplus_char_handling

例については、「データ処理の例」を参照してください。

'invalid_char_handling'='value'

クエリ結果に無効な UTF-8 文字値が含まれている場合に実行するアクションを指定します。以下のアクションを指定できます。

DISABLED

無効な文字の処理を実行しません。

FAIL

無効な UTF-8 値が含まれたデータを返すクエリをキャンセルします。

SET_TO_NULL

無効な UTF-8 値を null に置き換えます。

DROP_ROW

行内の各値を null に置き換えます。

REPLACE

`replacement_char` を使用して無効な文字を指定した置換文字に置き換えます。

'replacement_char'='character’

`invalid_char_handling` を `REPLACE` に設定するときに使用する置換文字を指定します。

'numeric_overflow_handling'='value’

ORC データに列定義 (例えば SMALLINT や int16) よりも大きい整数 (例えば BIGINT や int64) が含まれているときに実行するアクションを指定します。以下のアクションを指定できます。

DISABLED

無効な文字の処理が無効化されています。

FAIL

データに無効な文字が含まれているときにクエリをキャンセルします。

SET_TO_NULL

無効な文字を null に設定します。

DROP_ROW

行内の各値を null に設定します。

'surplus_bytes_handling'='value'

ロードされているデータで、VARBYTE データが含まれる列に定義されたデータ型の長さを超えるデータの処理方法を指定します。Redshift Spectrum はデフォルトで、列の幅を超えるデータの値を null に設定します。

クエリがデータ型の長さを超えるデータを返すときに実行する以下のアクションを指定できます。

SET_TO_NULL

列幅を超えるデータを null に置き換えます。

無効

余剰バイトの処理を実行しません。

FAIL

列幅を超えるデータを返すクエリをキャンセルします。

DROP_ROW

列幅を超えるデータを含むすべての行を削除します。

TRUNCATE

列に定義された最大文字数を超える文字を削除します。

'surplus_char_handling'='value'

ロードされているデータで、VARCHAR、CHAR、または文字列データが含まれる列に定義されたデータ型の長さを超えるデータの処理方法を指定します。Redshift Spectrum はデフォルトで、列の幅を超えるデータの値を null に設定します。

クエリが列幅を超えるデータを返すときに実行する以下のアクションを指定できます。

SET_TO_NULL

列幅を超えるデータを null に置き換えます。

無効

余剰文字の処理を実行しません。

FAIL

列幅を超えるデータを返すクエリをキャンセルします。

DROP_ROW

行内の各値を null に置き換えます。

TRUNCATE

列に定義された最大文字数を超える文字を削除します。

'column_count_mismatch_handling'='value'

ファイルに含まれる行の値が、外部テーブル定義で指定された列数よりも少ないか多いかを識別します。このプロパティは、非圧縮テキストファイル形式でのみ使用できます。以下のアクションを指定できます。

DISABLED

列数の不一致処理が無効化されています。

FAIL

列数の不一致が検出された場合、クエリは失敗します。

SET_TO_NULL

欠落した値を NULL で埋め、各行の追加の値を無視します。

DROP_ROW

列数の不一致エラーを含むすべての行をスキャンからドロップします。

'numRows'='row_count'

テーブル定義の numRows 値を設定するプロパティ。外部テーブルの統計を明示的に更新するには、テーブルのサイズを示す numRows プロパティを設定します。Amazon Redshift は、外部テーブルを分析して、クエリオプティマイザがクエリプランを生成するために使用するテーブル統計を生成することはありません。外部テーブルに対してテーブル統計が設定されていない場合、Amazon Redshift は、外部テーブルが大きなテーブルであり、ローカルテーブルが小さいテーブルであるという前提に基づいてクエリ実行プランを生成します。

'skip.header.line.count'='line_count'

各ソースファイルの最初に省略する行数を設定するプロパティ。

'serialization.null.format'=' '

範囲を示すプロパティは、フィールドで指定されたテキストに完全に一致するものがある場合に、値 `NULL` を返します。

'orc.schema.resolution'='mapping_type'

ORC データ形式を使用するテーブルの列マッピングタイプを設定するプロパティ。このプロパティは、他のデータ形式では無視されます。

列マッピングタイプの有効値は次のとおりです。

- name
- position

orc.schema.resolution プロパティが省略されている場合、列はデフォルトで、名前を基準としてマップされます。orc.schema.resolution が 'name' または 'position' 以外の値に設定されている場合、列は位置を基準としてマップされます。列マッピングの詳細については、「外部テーブル列を ORC 列にマッピングする」を参照してください。

> **Note**
>
> COPY コマンドは、位置のみを基準として ORC データファイルにマップします。orc.schema.resolution テーブルプロパティは、COPY コマンドの動作には影響しません。

'write.parallel'='on / off'

CREATE EXTERNAL TABLE AS がデータを並列で書き込む必要があるかどうかを設定するプロパティ。デフォルトでは、CREATE EXTERNAL TABLE AS は、クラスター内のスライスの数に応じて、データを複数のファイルに並列で書き込みます。デフォルトのオプションは on です。'write.parallel' が off に設定されている場合、CREATE EXTERNAL TABLE AS は 1 つ以上のデータファイルを Amazon S3 に順次に書き込みます。このテーブルプロパティは、同じ外部テーブルへの後続の INSERT ステートメントにも適用されます。

'write.maxfilesize.mb'='size'

CREATE EXTERNAL TABLE AS によって Amazon S3 に書き込まれる各ファイルの最大サイズ (MB 単位) を設定するプロパティ。サイズは 5 ～ 6200 の有効な整数であることが必要です。デフォルトの最大ファイルサイズは 6,200 MB です。このテーブルプロパティは、同じ外部テーブルへの後続の INSERT ステートメントにも適用されます。

‘write.kms.key.id’=‘value’

Amazon S3 オブジェクトのサーバー側の暗号化 (SSE) を有効にする AWS Key Management Service キーを指定できます。この value は、以下のいずれかになります。

- Amazon S3 バケットに保存されたデフォルトの AWS KMS キーを使用するための `auto`。
- データを暗号化するために指定する kms-key。

select_statement

クエリを定義して 1 つ以上の行を外部テーブルに挿入するステートメント。クエリによって生成されるすべての行は、テーブル定義に基づいて、テキストまたは Parquet 形式で Amazon S3 に書き込まれます。

例

例のコレクションは、例 にあります。

使用に関する注意事項

このトピックには、CREATE EXTERNAL TABLE の使用上の注意事項が含まれています。PG_TABLE_DEF、STV_TBL_PERM、PG_CLASS、または information_schema など、標準の Amazon Redshift テーブルに使用したものと同じリソースを使用して Amazon Redshift Spectrum テーブルの詳細を表示することはできません。ビジネスインテリジェンスまたは分析ツールが Redshift Spectrum 外部テーブルを認識しない場合は、SVV_EXTERNAL_TABLESおよび SVV_EXTERNAL_COLUMNS にクエリを実行するようにアプリケーションを設定します。

CREATE EXTERNAL TABLE AS

一部のケースでは、AWS Glueデータカタログ、AWS Lake Formation外部カタログ、または Apache Hive メタストアに対して、CREATE EXTERNAL TABLE AS コマンドを実行します。このような場合は、AWS Identity and Access Management(IAM) ロールを使用して外部スキーマを作成します。この IAM ロールには、Amazon S3 に対する読み込みと書き込みの両方のアクセス許可が必要です。

Lake Formation カタログを使用する場合、IAM ロールにはカタログにテーブルを作成するアクセス許可が必要です。この場合、ターゲット Amazon S3 パスに対するデータレイクの場所のアクセス許可が必要です。この IAM ロールは新しい AWS Lake Formation テーブルの所有者になります。

ファイル名が必ず一意になるように、Amazon Redshift ではデフォルトで、Amazon S3 にアップロードされる各ファイルの名前に以下の形式が使用されます。

*<date>*_*<time>*_*<microseconds>*_*<query_id>*_*<slice-number>*_part_*<part-number>*.*<format>*.

例は `20200303_004509_810669_1007_0001_part_00.parquet` です。

CREATE EXTERNAL TABLE AS コマンドを実行するときは、以下の点を考慮してください。

- Amazon S3 の場所は空であることが必要です。
- Amazon Redshift では、STORED AS 句を使用する場合、PARQUET および TEXTFILE 形式のみがサポートされます。
- 列定義リストを定義する必要はありません。新しい外部テーブルの列の名前とデータ型は、SELECT クエリから直接派生します。
- PARTITIONED BY 句でパーティション列のデータ型を定義する必要はありません。パーティションキーを指定する場合、この列の名前が SELECT クエリの結果にあることが必要です。複数のパーティション列がある場合、SELECT クエリでのそれらの順序は重要ではありません。Amazon Redshift は、PARTITIONED BY 句で定義された順序を使用して、外部テーブルを作成します。
- Amazon Redshift は、パーティションキーの値に基づいて、出力ファイルをパーティションフォルダに自動的にパーティション分割します。デフォルトでは、Amazon Redshift はパーティション列を出力ファイルから削除します。
- LINES TERMINATED BY 'delimiter' 句はサポートされていません。
- ROW FORMAT SERDE 'serde_name' 句はサポートされていません。
- マニフェストファイルの使用はサポートされていません。したがって、Amazon S3 のマニフェストファイルに LOCATION 句を定義することはできません。
- Amazon Redshift は、コマンドの最後で 'numRows' テーブルプロパティを自動的に更新します。
- 'compression_type' テーブルプロパティは、PARQUET ファイル形式に基づいて 'none' または 'snappy' のみを受け入れます。
- Amazon Redshift では、外側の SELECT クエリで LIMIT 句を使用できません。代わりに、ネストされた LIMIT 句を使用できます。
- STL_UNLOAD_LOG を使用して、各 CREATE EXTERNAL TABLE AS オペレーションによって Amazon S3 に書き込まれたファイルを追跡できます。

外部テーブルの作成およびクエリのアクセス許可

外部テーブルを作成するには、外部スキーマの所有者またはスーパーユーザーであることを確認してください。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用します。次の例は、`spectrum_schema`スキーマの所有者を `newowner` に変更します。

```
alter schema spectrum_schema owner to newowner;
```

Redshift Spectrum クエリを実行するには、次のアクセス権限が必要です。

- スキーマのアクセス権限の使用

- 現在のデータベースに一時テーブルを作成するアクセス権限

次の例では、スキーマ `spectrum_schema` の使用許可を `spectrumusers` ユーザーグループに付与しています。

```
grant usage on schema spectrum_schema to group spectrumusers;
```

次の例では、データベース `spectrumdb` の一時アクセス権限を `spectrumusers` ユーザーグループに付与しています。

```
grant temp on database spectrumdb to group spectrumusers;
```

疑似列

Amazon Redshift はデフォルトで疑似列 $path および $size を使用して外部テーブルを作成します。これらの列を選択すると、Amazon S3 のデータファイルへのパスとクエリによって返された各行のデータファイルのサイズが表示されます。列名 ($path および $size) は、二重引用符で囲む必要があります。SELECT * 句は、疑似列を返しません。次の例に示すように、$path と $size の列名をクエリに明示的に含める必要があります。

```
select "$path", "$size"
from spectrum.sales_part
where saledate = '2008-12-01';
```

セッションの疑似列の作成を無効にするには、spectrum_enable_pseudo_columns 設定パラメータを false に設定します。

> **Important**
>
> Redshift Spectrum では、Amazon S3 のデータファイルをスキャンして結果セットのサイズを確認しているため、$size または $path を選択すると料金が発生します。詳細については、[Amazon Redshift の料金](#)を参照してください。

データ処理オプションの設定

テーブルパラメータを設定して、外部テーブルでクエリされているデータの入力処理を指定できます。これには、以下が含まれます。

- VARCHAR、CHAR、および文字列データが含まれる列内の余剰文字。詳細については、外部テーブルプロパティ「`surplus_char_handling`」を参照してください。
- VARCHAR、CHAR、および文字列データが含まれる列内の無効な文字。詳細については、外部テーブルプロパティ「`invalid_char_handling`」を参照してください。
- 外部テーブルプロパティ `invalid_char_handling` に REPLACE を指定するときに使用する置換文字。
- 整数と小数のデータが含まれる列内でのキャストのオーバーフロー処理。詳細については、外部テーブルプロパティ「`numeric_overflow_handling`」を参照してください。
- Surplus_bytes_handling は、varbyte データを含む列の余剰バイトの入力処理を指定します。詳細については、外部テーブルプロパティ「`surplus_bytes_handling`」を参照してください。

例

次の例では、SALES という名前のテーブルを `spectrum` という名前の Amazon Redshift 外部スキーマに作成します。データはタブ区切りのテキストファイルになっています。TABLE PROPERTIES 句は、numRows プロパティを 170,000 行に設定します。

CREATE EXTERNAL TABLE の実行に使用する ID によっては、設定が必要な IAM アクセス許可が存在する場合があります。ベストプラクティスとして、アクセス許可ポリシーを IAM ロールにアタッチし、それを必要に応じてユーザーやグループに割り当てることをお勧めします。詳細については、「Amazon Redshift での Identity and Access Management」を参照してください。

```
create external table spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
saledate date,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales/'
table properties ('numRows'='170000');
```

次の例では、JsonSerDe を使用して JSON 形式のデータを参照するテーブルを作成します。

```
create external table spectrum.cloudtrail_json (
event_version int,
event_id bigint,
event_time timestamp,
event_type varchar(10),
awsregion varchar(20),
event_name varchar(max),
event_source varchar(max),
requesttime timestamp,
useragent varchar(max),
recipientaccountid bigint)
row format serde 'org.openx.data.jsonserde.JsonSerDe'
with serdeproperties (
'dots.in.keys' = 'true',
'mapping.requesttime' = 'requesttimestamp'
) location 's3://amzn-s3-demo-bucket/json/cloudtrail';
```

以下の CREATE EXTERNAL TABLE AS の例では、パーティション分割されていない外部テーブルを作成します。次に、SELECT クエリの結果を Apache Parquet として Amazon S3 のターゲットの場所に書き込みます。

```
CREATE EXTERNAL TABLE spectrum.lineitem
STORED AS parquet
LOCATION 'S3://amzn-s3-demo-bucket/cetas/lineitem/'
AS SELECT * FROM local_lineitem;
```

以下の例では、パーティション分割された外部テーブルを作成し、パーティション列を SELECT クエリに含めます。

```
CREATE EXTERNAL TABLE spectrum.partitioned_lineitem
PARTITIONED BY (l_shipdate, l_shipmode)
STORED AS parquet
LOCATION 'S3://amzn-s3-demo-bucket/cetas/partitioned_lineitem/'
AS SELECT l_orderkey, l_shipmode, l_shipdate, l_partkey FROM local_table;
```

外部データカタログ内の既存のデータベースについて、[SVV_EXTERNAL_DATABASES](#)システムビューにクエリを実行します。

```
select eskind,databasename,esoptions from svv_external_databases order by databasename;
```

```
eskind | databasename | esoptions
-------+--------------
+-------------------------------------------------------------------------------------
     1 | default      | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
     1 | sampledb     | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
     1 | spectrumdb   | {"REGION":"us-
west-2","IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

外部テーブルの詳細を表示するには、SVV_EXTERNAL_TABLESおよび SVV_EXTERNAL_COLUMNS システムビューにクエリを実行します。

次の例では、SVV_EXTERNAL_TABLES ビューにクエリを実行します。

```
select schemaname, tablename, location from svv_external_tables;
```

```
schemaname | tablename            | location
-----------+----------------------
+---------------------------------------------------
spectrum   | sales                | s3://redshift-downloads/tickit/spectrum/sales
spectrum   | sales_part           | s3://redshift-downloads/tickit/spectrum/
sales_partition
```

次の例では、SVV_EXTERNAL_COLUMNS ビューにクエリを実行します。

```
select * from svv_external_columns where schemaname like 'spectrum%' and tablename
 ='sales';
```

```
schemaname | tablename | columnname | external_type | columnnum | part_key
-----------+-----------+------------+---------------+-----------+---------
spectrum   | sales     | salesid    | int           |         1 |        0
spectrum   | sales     | listid     | int           |         2 |        0
spectrum   | sales     | sellerid   | int           |         3 |        0
spectrum   | sales     | buyerid    | int           |         4 |        0
spectrum   | sales     | eventid    | int           |         5 |        0
spectrum   | sales     | saledate   | date          |         6 |        0
spectrum   | sales     | qtysold    | smallint      |         7 |        0
spectrum   | sales     | pricepaid  | decimal(8,2)  |         8 |        0
spectrum   | sales     | commission | decimal(8,2)  |         9 |        0
```

```
spectrum    | sales      | saletime    | timestamp      |          10 |          0
```

テーブルパーティションを表示するには、次のクエリを使用します。

```
select schemaname, tablename, values, location
from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values          | location
-----------+------------+----------------
+-------------------------------------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

次の例では、外部テーブルの関連データファイルの合計サイズを返します。

```
select distinct "$path", "$size"
   from spectrum.sales_part;

 $path                                                                    | $size
```

```
---------------------------------------------------------------------------+-------
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/ |  1616
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
s3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/ |  1444
```

パーティション化の例

日付でパーティション化された外部テーブルを作成するには、次のコマンドを実行します。

```
create external table spectrum.sales_part(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
partitioned by (saledate date)
row format delimited
fields terminated by '|'
stored as textfile
location 's3://redshift-downloads/tickit/spectrum/sales_partition/'
table properties ('numRows'='170000');
```

パーティションを追加するには、次の ALTER TABLE コマンドを実行します。

```
alter table spectrum.sales_part
add if not exists partition (saledate='2008-01-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-01/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-02-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-02/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-03-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-03/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-04-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-04/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-05-01')
```

```
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-05/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-06-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-06/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-07-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-07/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-08-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-08/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-09-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-09/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-10-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-10/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-11-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-11/';
alter table spectrum.sales_part
add if not exists partition (saledate='2008-12-01')
location 's3://redshift-downloads/tickit/spectrum/sales_partition/saledate=2008-12/';
```

パーティション化されたテーブルからデータを選択するには、次のクエリを実行します。

```
select top 10 spectrum.sales_part.eventid, sum(spectrum.sales_part.pricepaid)
from spectrum.sales_part, event
where spectrum.sales_part.eventid = event.eventid
  and spectrum.sales_part.pricepaid > 30
  and saledate = '2008-12-01'
group by spectrum.sales_part.eventid
order by 2 desc;
```

```
eventid | sum
--------+---------
    914 | 36173.00
   5478 | 27303.00
   5061 | 26383.00
   4406 | 26252.00
   5324 | 24015.00
   1829 | 23911.00
   3601 | 23616.00
   3665 | 23214.00
```

```
    6069 | 22869.00
    5638 | 22551.00
```

外部テーブルパーティションを表示するには、SVV_EXTERNAL_PARTITIONSシステムビューにクエリを実行します。

```
select schemaname, tablename, values, location from svv_external_partitions
where tablename = 'sales_part';
```

```
schemaname | tablename  | values         | location
-----------+------------+----------------
+-------------------------------------------------
spectrum   | sales_part | ["2008-01-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-01
spectrum   | sales_part | ["2008-02-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-02
spectrum   | sales_part | ["2008-03-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-03
spectrum   | sales_part | ["2008-04-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-04
spectrum   | sales_part | ["2008-05-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-05
spectrum   | sales_part | ["2008-06-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-06
spectrum   | sales_part | ["2008-07-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-07
spectrum   | sales_part | ["2008-08-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-08
spectrum   | sales_part | ["2008-09-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-09
spectrum   | sales_part | ["2008-10-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-10
spectrum   | sales_part | ["2008-11-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-11
spectrum   | sales_part | ["2008-12-01"] | s3://redshift-downloads/tickit/spectrum/
sales_partition/saledate=2008-12
```

行形式の例

次に、AVRO 形式で保存されたデータファイルの ROW FORMAT SERDE パラメータの指定例を示します。

```
create external table spectrum.sales(salesid int, listid int, sellerid int,
 buyerid int, eventid int, dateid int, qtysold int, pricepaid decimal(8,2), comment
 VARCHAR(255))
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='{\"namespace\": \"dory.sample\",\"name\":
 \"dory_avro\",\"type\": \"record\", \"fields\": [{\"name\":\"salesid\", \"type\":\"int
\"},
{\"name\":\"listid\", \"type\":\"int\"},
{\"name\":\"sellerid\", \"type\":\"int\"},
{\"name\":\"buyerid\", \"type\":\"int\"},
{\"name\":\"eventid\",\"type\":\"int\"},
{\"name\":\"dateid\",\"type\":\"int\"},
{\"name\":\"qtysold\",\"type\":\"int\"},
{\"name\":\"pricepaid\", \"type\": {\"type\": \"bytes\", \"logicalType\": \"decimal\",
 \"precision\": 8, \"scale\": 2}}, {\"name\":\"comment\",\"type\":\"string\"}]}')
STORED AS AVRO
location 's3://amzn-s3-demo-bucket/avro/sales' ;
```

RegEx を使用して ROW FORMAT SERDE パラメータを指定する例を以下に示します。

```
create external table spectrum.types(
cbigint bigint,
cbigint_null bigint,
cint int,
cint_null int)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties ('input.regex'='([^\\x01]+)\\x01([^\\x01]+)\\x01([^\\x01]+)\
\x01([^\\x01]+)')
stored as textfile
location 's3://amzn-s3-demo-bucket/regex/types';
```

Grok を使用して ROW FORMAT SERDE パラメータを指定する例を以下に示します。

```
create external table spectrum.grok_log(
timestamp varchar(255),
pid varchar(255),
loglevel varchar(255),
progname varchar(255),
message varchar(255))
row format serde 'com.amazonaws.glue.serde.GrokSerDe'
```

```
with serdeproperties ('input.format'='[DFEWI], \\[%{TIMESTAMP_ISO8601:timestamp} #
%{POSINT:pid:int}\\] *(?<loglevel>:DEBUG|FATAL|ERROR|WARN|INFO) -- +%{DATA:progname}:
 %{GREEDYDATA:message}')
stored as textfile
location 's3://DOC-EXAMPLE-BUCKET/grok/logs';
```

次の例では、S3 バケットで Amazon S3 サーバーアクセスログを定義します。Redshift Spectrum を使用して、Amazon S3 アクセスログをクエリできます。

```
CREATE EXTERNAL TABLE spectrum.mybucket_s3_logs(
bucketowner varchar(255),
bucket varchar(255),
requestdatetime varchar(2000),
remoteip varchar(255),
requester varchar(255),
requested varchar(255),
operation varchar(255),
key varchar(255),
requesturi_operation varchar(255),
requesturi_key varchar(255),
requesturi_httpprotoversion varchar(255),
httpstatus varchar(255),
errorcode varchar(255),
bytessent bigint,
objectsize bigint,
totaltime varchar(255),
turnaroundtime varchar(255),
referrer varchar(255),
useragent varchar(255),
versionid varchar(255)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
'input.regex' = '([^ ]*) ([^ ]*) \\[(.*?)\\] ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 \"([^ ]*)\\s*([^ ]*)\\s*([^ ]*)\" (- |[^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)
 ([^ ]*) (\"[^\"]*\") ([^ ]*).*$')
LOCATION 's3://amzn-s3-demo-bucket/s3logs’;
```

次に、ION 形式のデータで ROW FORMAT SERDE パラメータを指定する例を示します。

```
CREATE EXTERNAL TABLE tbl_name (columns)
ROW FORMAT SERDE 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS
```

```
INPUTFORMAT 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/prefix'
```

データ処理の例

以下の例は、ファイル spi_global_rankings.csv にアクセスします。これらの例を試すには、spi_global_rankings.csvファイルを Amazon S3 バケットにアップロードできます。

以下の例は、外部スキーマ schema_spectrum_uddh とデータベース spectrum_db_uddh を作成します。aws-account-id には AWS アカウント ID、role-nameには Redshift Spectrum ロール名を入力します。

```
create external schema schema_spectrum_uddh
from data catalog
database 'spectrum_db_uddh'
iam_role 'arn:aws:iam::aws-account-id:role/role-name'
create external database if not exists;
```

以下の例は、外部スキーマ schema_spectrum_uddh で外部テーブル soccer_league を作成します。

```
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league
(
  league_rank smallint,
  prev_rank   smallint,
  club_name   varchar(15),
  league_name varchar(20),
  league_off  decimal(6,2),
  league_def  decimal(6,2),
  league_spi  decimal(6,2),
  league_nspi integer
)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n\l'
stored as textfile
LOCATION 's3://spectrum-uddh/league/'
table properties ('skip.header.line.count'='1');
```

soccer_league テーブル内の行数をチェックします。

```
select count(*) from schema_spectrum_uddh.soccer_league;
```

行数が表示されます。

```
count
645
```

以下のクエリは、上位 10 位のクラブを表示します。クラブ Barcelona は文字列に無効な文字が含まれているため、名前に NULL が表示されています。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

```
league_rank club_name league_name   league_nspi
1  Manchester City Barclays Premier Lea  34595
2  Bayern Munich German Bundesliga  34151
3  Liverpool Barclays Premier Lea  33223
4  Chelsea  Barclays Premier Lea  32808
5  Ajax  Dutch Eredivisie  32790
6  Atletico  Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi  31469
8  NULL         Spanish Primera Divi            31321
9  RB Leipzig German Bundesliga  31014
10  Paris Saint-Ger French Ligue 1   30929
```

以下の例は、soccer_league テーブルを変更して invalid_char_handling、replacement_char、および data_cleansing_enabled の外部テーブルプロパティを指定し、予期しない文字の代わりに疑問符 (?) を挿入するようにします。

```
alter  table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='REPLACE','replacement_char'='?','data_cleansing_enabled'='true');
```

以下の例は、ランクが 1 から 10 のチームについてテーブル soccer_league をクエリします。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

テーブルプロパティが変更されたため、結果は上位 10 位のクラブを表示し、Barcelona のクラブの第 8 列に疑問符 (?) 置換文字があります。

```
league_rank club_name league_name  league_nspi
1  Manchester City Barclays Premier Lea 34595
2  Bayern Munich German Bundesliga 34151
3  Liverpool Barclays Premier Lea 33223
4  Chelsea  Barclays Premier Lea 32808
5  Ajax  Dutch Eredivisie 32790
6  Atletico Madrid Spanish Primera Divi 31517
7  Real Madrid Spanish Primera Divi 31469
8  Barcel?na Spanish Primera Divi 31321
9  RB Leipzig German Bundesliga 31014
10  Paris Saint-Ger French Ligue 1  30929
```

お以下の例は、soccer_league テーブルを変更して invalid_char_handling 外部テーブルプロパティを指定し、予期しない文字が含まれる行をドロップするようにします。

```
alter table schema_spectrum_uddh.soccer_league
set table properties
 ('invalid_char_handling'='DROP_ROW','data_cleansing_enabled'='true');
```

以下の例は、ランクが 1 から 10 のチームについてテーブル soccer_league をクエリします。

```
select league_rank,club_name,league_name,league_nspi
from schema_spectrum_uddh.soccer_league
where league_rank between 1 and 10;
```

結果は上位のクラブを表示しますが、Barcelona のクラブの第 8 列は含まれません。

```

league_rank   club_name         league_name            league_nspi
1             Manchester City   Barclays Premier Lea   34595
2             Bayern Munich     German Bundesliga      34151
3             Liverpool         Barclays Premier Lea   33223
4             Chelsea           Barclays Premier Lea   32808
5             Ajax              Dutch Eredivisie       32790
6             Atletico Madrid   Spanish Primera Divi   31517
7             Real Madrid       Spanish Primera Divi   31469
9             RB Leipzig        German Bundesliga      31014
10            Paris Saint-Ger   French Ligue 1         30929
```

CREATE EXTERNAL VIEW

データカタログビューのプレビュー機能は以下のリージョンでのみ利用できます。

- 米国東部 (オハイオ) (us-east-2)
- 米国東部 (バージニア北部) (us-east-1)
- 米国西部 (北カリフォルニア) (us-west-1)
- アジアパシフィック (東京) (ap-northeast-1)
- 欧州 (アイルランド) (eu-west-1)
- 欧州 (ストックホルム) (eu-north-1)

データカタログでビューを作成します。データカタログビューは、Amazon Athena や Amazon EMR のような他の SQL エンジンと機能する単一のビュースキーマです。選択したエンジンからビューをクエリできます。データカタログビューの詳細については、「データカタログビューの作成」を参照してください。

構文

```
CREATE EXTERNAL VIEW schema_name.view_name [ IF NOT EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
AS query_definition;
```

パラメータ

schema_name.view_name

AWS Glue データベースにアタッチされているスキーマ。その後にビューの名前が続きます。

PROTECTED

query_definition 内のクエリが正常に完了した場合にのみ CREATE EXTERNAL VIEW コマンドが完了するように指定します。

IF NOT EXISTS

ビューがまだ存在しない場合、ビューを作成します。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

ビューを作成するときに使用するスキーマの表記法。AWS Glue Data Catalog、作成した Glue データベース、または作成した外部スキーマを使用するように指定できます。詳細については、「CREATE DATABASE」と「CREATE EXTERNAL SCHEMA 」を参照してください。

query_definition

Amazon Redshift がビューを変更するために実行する SQL クエリの定義。

例

次の例では、sample_schema.glue_data_catalog_view という名前のデータカタログビューを作成します。

```
CREATE EXTERNAL PROTECTED VIEW sample_schema.glue_data_catalog_view IF NOT EXISTS
AS SELECT * FROM sample_database.remote_table "remote-table-name";
```

CREATE FUNCTION

SQL SELECT 句または Python プログラムを使用して、新しいスカラーユーザー定義関数 (UDF) を作成します。

詳細な説明と例については、「Amazon Redshift のユーザー定義関数」を参照してください。

必要な権限

CREATE OR REPLACE FUNCTION を実行するには、以下のいずれかの方法によるアクセス許可が必要です。

- CREATE FUNCTION の場合:
  - スーパーユーザーは、関数を作成する際に、信頼された言語と信頼できない言語の両方を使用できます。
  - CREATE [または REPLACE] FUNCTION の権限を持つユーザーは、信頼できる言語による関数の作成が行えます。
- REPLACE FUNCTION の場合:
  - スーパーユーザー
  - CREATE [または REPLACE] FUNCTION の権限を持つユーザー

- 関数の所有者

構文

```
CREATE [ OR REPLACE ] FUNCTION f_function_name
( { [py_arg_name  py_arg_data_type |
sql_arg_data_type } [ , ... ] ] )
RETURNS data_type
{ VOLATILE | STABLE | IMMUTABLE }
AS $$
  { python_program | SELECT_clause }
$$ LANGUAGE { plpythonu | sql }
```

パラメータ

OR REPLACE

その関数の名前、入力引数のデータ型、あるいは署名が既存の関数と同じである場合に既存の関数を置き換えることを指定します。同一のデータタイプセットを定義する新しい関数によってのみ既存の関数を置き換えることができます。関数の置き換えは、スーパーユーザーのみが行うことができます。

すでに存在するの関数と同じ名前と入力引数のデータタイプで、異なる署名の関数を定義する場合は、新しい関数を作成することになります。つまり、関数名はオーバーロードされます。詳細については、「関数名の多重定義」を参照してください。

f_function_name

関数の名前。スキーマ名を指定すると (`myschema.myfunction` など)、指定したスキーマを使用して関数が作成されます。指定しない場合、現在のスキーマに関数が作成されます。有効な名前の詳細については、「名前と識別子」を参照してください。

すべての UDF 名の前に `f_` を付けることをお勧めします。Amazon Redshift は、UDF 名のプレフィックスとして `f_` を予約しており、プレフィックスとして `f_` を使用することで、UDF 名が現在使用されている、または将来使用される Amazon Redshift 組み込み SQL 関数名と競合することを回避できます。詳細については、「UDF 名の競合の回避」を参照してください。

入力引数のデータタイプが異なる場合、同じ関数名で 1 つ以上の関数を定義することができます。つまり、関数名はオーバーロードされます。詳細については、「関数名の多重定義」を参照してください。

py_arg_name py_arg_data_type | sql_arg_data_type

Python UDF の場合、入力引数の名前とデータタイプの一覧。SQL UDF の場合、引数名のないデータタイプの一覧。Python UDF の場合は、引数名を使用して引数を参照します。SQL UDF では、引数の一覧での引数の順序に基づいて、$1、$2 などを使用して引数を参照します。

SQL UDF の場合、入力および戻りデータタイプは、どの標準 Amazon Redshift データタイプでも可能です。Python UDF では、入力データ型および戻りデータ型に SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE PRECISION、BOOLEAN、CHAR、VARCHAR、DATE、または TIMESTAMP を使用できます。さらに、Python ユーザー定義関数 (UDF) はデータ型として ANYELEMENT をサポートしています。このタイプは、実行時に渡される対応する引数のデータタイプに基づいて、標準のデータタイプに自動的に変換されます。複数の引数が ANYELEMENT を使用している場合は、一覧の最初の ANYELEMENT 引数に基づいて、すべてが実行時に同じデータタイプに解決されます。詳細については、「Python UDF データ型」および「データ型」を参照してください。

最大 32 の引数を指定できます。

RETURNS data_type

関数によって返される値のデータ型。RETURNS データ型には、Amazon Redshift のすべての標準データ型を使用できます。また、Python UDF ではデータタイプとして ANYELEMENT を使用できます。このタイプは、実行時に渡される引数に基づいて、標準データタイプに自動的に変換されます。戻り値のデータタイプとして ANYELEMENT を指定する場合は、少なくとも 1 つの引数で ANYELEMENT を使用する必要があります。実際の戻り型のデータタイプは、関数が呼び出された場合に ANYELEMENT 引数から提供されるデータタイプと一致することになります。詳細については、「Python UDF データ型」を参照してください。

VOLATILE | STABLE | IMMUTABLE

関数の変動率についてのクエリオプティマイザを報告します。

関数の最適な変動率の分類を厳正に設定することで、最高の最適化が得られます。ただし、変動幅が厳正すぎると、オプティマイザはその呼び出しを誤って省略してしまい、よって不正確な結果セットを報告することになってしまいます。厳正度の順に、低度の厳正度から変動率を分類すると、以下のようになります。

- VOLATILE
- 安定
- 不変

変動性

同じ引数が入っている場合、単一のステートメントに含まれる行であっても、関数は連続の呼び出しに異なる結果を返すことがあります。クエリオプティマイザは変動的な関数の動作を仮定しないので、変動性の関数を使用するクエリは各入力行につき関数の再評価が必要になります。

安定

同じ引数が入っている場合、関数が単一のステートメント内で処理されるすべての行に対して同じ結果を返すことが保証されます。異なるステートメントから呼び出された場合、関数が異なる結果を返すことがあります。この分類は、単一のステートメント内でのステートメントへの 1 回の呼び出しにおける関数の複数の呼び出しを最適化するオプティマイザを可能にします。

不変

同じ引数が入っている場合、関数は常に永遠に同じ結果を返します。クエリが定数引数の `IMMUTABLE` 関数を呼び出すと、オプティマイザは関数を前評価します。

AS $$ statement $$

実行するステートメントを囲む構造。リテラルキーワードの `AS $$` と `$$` は必須です。

Amazon Redshift では、関数のステートメントをドル引用符という形式を使用して囲む必要があります。囲まれた内容がそのまま渡されます。文字列の内容がそのまま書き込まれるため、特殊文字のエスケープは一切不要です。

ドル引用符付けでは、次の例に示すように、実行するステートメントの開始と終了を 2 つのドル記号のペア ($$) で指定します。

```
$$ my statement $$
```

必要に応じて、各ペアのドル記号間にステートメントの識別に役立つ文字列を指定できます。使用する文字列は、開始と終了の囲い文字のペアで同じにする必要があります。この文字列では大文字と小文字が区別され、ドル記号を含めることができない点を除いては、引用符で囲まれていない識別子と同じ制約事項に従います。次の例では、文字列 `test` を使用しています

```
$test$ my statement $test$
```

ドル引用符付けの詳細については、PostgreSQL ドキュメントの「Lexical StructureDollar」の「Dollar-quoted String Constants」を参照してください。

python_program

値を返す有効で実行可能な Python プログラム。関数とともに渡すステートメントは、Python ウェブサイトの Style Guide for Python Code に示されるインデント要件に準拠する必要があります。詳細については、「UDF のための Python 言語のサポート」を参照してください。

SQL_clause

SQL SELECT 句。

SELECT 句には、以下のタイプの句を含めることはできません。

- FROM
- INTO
- WHERE
- GROUP BY
- ORDER BY
- 制限

言語 { plpythonu | sql }

Python の場合は、`plpythonu`を指定します。SQL の場合は、`sql`を指定します。SQL または plpythonu 用の言語で使用のアクセス権限を持っている必要があります。詳細については、「UDF のセキュリティとアクセス許可」を参照してください。

使用に関する注意事項

ネストされた関数

SQL UDF 内から別の SQL ユーザー定義関数 (UDF) を呼び出すことができます。CREATE FUNCTION コマンドを実行するときは、ネストされた関数が存在している必要があります。Amazon Redshift は UDF の依存関係を追跡しないため、ネストされた関数を削除しても、Amazon Redshift はエラーを返しません。ただし、ネストされた関数が存在しない場合、UDF は失敗します。例えば、次の関数は SELECT 句で `f_sql_greater` 関数を呼び出します。

```
create function f_sql_commission (float, float )
  returns float
stable
as $$
```

```
  select f_sql_greater ($1, $2)
$$ language sql;
```

UDF のセキュリティおよび権限

UDF を作成するには、SQL または plpythonu (Python) 用の言語で使用のアクセス権限を持っている必要があります。デフォルトでは、USAGE ON LANGUAGE SQL は PUBLIC に付与されます。ただし、特定のユーザーやグループに対しては USAGE ON LANGUAGE PLPYTHONU を明示的に付与する必要があります。

SQL の使用を取り消すには、最初に PUBLIC に対して使用を取り消します。次に、SQL UDF の作成を許可された特定のユーザーやグループにのみ、SQL の使用を許可します。次の例では、最初に PUBLIC に対して SQL の使用を取り消し、次にユーザーグループ `udf_devs` に使用を許可します。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

UDF を実行するには、関数ごとに実行許可が付与されている必要があります。デフォルトでは、新しい UDF を実行する権限が PUBLIC に付与されます。使用を制限するには、対象の関数の PUBLIC から実行許可を取り消します。次に、特定の個人またはグループに権限を付与します。

次の例では、PUBLIC から関数 `f_py_greater` の実行許可を取り消し、ユーザーグループ `udf_devs` に使用を許可しています。

```
revoke execute on function f_py_greater(a float, b float) from PUBLIC;
grant execute on function f_py_greater(a float, b float) to group udf_devs;
```

スーパーユーザーは、デフォルトですべての権限を持っています。

詳細については、「GRANT」および「REVOKE」を参照してください。

例

スカラー Python UDF の例

次の例は、2 つの整数を比較し、大きいほうの数値を返す Python UDF を作成する方法を示しています。

```
create function f_py_greater (a float, b float)
```

```
  returns float
stable
as $$
  if a > b:
    return a
  return b
$$ language plpythonu;
```

次の例は、SALES テーブルを検索して新しい `f_py_greater` 関数を呼び出すことによって、COMMISSION または PRICEPAID の 20% のどちらか大きいほうを返します。

```
select f_py_greater (commission, pricepaid*0.20) from sales;
```

スカラー SQL UDF の例

次の例は、2 つの数値を比較し、大きいほうの数値を返す関数を作成する方法を示しています。

```
create function f_sql_greater (float, float)
  returns float
stable
as $$
  select case when $1 > $2 then $1
    else $2
  end
$$ language sql;
```

次のクエリは、新しい `f_sql_greater` 関数を呼び出して SALES テーブルをクエリし、COMMISSION または PRICEPAID の 20% のどちらか大きいほうを返します。

```
select f_sql_greater (commission, pricepaid*0.20) from sales;
```

CREATE GROUP

新しいユーザーグループを定義します。スーパーユーザーのみがグループを作成できます。

構文

```
CREATE GROUP group_name
[ [ WITH ] [ USER username ] [, ...] ]
```

パラメータ

group_name

新しいユーザーグループ名。2 個のアンダースコアで始まるグループ名は Amazon Redshift 内部で使用するために予約されています。有効な名前の詳細については、「名前と識別子」を参照してください。

WiTH

CREATE GROUP の追加のパラメータを指定するオプションの構文。

USER

1 人または複数のユーザーをグループに追加します。

username

グループに追加するユーザーの名前。

例

次の例では、ADMIN1 と ADMIN2 の 2 ユーザー が属する ADMIN_GROUP というユーザーグループを作成します。

```
create group admin_group with user admin1, admin2;
```

ID プロバイダーを作成する

新しい ID プロバイダーを定義します。スーパーユーザーのみが ID プロバイダーを作成できます。

構文

```
CREATE IDENTITY PROVIDER identity_provider_name TYPE type_name
NAMESPACE namespace_name
[PARAMETERS parameter_string]
[APPLICATION_ARN arn]
[IAM_ROLE iam_role]
[AUTO_CREATE_ROLES
    [ TRUE [ { INCLUDE | EXCLUDE } GROUPS LIKE filter_pattern] |
      FALSE
    ]
```

```
];
```

パラメータ

identity_provider_name

ID プロバイダーの名前。有効な名前の詳細については、「名前と識別子」を参照してください。

type_name

インターフェイスとなる ID プロバイダー。Azure は現在、唯一サポートされている ID プロバイダーです。

namespace_name

名前空間。これは、ID プロバイダーディレクトリの一意の簡略化された識別子です。

parameter_string

ID プロバイダーに必要なパラメータと値を含む、適切にフォーマットされた JSON オブジェクトを含む文字列。

arn

IAM アイデンティティセンターのマネージドアプリケーションの Amazon リソースネーム (ARN)。このパラメータは、ID プロバイダーのタイプが AWSIDC である場合にのみ適用されます。

iam_role

IAM アイデンティティセンターに接続するためのアクセス許可を提供する IAM ロール。このパラメータは、ID プロバイダーのタイプが AWSIDC である場合にのみ適用されます。

auto_create_roles

ロールの自動作成機能を有効または無効にします。SQL でオプションが指定されていない場合、デフォルトは `FALSE`、オプションが指定されているが値が指定されていない場合、デフォルトは `TRUE` です。

グループを含めるには、`INCLUDE` を指定します。デフォルトは空です。つまり、`AUTO_CREATES_ROLES` がオンの場合、すべてのグループが含まれます。

グループを除外するには、`EXCLUDE` を指定します。デフォルトは空です。つまり、`AUTO_CREATES_ROLES` がオンの場合、グループは除外されません。

例

次の例では、Microsoft Azure Active Directory (AD) との通信を確立する oauth_standard という名前の ID プロバイダーを TYPE Azure で作成します。

```
CREATE IDENTITY PROVIDER oauth_standard TYPE azure
NAMESPACE 'aad'
PARAMETERS '{"issuer":"https://sts.windows.net/2sdfdsf-d475-420d-b5ac-667adad7c702/",
"client_id":"87f4aa26-78b7-410e-bf29-57b39929ef9a",
"client_secret":"BUAH~ewrqewrqwerUUY^%tHe1oNZShoiU7",
"audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift"]
}'
```

IAM アイデンティティセンターのマネージドアプリケーションは、既存のプロビジョニングされたクラスターまたは Amazon Redshift Serverless ワークグループに接続できます。これにより、IAM アイデンティティセンターを通じて Redshift データベースへのアクセスを管理できます。これを行うには、次の例に示すような SQL コマンドを実行します。データベース管理者である必要があります。

```
CREATE IDENTITY PROVIDER "redshift-idc-app" TYPE AWSIDC
NAMESPACE 'awsidc'
APPLICATION_ARN 'arn:aws:sso::123456789012:application/ssoins-12345f67fe123d4/apl-
a0b0a12dc123b1a4'
IAM_ROLE 'arn:aws:iam::123456789012:role/MyRedshiftRole';
```

この例のアプリケーション ARN は、接続先のマネージドアプリケーションを識別します。これは、`SELECT * FROM SVV_IDENTITY_PROVIDERS;` を実行することで確認できます。

その他の例を含め、CREATE IDENTITY PROVIDER の使用方法の詳細については、「Amazon Redshift 用のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。Redshift から IAM アイデンティティセンターへの接続の設定の詳細については、「Redshift を IAM アイデンティティセンターに接続してユーザーにシングルサインオンエクスペリエンスを提供する」を参照してください。

ライブラリを作成する

CREATE FUNCTION コマンドのユーザー定義関数 (UDF) を作成するときに、ユーザーは Python ライブラリをインストールすることができます。ユーザーがインストールするライブラリの合計サイズは 100 MB を超えられません。

CREATE LIBRARY は、トランザクションブロック内で実行することはできません (BEGIN … END)。トランザクションの詳細については、「直列化可能分離」を参照してください。

Amazon Redshift は Python バージョン 2.7 をサポートしています。詳細については、www.python.org を参照してください。

詳細については、「例: カスタム Python ライブラリモジュールのインポート」を参照してください。

必要な権限

CREATE LIBRARY に必要な権限を以下に示します。

- スーパーユーザー
- CREATE LIBRARY の権限を持つユーザー、または指定した言語の権限を持つユーザー

構文

```
CREATE [ OR REPLACE ] LIBRARY library_name LANGUAGE plpythonu
FROM
{ 'https://file_url'
| 's3://bucketname/file_name'
authorization
  [ REGION [AS] 'aws_region']
  IAM_ROLE { default | ‘arn:aws:iam::<AWS #####-id>:role/<role-name>’ }
}
```

パラメータ

OR REPLACE

このライブラリと同じ名前のライブラリがすでに存在する場合、既存のライブラリが置き換えられることを指定します。REPLACE は即座にコミットされます。ライブラリに依存する UDF が同時に実行されている場合、UDF がトランザクション内で実行されていても、UDF は失敗するか、予期しない結果を返す場合があります。ライブラリを置き換えるには、所有者またはスーパーユーザーである必要があります。

library_name

インストールされるライブラリの名前。Python 標準ライブラリモジュール、またはインストール済みの Amazon Redshift Python モジュールと同じ名前のモジュールを含むライブラリを作成す

ることはできません。ユーザーがインストールした既存のライブラリがインストールするライブラリと同じ Python パッケージを使用している場合は、新しいライブラリをインストールする前に既存のライブラリを削除する必要があります。詳細については、「UDF のための Python 言語のサポート」を参照してください。

plpythonu 言語

使用される言語。サポートされている言語は Python (plpythonu) のみです。Amazon Redshift は Python バージョン 2.7 をサポートしています。詳細については、www.python.org を参照してください。

FROM

ライブラリファイルの場所。Amazon S3 バケットとオブジェクト名を指定できます。また、公開ウェブサイトからファイルをダウンロードする URL を指定できます。ライブラリは `.zip` ファイルの形式でパッケージ化される必要があります。詳細については、Python ドキュメントから、Python モジュールの構築とインストールを参照してください。

https://file_url

公開ウェブサイトからファイルをダウンロードする URL。URL には 3 つまでのリダイレクトを含めることができます。次は URL ファイルの例です。

```
'https://www.example.com/pylib.zip'
```

s3://bucket_name/file_name

ライブラリファイルを含む単一の Amazon S3 オブジェクトのパス。以下は、Amazon S3 オブジェクトパスの例です。

```
's3://amzn-s3-demo-bucket/my-pylib.zip'
```

Amazon S3 バケットを指定する場合、ファイルをダウンロードする権限を持つ AWS のユーザーの、認証情報を指定する必要もあります。

> **Important**
>
> Amazon S3 バケットが Amazon Redshift クラスターと同じ AWS リージョンに存在しない場合は、REGION オプションを使用して、データが置かれている AWS リージョンを指定する必要があります。aws_region の値は、COPY コマンドの REGION パラメータの説明に示されている AWS リージョンと一致する必要があります。

authorization

ライブラリファイルを含む Amazon S3 バケットへアクセスするための認証と認可にクラスターが使用する方法を示す句です。クラスターは、LIST と GET のアクションにより Amazon S3 にアクセスするためのアクセス許可が必要です。

認可の構文は COPY コマンドの認可の構文と同じです。詳細については、「認可パラメータ」を参照してください。

```
IAM_ROLE { default | ‘arn:aws:iam::<AWS #####-id>:role/<role-name>’
```

デフォルトキーワードを使用して、CREATE LIBRARY コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。IAM_ROLE を指定すると、ACCESS_KEY_ID および SECRET_ACCESS_KEY、SESSION_TOKEN、または CREDENTIALS は使用できません。

必要に応じて、Amazon S3 バケットがサーバー側の暗号化を使用する場合は、credentials-args 文字列の暗号化キーを指定します。一時的なセキュリティ認証情報を使う場合は、credentials-args 文字列の一時トークンを指定します。

詳細については、「一時的な認証情報」を参照してください。

REGION [AS] aws_region

Amazon S3 バケットがある AWS リージョン。REGION は、Amazon S3 バケットが Amazon Redshift クラスターと同じ AWS リージョンにない場合に必須です。aws_region の値は、COPY コマンドの REGION パラメータの説明に示されている AWS リージョンと一致する必要があります。

CREATE LIBRARY のデフォルトでは、Amazon S3 バケットが Amazon Redshift クラスターと同じ AWS リージョンにあると見なします。

例

次の 2 つの例では、`urlparse3-1.0.3.zip`という名前のファイルにパッケージ化されている urlparse Python モジュールをインストールします。

次のコマンドは、米国東部リージョンにある Amazon S3 バケットに対しアップロードされたパッケージから、`f_urlparse`という名前の UDF ライブラリをインストールします。

```
create library f_urlparse
language plpythonu
from 's3://amzn-s3-demo-bucket/urlparse3-1.0.3.zip'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-id>:role/<role-name>'
region as 'us-east-1';
```

次の例では、ウェブサイトのライブラリから `f_urlparse` 名のライブラリをインストールします。

```
create library f_urlparse
language plpythonu
from 'https://example.com/packages/urlparse3-1.0.3.zip';
```

CREATE MASKING POLICY

新しい動的データマスキングポリシーを作成して、指定した形式のデータを難読化します。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、マスキングポリシーを作成できます。

構文

```
CREATE MASKING POLICY
   policy_name [IF NOT EXISTS]
   WITH (input_columns)
   USING (masking_expression);
```

パラメータ

policy_name

マスキングポリシーの名前。マスキングポリシーには、データベースに既に存在する別のマスキングポリシーと同じ名前を付けることはできません。

input_columns

(col1 データ型、col2 データ型...) 形式の列名のタプル。

列名はマスキング式の入力として使用されます。列名はマスクされる列の名前と一致する必要はありませんが、入力と出力のデータ型は一致する必要があります。

masking_expression

ターゲット列の変換に使用される SQL 式。文字列操作関数などのデータ操作関数を使用して記述することも、SQL、Python、または AWS Lambda で記述されたユーザー定義関数と組み合わせて記述することもできます。マスキングポリシーに複数の出力がある場合は、列式のタプルを含めることができます。マスク式として定数を使用する場合は、入力型と一致する型に明示的にキャストする必要があります。

マスキング式で使用するユーザー定義関数には USAGE アクセス許可が必要です。

CREATE MATERIALIZED VIEW

1 つ以上の Amazon Redshift テーブルに基づいてマテリアライズドビューを作成します。また、Spectrum やフェデレーションクエリを使用して作成した外部テーブルに基づいてマテリアライズドビューを作成することもできます。Spectrum の詳細については、「Amazon Redshift Spectrum」を参照してください。フェデレーションクエリの詳細については、「Amazon Redshift での横串検索を使用したデータのクエリの実行」を参照してください。

構文

```
CREATE MATERIALIZED VIEW mv_name
[ BACKUP { YES | NO } ]
[ table_attributes ]
[ AUTO REFRESH { YES | NO } ]
AS query
```

パラメータ

BACKUP

マテリアライズドビューを自動および手動クラスタースナップショットに含めるかどうかを指定する句。

重要なデータを含まないマテリアライズドビューについては、スナップショットの作成やスナップショットからの復元にかかる時間を節約し、Amazon Simple Storage Service のストレージスペースを節約するため、BACKUP NO を指定します。BACKUP NO の設定は、クラスター内の別ノードへのデータの自動レプリケーションには影響しません。そのため、BACKUP NO が指定されたマテリアライズドビューはノードの障害時に回復されます。デフォルトは BACKUP YES です。

table_attributes

マテリアライズドビュー内のデータの分散方法を指定する句。これには、以下が含まれます。

- `DISTSTYLE { EVEN | ALL | KEY }` の形式のマテリアライズドビュー向けの分散スタイル。この句を省略すると、分散スタイルは `EVEN` になります。詳細については、「分散スタイル」を参照してください。
- `DISTKEY ( distkey_identifier )` の形式のマテリアライズドビュー向けの分散キー。詳細については、「分散スタイルの指定」を参照してください。
- `SORTKEY ( column_name [, ...] )` の形式のマテリアライズドビュー向けのソートキー。詳細については、「ソートキー」を参照してください。

AS query

マテリアライズドビューとその内容を定義する有効な `SELECT` ステートメント。クエリからの結果は、マテリアライズドビューの列および行を定義します。マテリアライズドビューの作成時の制約事項については、「制約事項」を参照してください。

また、クエリで使用する特定の SQL 言語の構造によって、マテリアライズドビューを増分更新できるかフル更新できるかが決まります。更新方法の詳細については、「REFRESH MATERIALIZED VIEW」を参照してください。増分更新の制約事項については、「増分更新の制約事項」を参照してください。

クエリに増分更新をサポートしていない SQL コマンドが含まれている場合、Amazon Redshift ではマテリアライズドビューがフル更新を使用することを示すメッセージが表示されます。このメッセージは、SQL クライアントアプリケーションによって表示される場合と表示されない場合があります。マテリアライズドビューで使用されている更新のタイプについては、STV_MV_INFOの `state` 列を確認してください。

AUTO REFRESH

マテリアライズドビューを、そのベーステーブルからの最新の変更で自動的に更新する必要があるかどうかを定義する句。デフォルト値は `NO` です。詳細については、「マテリアライズドビューの更新」を参照してください。

使用に関する注意事項

マテリアライズドビューを作成するには、次の権限が必要です。

- スキーマの CREATE 権限。

- マテリアライズドビューを作成するためのベーステーブルに対するテーブルレベルまたは列レベルの SELECT 権限。特定の列に対する列レベルの権限がある場合は、それらの列にのみマテリアライズドビューを作成することができます。

データ共有内のマテリアライズドビューの増分更新

Amazon Redshift は、ベーステーブルを共有している場合、コンシューマーデータ共有でのマテリアライズドビューの自動更新と増分更新をサポートしています。増分更新は、Amazon Redshift が前回の更新後に発生したベーステーブルの変更を特定し、マテリアライズドビューの対応するレコードのみを更新する操作です。これにより、フル更新と比べて、実行が迅速化し、ワークロードのパフォーマンスが向上します。増分更新を利用するために、マテリアライズドビュー定義を変更する必要はありません。

マテリアライズドビューによる増分更新を利用する場合、次の 2 つの注意すべき制限があります。

- マテリアライズドビューは、ローカルまたはリモートの 1 つのデータベースのみを参照する必要があります。
- 増分更新は、新しいマテリアライズドビューでのみ使用できます。したがって、増分更新を行うには、既存のマテリアライズドビューを削除して再作成する必要があります。

データ共有でのマテリアライズドビューの作成の詳細については、「Amazon Redshift データ共有でのビューの使用」を参照してください。これには、いくつかのクエリ例も含まれています。

マテリアライズドビューまたはベーステーブルの DDL の更新

Amazon Redshift でマテリアライズドビューを使用する場合は、マテリアライズドビューまたはベーステーブルのデータ定義言語 (DDL) の更新に関する以下の注意事項に従ってください。

- ベーステーブルを参照するマテリアライズドビューに影響を与えることなく、ベーステーブルに列を追加できます。
- 操作によっては、マテリアライズドビューをまったく更新できない状態になる場合があります。該当する操作として、名前の変更、列の削除、列の種類の変更、スキーマ名の変更などがあります。このようなオペレーションが行われたマテリアライズドビューは、クエリできますが更新できません。この場合、マテリアライズドビューを削除または再作成する必要があります。
- 一般的に、マテリアライズドビューの定義 (SQL ステートメント) は変更できません。
- マテリアライズドビューの名前は変更できません。

制約事項

次のものを参照する、または含むマテリアライズドビューは定義できません。

- 標準ビュー、またはシステムテーブルとビュー。
- 一時テーブル。
- ユーザー定義関数。
- ORDER BY 句、LIMIT 句、OFFSET 句。
- ベーステーブルの遅延バインディングの参照。つまり、マテリアライズドビューを定義する SQL クエリで参照されるベーステーブルや関連列は存在し、有効である必要があります。
- リーダーノードのみの関数: CURRENT_SCHEMA、CURRENT_SCHEMAS、HAS_DATABASE_PRIVILEGE、HAS_SCHEMA_PRIVIL
- マテリアライズドビューの定義の中に、変更可能な関数または外部スキーマが含まれている場合、AUTO REFRESH YES オプションは使用できません。また、別のマテリアライズドビューに基づいてマテリアライズドビューを定義する場合にも使用できません。
- マテリアライズドビューで ANALYZE を手動で実行する必要はありません。これは現在のところ、AUTO ANALYZE でのみ発生しています。詳細については、「テーブルを分析する」を参照してください。

例

次の例では、結合および集計された 3 つのベーステーブルからマテリアライズドビューを作成します。各行は、カテゴリと販売されたチケット数を表します。tickets_mv マテリアライズドビューのクエリを実行すると、tickets_mv マテリアライズドビューの計算済みのデータに直接アクセスします。

```
CREATE MATERIALIZED VIEW tickets_mv AS
    select   catgroup,
    sum(qtysold) as sold
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;
```

以下の例は、前の例と同様のマテリアライズドビューを作成し、集計関数 MAX() を使用します。

```
CREATE MATERIALIZED VIEW tickets_mv_max AS
    select   catgroup,
    max(qtysold) as sold
```

```
    from     category c, event e, sales s
    where    c.catid = e.catid
    and      e.eventid = s.eventid
    group by catgroup;

SELECT name, state FROM STV_MV_INFO;
```

次の例では、UNION ALL 句を使用して Amazon Redshift `public_sales` テーブルと Redshift Spectrum `spectrum.sales` テーブルを結合し、`mv_sales_vw` マテリアルビューを作成します。Amazon Redshift Spectrum の CREATE EXTERNAL TABLE コマンドの詳細については、CREATE EXTERNAL TABLE を参照してください。Redshift Spectrum 外部テーブルは、Amazon S3 のデータを参照します。

```
CREATE MATERIALIZED VIEW mv_sales_vw as
select salesid, qtysold, pricepaid, commission, saletime from public.sales
union all
select salesid, qtysold, pricepaid, commission, saletime from spectrum.sales
```

次の例では、横串検索の外部テーブルに基づいてマテリアライズドビュー mv_fq を作成します。フェデレーションクエリの詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

```
CREATE MATERIALIZED VIEW mv_fq as select firstname, lastname from apg.mv_fq_example;

select firstname, lastname from mv_fq;
 firstname | lastname
-----------+----------
 John      | Day
 Jane      | Doe
(2 rows)
```

次の例は、マテリアライズドビューの定義を示しています。

```
SELECT pg_catalog.pg_get_viewdef('mv_sales_vw'::regclass::oid, true);

pg_get_viewdef
---------------------------------------------------
create materialized view mv_sales_vw as select a from t;
```

次のサンプルは、マテリアライズドビューの定義で AUTO REFRESH を設定する方法を示しています。また DISTSTYLE を指定しています。まず、簡単なベーステーブルを作成します。

```
CREATE TABLE baseball_table (ball int, bat int);
```

次に、マテリアライズドビューを作成します。

```
CREATE MATERIALIZED VIEW mv_baseball DISTSTYLE ALL AUTO REFRESH YES AS SELECT ball AS
 baseball FROM baseball_table;
```

これで mv_baseball マテリアライズドビューをクエリできるようになりました。マテリアライズドビューの AUTO REFRESH がオンになっているかどうかを確認するには、「STV_MV_INFO」を参照してください。

次のサンプルでは、別のデータベースのソーステーブルを参照するマテリアライズドビューを作成します。ソーステーブル database_A を含むデータベースが、database_B で作成したマテリアライズドビューと同じクラスターまたはワークグループにあることを前提としています (サンプルの代わりに独自のデータベースを使用できます)。まず、database_A に、cityname 列がある、cities という名前のテーブルを作成します。列のデータ型を VARCHAR にします。ソーステーブルを作成したら、database_B で次のコマンドを実行して、cities テーブルをソースとするマテリアライズドビューを作成します。FROM 句に必ずソーステーブルのデータベースとスキーマを指定してください。

```
CREATE MATERIALIZED VIEW cities_mv AS
SELECT  cityname
FROM    database_A.public.cities;
```

作成したマテリアライズドビューをクエリします。クエリは、元のソースが database_A の cities テーブルであるレコードを取得します。

```
select * from cities_mv;
```

SELECT ステートメントを実行すると、cities_mv はレコードを返します。REFRESH ステートメントが実行されたときにのみ、レコードはソーステーブルから更新されます。また、マテリアライズドビューでレコードを直接更新できないことに注意してください。マテリアライズドビューのデータを更新する方法については、REFRESH MATERIALIZED VIEW を参照してください。

マテリアライズドビューの概要およびマテリアライズドビューの更新や削除に使用する SQL コマンドの詳細については、以下のトピックを参照してください。

- Amazon Redshift でのマテリアライズドビュー
- REFRESH MATERIALIZED VIEW

- DROP MATERIALIZED VIEW

モデルを作成する

トピック

- 前提条件
- 必要な権限
- コスト管理
- フル CREATE MODEL
- パラメータ
- 使用に関する注意事項
- ユースケース

前提条件

CREATE MODEL ステートメントを使用する前に、Amazon Redshift ML を使用するためのクラスターの設定 の前提条件を満たしてください。前提条件の概要は次のとおりです。

- AWS マネジメントコンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Amazon Redshift クラスターを作成します。
- クラスターの作成中に AWS Identity and Access Management (IAM) ポリシーをアタッチします。
- Amazon Redshift と SageMaker が、他のサービスとやり取りするロールを引き受けることを許可するには、IAM ロールに適切な信頼ポリシーを追加します。

IAM ロール、信頼ポリシー、およびその他の前提条件の詳細については、「Amazon Redshift ML を使用するためのクラスターの設定」を参照してください。

以下では、CREATE MODEL ステートメントのさまざまなユースケースを見つけることができます。

- 単純な CREATE MODEL
- ユーザーガイダンス付きの CREATE MODEL
- AUTO OFF 付きの CREATE XGBoost モデル
- 独自のモデルを持参 (BYOM) - ローカル推論

- 独自のモデルを持参 (BYOM) - リモート推論
- K-MEANS を使用した CREATE MODEL
- フル CREATE MODEL

必要な権限

CREATE MODEL に必要な権限を以下に示します。

- スーパーユーザー
- CREATE MODEL の権限を持つユーザー
- GRANT CREATE MODEL の権限を持つロール

コスト管理

Amazon Redshift 機械学習は既存のクラスターリソースを使用して予測モデルを作成するため、追加料金は発生しません。ただし、クラスターのサイズを変更する必要がある場合、またはモデルをトレーニングする場合は、追加料金が発生する可能性があります。Amazon Redshift 機械学習は、モデルのトレーニングに Amazon SageMaker を使用します。これには追加費用がかかります。トレーニングにかかる最大時間を制限したり、モデルのトレーニングに使用するトレーニング例の数を制限したりするなど、追加料金を管理する方法があります。詳細については、「Amazon Redshift 機械学習を使用するためのコスト」を参照してください。

フル CREATE MODEL

以下に、完全な CREATE MODEL 構文の基本的なオプションをまとめます。

フル CREATE MODEL 構文

以下は、CREATE MODEL ステートメントの完全な構文です。

> **Important**
>
> CREATE MODEL ステートメントを使用してモデルを作成する場合は、次の構文内のキーワードの順序に従います。

```
CREATE MODEL model_name
FROM { table_name | ( select_statement )  | 'job_name' }
```

```
[ TARGET column_name ]
FUNCTION function_name [ ( data_type [, ...] ) ]
[ RETURNS data_type ]
  -- supported only for BYOM
[ SAGEMAKER 'endpoint_name'[:'model_name']]
  -- supported only for BYOM remote inference
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
[ AUTO ON / OFF ]
  -- default is AUTO ON
[ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST } ]
  -- not required for non AUTO OFF case, default is the list of all supported types
  -- required for AUTO OFF
[ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
  -- not supported when AUTO OFF
[ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1_Macro' | 'AUC' |
             'reg:squarederror' | 'reg:squaredlogerror'| 'reg:logistic'|
             'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' |
 'binary:hinge',
             'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' |
 'AverageWeightedQuantileLoss' ) ]
  -- for AUTO ON: first 5 are valid
  -- for AUTO OFF: 6-13 are valid
  -- for FORECAST: 14-18 are valid
[ PREPROCESSORS 'string' ]
  -- required for AUTO OFF, when it has to be 'none'
  -- optional for AUTO ON
[ HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT ( Key 'value' (,...) ) } ]
  -- support XGBoost hyperparameters, except OBJECTIVE
  -- required and only allowed for AUTO OFF
  -- default NUM_ROUND is 100
  -- NUM_CLASS is required if objective is multi:softmax (only possible for AUTO OFF)
 [ SETTINGS (
   S3_BUCKET 'amzn-s3-demo-bucket',  |
    -- required
  TAGS 'string', |
    -- optional
  KMS_KEY_ID 'kms_string', |
    -- optional
  S3_GARBAGE_COLLECT on / off, |
    -- optional, defualt is on.
  MAX_CELLS integer, |
    -- optional, default is 1,000,000
  MAX_RUNTIME integer (, ...) |
    -- optional, default is 5400 (1.5 hours)
```

```
  HORIZON integer, |
    -- required if creating a forecast model
  FREQUENCY integer, |
    -- required if creating a forecast model
  PERCENTILES string, |
    -- optional if creating a forecast model
  MAX_BATCH_ROWS integer -- optional for BYOM remote inference
    ) ]
```

パラメータ

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

FROM { table_name | (select_query) | 'job_name'}

table_name またはトレーニングデータを指定するクエリ。これらは、システム内の既存のテーブル、または丸括弧で囲まれた Amazon RedShift 互換の SELECT クエリ、つまり () のいずれかです。クエリ結果には少なくとも 2 つの列が必要です。

TARGET column_name

予測対象となる列の名前。列は、FROM 句内に存在する必要があります。

FUNCTION function_name (data_type [, ...])

作成する関数の名前と、入力引数のデータ型。関数名の代わりに、データベース内のスキーマのスキーマ名を指定できます。

RETURNS data_type

モデルの関数から返されるデータ型。返される SUPER データ型はリモート推論を使用する BYOM モデルにのみ適用されます。

SAGEMAKER 'endpoint_name'[:'model_name']

Amazon SageMaker エンドポイントの名前。エンドポイント名がマルチモデルのエンドポイントを指している場合は、使用するモデルの名前を追加します。エンドポイントは、Amazon Redshift クラスターと同じ AWS リージョン 内でホストされる必要があります。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

デフォルトキーワードを使用して、CREATE MODEL コマンドの実行時にデフォルトとして設定され、同時にクラスターに関連付けられた IAM ロールを、Amazon Redshift が使用するようにします。または、IAM ロールの ARN を指定して、そのロールを使用することもできます。

[AUTO ON / OFF]

プリプロセッサ、アルゴリズム、およびハイパーパラメータの選択での、CREATE MODEL による自動検出をオンまたはオフにします。予測モデルを作成するときに on を指定すると、AutoPredictor を使用することを示します。ここで、Amazon Forecast は、データセット内の各時系列に最適なアルゴリズムの組み合わせを適用します。

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER | KMEANS | FORECAST }

(オプション) モデルタイプを指定します。XGBoost、多層パーセプトロン (MLP)、KMEANS、線形学習など (すべては Amazon SageMaker Autopilot でサポートされているアルゴリズム)、特定のモデルタイプのモデルをトレーニングするかを指定できます。パラメータを指定しない場合、トレーニング中にサポートされているすべてのモデルタイプが最適なモデルを検索します。Redshift ML で予測モデルを作成して、正確な時系列予測を作成することもできます。

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

(オプション) 問題の種類を指定します。問題の種類がわかっている場合は、Amazon Redshift をその特定のモデルタイプの最適なモデルだけを検索するように制限できます。このパラメータを指定しない場合、トレーニング中にデータに基づく問題の種類が検出されます。

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC' | 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' | 'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge' | 'multi:softmax' | 'RMSE' | 'WAPE' | 'MAPE' | 'MASE' | 'AverageWeightedQuantileLoss')

(オプション) 機械学習システムの予測品質を測定するために使用する目標メトリクスの名前を指定します。このメトリクスは、トレーニング中に最適化され、データからモデルパラメータ値の最良の推定値を提供します。メトリクスを明示的に指定しない場合、デフォルトの動作では、MSE が回帰に、F1 がバイナリ分類に、精度がマルチクラス分類に自動的に使用されます。目標の詳細については、Amazon SageMaker API リファレンスの「AutoMLJobObjective」および XGBOOST ドキュメントの「Learning task parameters」を参照してください。RMSE、WAPE、MAPE、MASE、および AverageWeightedQuantileLoss の値は、予測モデルにのみ適用されます。詳細については、CreateAutoPredictor API オペレーションを参照してください。

PREPROCESSORS 'string'

(オプション) 特定の列セットに対するプリプロセッサの特定の組み合わせを指定します。形式は、columnSets のリストであり、各列のセットに適用される適切な変換です。Amazon Redshift は、特定のトランスフォーマのリスト内にあるすべてのトランスフォーマを対応する ColumnSet

内のすべての列に適用します。例えば、Imputer を使用した OneHotEncoder を列 t1 と t2 に適用するには、次のサンプルコマンドを使用します。

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
...
  {"ColumnSet": [
      "t1",
      "t2"
    ],
    "Transformers": [
      "OneHotEncoder",
      "Imputer"
    ]
  },
  {"ColumnSet": [
      "t3"
    ],
    "Transformers": [
      "OneHotEncoder"
    ]
  },
  {"ColumnSet": [
      "temp"
    ],
    "Transformers": [
      "Imputer",
      "NumericPassthrough"
    ]
  }
]'
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket'
)
```

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

デフォルトの XGBoost パラメータを使用するか、それをユーザー指定の値で上書きするかを指定します。値は一重引用符で囲む必要があります。以下に、XGBoost のパラメータとそのデフォルトの例を示します。

パラメータ名	パラメータ値	デフォルト値	コメント
num_class	整数	マルチクラス分類に必須です。	該当なし
num_round	整数	100	該当なし
tree_method	文字列	Auto	該当なし
max_depth	整数	6	[0 , 10]
min_child_weight	浮動小数点	1	MinValue: 0、MaxValue: 120
subsample	浮動小数点	1	MinValue: 0.5、MaxValue: 1
gamma	浮動小数点	0	MinValue: 0、MaxValue: 5
alpha	浮動小数点	0	MinValue: 0、MaxValue: 1000
eta	浮動小数点	0.3	MinValue: 0.1、MaxValue: 0.5
colsample_byleve	浮動小数点	1	MinValue: 0.1、MaxValue: 1

パラメータ名	パラメータ値	デフォルト値	コメント
colsample_bynode	浮動小数点	1	MinValue: 0.1、MaxValue: 1
colsample_bytree	浮動小数点	1	MinValue: 0.5、MaxValue: 1
lambda	浮動小数点	1	MinValue: 0、MaxValue: 1000
max_delta_step	整数	0	[0, 10]

SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket', | TAGS 'string', | KMS_KEY_ID 'kms_string' , | S3_GARBAGE_COLLECT on / off, | MAX_CELLS integer , | MAX_RUNTIME (,...) , | HORIZON integer, | FREQUENCY forecast_frequency, | PERCENTILES array of strings)

S3_BUCKET 句は、中間結果の保存に使用される Amazon S3 の場所を指定します。

(オプション) TAGS パラメータは、キーと値のペアをカンマで区切ったリストで、Amazon SageMaker および Amazon Forecast で作成されたリソースにタグを付けるために使用できます。タグを使用すると、リソースの整理やコストの割り当てに役立ちます。ペアの値はオプションであるため、`key=value` 形式を使用するか、単にキーを作成するだけで、タグを作成できます。Amazon Redshift のタグの詳細については、「タグ付けの概要」を参照してください。

(オプション) KMS_KEY_ID は、Amazon Redshift が AWS KMS キーを使用したサーバー側の暗号化を使用して、保管中のデータを保護するかどうかを指定します。転送中のデータは Secure Sockets Layer (SSL) で保護されています。

(オプション) S3_GARBAGE_COLLECT { ON | OFF } は、Amazon Redshift がモデルのトレーニングに使用される結果のデータセットに対してガベージコレクションを実行するかどうかを指定します。OFF に設定すると、モデルとモデルのトレーニングに使用される結果のデータセットは Amazon S3 に残り、他の目的に使用できます。ON に設定すると、Amazon Redshift はトレーニングの完了後に Amazon S3 内のアーティファクトを削除します。デフォルトはオンです。

(オプション) MAX_CELLS は、トレーニングデータのセル数を指定します。この値は、(トレーニングクエリまたはテーブル内の) レコード数と列数の積です。デフォルトは 1,000,000 です。

(オプション) MAX_RUNTIME は、トレーニングする最大時間を指定します。トレーニングジョブは、データセットのサイズに応じてより早く完了することがよくあります。これは、トレーニングにかかる最大時間を指定します。デフォルトは 5,400 (90 分) です。

HORIZON は、予測モデルが返すことができる予測の最大数を指定します。モデルのトレーニングが完了すると、この整数は変更できません。このパラメータは、予測モデルをトレーニングする場合に必要です。

FREQUENCY は、予測の詳細度を時間単位で指定します。使用できるオプションは、`Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min` です。このパラメータは、予測モデルをトレーニングする場合に必要です。

(オプション) PERCENTILES は、予測子のトレーニングに使用される予測タイプを指定するカンマ区切り文字列です。予測タイプは、0.01 以上の増分で 0.01 から 0.99 までの分位数にすることができます。mean で平均予測を指定することもできます。最大 5 つの予測タイプを指定できます。

MAX_BATCH_ROWS 整数

(オプション) Amazon Redshift が 1 回の SageMaker 呼び出しに対して 1 回のバッチリクエストで送信する最大行数。リモート推論を使用する BYOM でのみサポートされます。このパラメータの最小値は 1 です。最大値は `INT_MAX` または 2,147,483,647 です。このパラメータは、入力データ型と返されたデータ型の両方が SUPER である場合にのみ必要です。デフォルト値は `INT_MAX` または 2,147,483,647 です。

使用に関する注意事項

CREATE MODEL を使用するときは、次の点を考慮してください。

- CREATE MODEL ステートメントは非同期モードで動作し、Amazon S3 へのトレーニングデータのエクスポート時に戻ります。Amazon SageMaker でのトレーニングの残りステップは、バックグラウンドで行われます。トレーニングが進行している間は、対応する推論関数が表示されますが、実行することはできません。STV_ML_MODEL_INFO にクエリを実行して、トレーニングの状態を確認できます。
- トレーニングはバックグラウンドで最大 90 分間実行できますが、デフォルトでは自動モデルで実行し、延長することもできます。DROP MODEL コマンドを実行するだけで、トレーニングをキャンセルできます。

- モデルの作成に使用する Amazon Redshift クラスターと、トレーニングデータとモデルアーティファクトのステージングに使用される Amazon S3 バケットは、同じ AWS リージョンに置かれている必要があります。
- モデルトレーニング中に、Amazon Redshift と SageMaker は、指定した Amazon S3 バケットに中間成果物を保存します。Amazon Redshift のデフォルトでは、CREATE MODEL オペレーションの最後にガベージコレクションが実行されます。Amazon Redshift は、Amazon S3 からこれらのオブジェクトを削除します。これらのアーティファクトを Amazon S3 で保持するには、S3_GARBAGE COLLECT OFF オプションを設定します。
- FROM 句で提供されるトレーニングデータには、少なくとも 500 行を使用する必要があります。
- CREATE MODEL ステートメントを使用する場合、FROM { table_name | (select_query) } 句には、最大 256 個の特徴 (入力) 列しか指定できません。
- AUTO ON の場合、トレーニングセットとして使用できる列タイプは、SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE、BOOLEAN、CHAR、VARCHAR、TIMESTAMP、TIMESTAMPTZ です。AUTO OFF の場合、トレーニングセットとして使用できる列タイプは、SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、DOUBLE、BOOLEAN です。
- ターゲット列のデータ型として、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、TIMESTAMPTZ、GEOMETRY、GEOGRAPHY または VARBYTE を使用することはできません。
- モデルの精度を上げるには、以下のいずれかを実行します。
  - FROM 句でトレーニングデータを指定するときに、CREATE MODEL コマンドに関連する列をできるだけ多く追加します。
  - MAX_RUNTIME と MAX_CELLS にはより大きな値を使用してください。このパラメータの値を大きくすると、モデルのトレーニングにかかるコストが増加します。
- トレーニングデータが計算され、Amazon S3 バケットにエクスポートされるとすぐに CREATE MODEL ステートメントの実行が返されます。その後、SHOW MODEL コマンドを使用して、トレーニングのステータスを確認できます。バックグラウンドでトレーニングされているモデルに障害が発生した場合は、SHOW MODEL を使用してエラーを確認できます。失敗したモデルを再試行することはできません。DROP MODEL を使用すると、失敗したモデルを削除し、新しいモデルを再作成できます。SHOW MODEL の詳細については、「SHOW MODEL」を参照してください。
- ローカル BYOM は、Amazon Redshift ML が BYOM 以外の場合にサポートするのと同じ種類のモデルをサポートします。Amazon Redshift は、プリプロセッサを使用しないプレーンな XGBoost (XGBoost バージョン 1.0 以降を使用)、KMEANS モデルをサポートし、Amazon SageMaker Autopilot によってトレーニングされた XGBoost/MLP/線形学習モデルをサポートします。Autopilot

が指定したプリプロセッサで後者をサポートします。プリプロセッサは Amazon SageMaker Neo でもサポートされています。

- Amazon Redshift クラスターで Virtual Private Cloud (VPC) の拡張ルーティングが有効になっている場合は、クラスターがある VPC に対して Amazon S3 VPC エンドポイントと SageMaker VPC エンドポイントを作成してください。これにより、CREATE MODEL の実行中にトラフィックがこれらのサービス間で VPC を通過できるようになります。詳細については、「SageMaker Clarify Job Amazon VPC Subnets and Security Groups」を参照してください。

ユースケース

次のユースケースは、ニーズに合わせて CREATE MODEL を使用する方法を示しています。

単純な CREATE MODEL

次に、CREATE MODEL 構文の基本的なオプションをまとめます。

単純な CREATE MODEL 構文

```
CREATE MODEL model_name
FROM { table_name | ( select_query ) }
TARGET column_name
FUNCTION prediction_function_name
IAM_ROLE { default }
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  [ MAX_CELLS integer ]
)
```

単純な CREATE MODEL パラメータ

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

FROM { table_name | (select_query) }

table_name またはトレーニングデータを指定するクエリ。これらは、システム内の既存のテーブル、または丸括弧で囲まれた Amazon RedShift 互換の SELECT クエリ、つまり () のいずれかです。クエリ結果には少なくとも 2 つの列が必要です。

TARGET column_name

予測対象となる列の名前。列は、FROM 句内に存在する必要があります。

FUNCTION prediction_function_name

CREATE MODEL によって生成され、このモデルを使用して予測を行うために使用される Amazon Redshift 機械学習関数の名前を指定する値。この関数は、モデルオブジェクトと同じスキーマで作成され、オーバーロードされる可能性があります。

Amazon Redshift の機械学習では、回帰および分類用の Xtreme Gradient Boosted ツリー (XgBoost) モデルなどのモデルがサポートされています。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

デフォルトキーワードを使用して、CREAT MODEL コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。または、IAM ロールの ARN を指定して、そのロールを使用することもできます。

S3_BUCKET 'amzn-s3-demo-bucket'

以前に作成した Amazon S3 バケットの名前は、Amazon Redshift と SageMaker の間でトレーニングデータとアーティファクトを共有するために使用されていました。Amazon Redshift は、トレーニングデータをアンロードする前に、このバケットにサブフォルダを作成します。トレーニングが完了すると、Amazon Redshift は作成したサブフォルダとその内容を削除します。

MAX_CELLS 整数

FROM 句からエクスポートするセルの最大数。デフォルトは 1,000,000 です。

セル数は、トレーニングデータ (FROM 句のテーブルまたはクエリによって生成される) の行数と列数を掛けた積です。トレーニングデータのセル数が max_cells パラメータで指定された数よりも多い場合、CREATE MODEL は FROM 句のトレーニングデータをダウンサンプリングして、トレーニングセットのサイズを MAX_CELLS 未満に減らします。大規模なトレーニングデータセットを許可すると、精度が高くなる可能性がありますが、モデルのトレーニングに時間がかかり、コストも高くなる可能性があります。

Amazon Redshift の使用コストについては、「Amazon Redshift 機械学習を使用するためのコスト」を参照してください。

さまざまなセル番号に関連するコストと無料トライアルの詳細については、Amazon Redshift の料金を参照してください。

ユーザーガイダンス付きの CREATE MODEL

以下に、単純な CREATE MODEL で説明されているオプションに加えて、CREATE MODEL オプションの説明を示します。

デフォルトでは、CREATE MODEL は特定のデータセットの前処理とモデルの最適な組み合わせを検索します。モデルに対して追加の制御が必要な場合や、追加のドメイン知識 (問題の種類や目的など) を導入する場合があります。顧客解約シナリオでは、「顧客がアクティブではない」という結果がまれである場合、精度目標よりも F1 目標が優先されることがよくあります。高精度モデルでは、常に「顧客がアクティブ」であると予測される可能性があるため、精度は高くなりますが、ビジネス価値はほとんどありません。F1 目標の詳細については、Amazon SageMaker API リファレンスの AutoMLJobObjective を参照してください。

次に、CREATE MODELは、目標など、指定された側面に関する提案に従います。同時に、CREATE MODEL は最適なプリプロセッサと最適なハイパーパラメータを自動的に検出します。

ユーザーガイダンス構文付きの CREATE MODEL

CREATE MODEL を使用すると、指定できる側面と Amazon Redshift が自動的に検出する側面について、より柔軟性が高まります。

```
CREATE MODEL model_name
FROM { table_name | ( select_statement ) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default }
[ MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER} ]
[ PROBLEM_TYPE ( REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION ) ]
[ OBJECTIVE ( 'MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC') ]
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', |
  S3_GARBAGE_COLLECT { ON | OFF }, |
  KMS_KEY_ID 'kms_key_id', |
  MAX_CELLS integer, |
  MAX_RUNTIME integer (, ...)
)
```

ユーザーガイダンスパラメータ付きの CREATE MODEL

MODEL_TYPE { XGBOOST | MLP | LINEAR_LEARNER }

(オプション) モデルタイプを指定します。XGBoost、多層パーセプトロン (MLP)、線形学習など、Amazon SageMaker Autopilot でサポートされているすべてのアルゴリズムである、特定のモデルタイプのモデルをトレーニングするかを指定できます。パラメータを指定しない場合、トレーニング中にサポートされているすべてのモデルタイプが最適なモデルを検索します。

PROBLEM_TYPE (REGRESSION | BINARY_CLASSIFICATION | MULTICLASS_CLASSIFICATION)

(オプション) 問題の種類を指定します。問題の種類がわかっている場合は、Amazon Redshift をその特定のモデルタイプの最適なモデルだけを検索するように制限できます。このパラメータを指定しない場合、トレーニング中にデータに基づく問題の種類が検出されます。

OBJECTIVE ('MSE' | 'Accuracy' | 'F1' | 'F1Macro' | 'AUC')

(オプション) 機械学習システムの予測品質を測定するために使用する目標メトリクスの名前を指定します。このメトリクスは、トレーニング中に最適化され、データからモデルパラメータ値の最良の推定値を提供します。メトリクスを明示的に指定しない場合、デフォルトの動作では、MSE が回帰に、F1 がバイナリ分類に、精度がマルチクラス分類に自動的に使用されます。目標の詳細については、Amazon SageMaker API リファレンスの AutoMLJobObjective を参照してください。

MAX_CELLS 整数

(オプション) トレーニングデータのセル数を指定します。この値は、(トレーニングクエリまたはテーブル内の) レコード数と列数の積です。デフォルトは 1,000,000 です。

MAX_RUNTIME 整数

(オプション) トレーニングする最大時間を指定します。トレーニングジョブは、データセットのサイズに応じてより早く完了することがよくあります。これは、トレーニングにかかる最大時間を指定します。デフォルトは 5,400 (90 分) です。

S3_GARBAGE_COLLECT { ON | OFF }

(オプション) Amazon Redshift がモデルおよびモデルのトレーニングに使用される結果のデータセットに対してガベージコレクションを実行するかどうかを指定します。OFF に設定すると、モデルとモデルのトレーニングに使用される結果のデータセットは Amazon S3 に残り、他の目的に使用できます。ON に設定すると、Amazon Redshift はトレーニングの完了後に Amazon S3 内のアーティファクトを削除します。デフォルトはオンです。

KMS_KEY_ID 'kms_key_id'

(オプション) AWS KMS キーを使用したサーバー側の暗号化を使用して、Amazon Redshift が保管中のデータを保護するかどうかを指定します。転送中のデータは Secure Sockets Layer (SSL) で保護されています。

PREPROCESSORS 'string'

(オプション) 特定の列セットに対するプリプロセッサの特定の組み合わせを指定します。形式は、columnSets のリストであり、各列のセットに適用される適切な変換です。Amazon Redshift

は、特定のトランスフォーマのリスト内にあるすべてのトランスフォーマを対応する ColumnSet 内のすべての列に適用します。例えば、Imputer を使用した OneHotEncoder を列 t1 と t2 に適用するには、次のサンプルコマンドを使用します。

```
CREATE MODEL customer_churn
FROM customer_data
TARGET 'Churn'
FUNCTION predict_churn
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
PROBLEM_TYPE BINARY_CLASSIFICATION
OBJECTIVE 'F1'
PREPROCESSORS '[
...
{"ColumnSet": [
    "t1",
    "t2"
  ],
  "Transformers": [
    "OneHotEncoder",
    "Imputer"
  ]
},
{"ColumnSet": [
    "t3"
  ],
  "Transformers": [
    "OneHotEncoder"
  ]
},
{"ColumnSet": [
    "temp"
  ],
  "Transformers": [
    "Imputer",
    "NumericPassthrough"
  ]
}
]'
SETTINGS (
S3_BUCKET 'amzn-s3-demo-bucket'
)
```

Amazon Redshift では、次のトランスフォーマをサポートしています。

- OneHotEncoder – 通常、離散値をゼロ以外の値を持つバイナリベクトルにエンコードするために使用されます。このトランスフォーマは、多くの機械学習モデルに適しています。
- OrdinalEncoder – 離散値を 1 つの整数にエンコードします。このトランスフォーマは、MLP や線形学習など特定の機械学習モデルに適しています。
- NumericPassthrough – 入力をそのままモデルに渡します。
- Imputer – 数値 (NaN) の値ではなく、欠損値を記入します。
- ImputerWithIndicator – 欠損値と NaN 値を埋めます。このトランスフォーマは、値が欠落して入力されているかどうかを示すインジケータも作成します。
- Normalizer – 値を正規化します。これにより、多くの機械学習アルゴリズムのパフォーマンスが向上します。
- DateTimeVectorizer – 機械学習モデルで使用できる日時データ型の列を表すベクトル埋め込みを作成します。
- PCA – できるだけ多くの情報を保持しながら、特徴の数を減らすためにデータを低い次元空間に投影します。
- StandardScaler – 平均を除去し、単位分散にスケーリングして、特徴を標準化します。
- MinMax – それぞれの特徴を指定された範囲にスケーリングすることで、特徴を変換します。

Amazon Redshift ML は、トレーニング済みのトランスフォーマを保存し、予測クエリの一部として自動的に適用します。モデルから予測を生成するときにそれらを指定する必要はありません。

AUTO OFF 付きの CREATE XGBoost モデル

AUTO OFF CREATE MODEL の目的は、通常、デフォルトの CREATE MODEL の目的とは異なります。

使用するモデル型と、これらのモデルをトレーニングする際に使用するハイパーパラメータを既に把握している上級ユーザーであれば、CREATE MODEL で AUTO OFF を指定することで、プリプロセッサとハイパーパラメータに関する CREATE MODEL による自動検出をオフにすることができます。そのためには、明示的にモデルタイプを指定します。XGBoost は現在、AUTO が OFF に設定されている場合にサポートされる唯一のモデルタイプです。ハイパーパラメータを指定できます。Amazon Redshift では、指定したハイパーパラメータにデフォルト値が使用されます。

AUTO OFF 構文を使用する CREATE XGBoost モデル

```
CREATE MODEL model_name
```

```
FROM { table_name | (select_statement ) }
TARGET column_name
FUNCTION function_name
IAM_ROLE { default }
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE { 'reg:squarederror' | 'reg:squaredlogerror' | 'reg:logistic' |
            'reg:pseudohubererror' | 'reg:tweedie' | 'binary:logistic' | 'binary:hinge'
 |
            'multi:softmax' | 'rank:pairwise' | 'rank:ndcg' }
HYPERPARAMETERS DEFAULT EXCEPT (
    NUM_ROUND '10',
    ETA '0.2',
    NUM_CLASS '10',
    (, ...)
)
PREPROCESSORS 'none'
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', |
  S3_GARBAGE_COLLECT { ON | OFF }, |
  KMS_KEY_ID 'kms_key_id', |
  MAX_CELLS integer, |
  MAX_RUNTIME integer (, ...)
)
```

AUTO OFF パラメータ付き CREATE XGBoost モデル

AUTO OFF

プリプロセッサ、アルゴリズム、およびハイパーパラメータの選択での、CREATE MODEL による自動検出をオフにします。

MODEL_TYPE XGBOOST

XGBOOST を使用してモデルをトレーニングするよう指定します。

OBJECTIVE str

アルゴリズムによって認識される目標を指定します。Amazon Redshift は reg:squarederror、reg:squaredlogerror、reg:logistic、reg:pseudohubererror、reg:tweedie、binary:logisti をサポートしています。これらの目的の詳細については、XGBoost のドキュメントから[ラーニングタスクのパラメータ](https://xgboost.readthedocs.io/)を参照してください。

HYPERPARAMETERS { DEFAULT | DEFAULT EXCEPT (key ‘value’ (,..)) }

デフォルトの XGBoost パラメータを使用するか、それをユーザー指定の値で上書きするかを指定します。値は一重引用符で囲む必要があります。以下に、XGBoost のパラメータとそのデフォルトの例を示します。

パラメータ名	パラメータ値	デフォルト値	コメント
num_class	整数	マルチクラス分類に必須です。	該当なし
num_round	整数	100	該当なし
tree_method	文字列	Auto	該当なし
max_depth	整数	6	[0 , 10]
min_child_weight	浮動小数点	1	MinValue: 0、MaxValue: 120
subsample	浮動小数点	1	MinValue: 0.5、MaxValue: 1
gamma	浮動小数点	0	MinValue: 0、MaxValue: 5
alpha	浮動小数点	0	MinValue: 0、MaxValue: 1000
eta	浮動小数点	0.3	MinValue: 0.1、MaxValue: 0.5
colsample_byleve	浮動小数点	1	MinValue: 0.1、MaxValue: 1

パラメータ名	パラメータ値	デフォルト値	コメント
colsample_bynode	浮動小数点	1	MinValue: 0.1、MaxValue: 1
colsample_bytree	浮動小数点	1	MinValue: 0.5、MaxValue: 1
lambda	浮動小数点	1	MinValue: 0、MaxValue: 1000
max_delta_step	整数	0	[0, 10]

次の例では、XGBoost 用のデータを準備します。

```
DROP TABLE IF EXISTS abalone_xgb;

CREATE TABLE abalone_xgb (
length_val float,
diameter float,
height float,
whole_weight float,
shucked_weight float,
viscera_weight float,
shell_weight float,
rings int,
record_number int);

COPY abalone_xgb
FROM 's3://redshift-downloads/redshift-ml/abalone_xg/'
REGION 'us-east-1'
IAM_ROLE default
IGNOREHEADER 1 CSV;
```

次の例では、MODEL_TYPE、OBJECTIVE、PREPROCESSORS など、指定された詳細オプションを使用して XGBoost モデルを作成します。

```
DROP MODEL abalone_xgboost_multi_predict_age;

CREATE MODEL abalone_xgboost_multi_predict_age
FROM ( SELECT length_val,
              diameter,
              height,
              whole_weight,
              shucked_weight,
              viscera_weight,
              shell_weight,
              rings
   FROM abalone_xgb WHERE record_number < 2500 )
TARGET rings FUNCTION ml_fn_abalone_xgboost_multi_predict_age
IAM_ROLE default
AUTO OFF
MODEL_TYPE XGBOOST
OBJECTIVE 'multi:softmax'
PREPROCESSORS 'none'
HYPERPARAMETERS DEFAULT EXCEPT (NUM_ROUND '100', NUM_CLASS '30')
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

次の例では、推論クエリを使用して、レコード番号が 2500 より大きい魚の年齢を予測します。上記のコマンドで作成した関数 ml_fn_abalone_xgboost_multi_predict_age を使用します。

```
select ml_fn_abalone_xgboost_multi_predict_age(length_val,
                                                   diameter,
                                                   height,
                                                   whole_weight,
                                                   shucked_weight,
                                                   viscera_weight,
                                                   shell_weight)+1.5 as age
from abalone_xgb where record_number > 2500;
```

独自のモデルを持参 (BYOM) - ローカル推論

Amazon Redshift 機械学習では、ローカルの推論において独自のモデルを持参 (BYOM) できるようサポートしています。

次に、BYOM の CREATE MODEL 構文のオプションをまとめます。Amazon Redshift の外部でトレーニングされたモデルを Amazon SageMaker とともに使用して、Amazon Redshift でローカルにデータベース内推論を行うことができます。

ローカル推論のための CREATE MODEL 構文

以下では、ローカル推論のための CREATE MODEL 構文について説明します。

```
CREATE MODEL model_name
FROM ('job_name' | 's3_path' )
FUNCTION function_name ( data_type [, ...] )
RETURNS data_type
IAM_ROLE { default }
[ SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket', | --required
  KMS_KEY_ID 'kms_string') --optional
];
```

Amazon Redshift は現在、BYOM 用に事前トレーニングされた XGBoost MLP および線形学習モデルのみをサポートしています。このパスを使用して、ローカル推論のために SageMaker Autopilot と Amazon SageMaker で直接トレーニングされたモデルをインポートできます。

ローカル推論のための CREATE MODEL パラメータ

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

FROM ('job_name' | 's3_path')

job_name では、入力として Amazon SageMaker のジョブ名を指定します。ジョブ名は、Amazon SageMaker トレーニングジョブ名または Amazon SageMaker Autopilot ジョブ名のいずれかです。ジョブは、Amazon Redshift クラスターを所有するのと同じ AWS アカウントで作成する必要があります。ジョブ名を見つけるには、Amazon SageMaker を起動します。[トレーニング] ドロップダウンメニューで、[トレーニングジョブ] を選択します。

's3_path' では、モデルの作成時に使用される .tar.gz モデルアーティファクトファイルを格納する S3 の場所を指定します。

FUNCTION function_name (data_type [, ...])

作成する関数の名前と、入力引数のデータ型。スキーマ名を指定できます。

RETURNS data_type

関数によって返される値のデータ型。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

デフォルトキーワードを使用して、CREATE MODEL コマンドの実行時にデフォルトとして設定され、同時にクラスターに関連付けられた IAM ロールを、Amazon Redshift が使用するようにします。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。

SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket', | KMS_KEY_ID 'kms_string')

S3_BUCKET 句は、中間結果の保存に使用される Amazon S3 の場所を指定します。

(オプション) KMS_KEY_ID 句は、Amazon Redshift が AWS KMS キーを使用したサーバー側の暗号化を使用して、保管中のデータを保護するかどうかを指定します。転送中のデータは Secure Sockets Layer (SSL) で保護されています。

詳細については、「ユーザーガイダンス付きの CREATE MODEL」を参照してください。

ローカル推論のための CREATE MODEL 例

次の例では、Amazon Redshift 以外の Amazon SageMaker で以前にトレーニングされたモデルを作成します。モデル型は Amazon Redshift ML によってローカル推論用にサポートされているため、以下の CREATE MODEL では、Amazon Redshift でローカルで使用できる関数を作成します。SageMaker のトレーニングジョブ名を指定できます。

```
CREATE MODEL customer_churn
FROM 'training-job-customer-churn-v4'
FUNCTION customer_churn_predict (varchar, int, float, float)
RETURNS int
IAM_ROLE default
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');
```

モデルが作成されたら、指定された引数タイプで関数 customer_churn_predict を使用して予測を行うことができます。

独自のモデルを持参 (BYOM) - リモート推論

また、Amazon Redshift 機械学習では、リモート推論においても独自のモデルを持参 (BYOM) できるようサポートしています。

次に、BYOM の CREATE MODEL 構文のオプションをまとめます。

リモート推論のための CREATE MODEL 構文

次に、リモート推論用の CREATE MODEL 構文について説明します。

```
CREATE MODEL model_name
FUNCTION function_name ( data_type [, ...] )
RETURNS data_type
SAGEMAKER 'endpoint_name'[:'model_name']
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
[SETTINGS (MAX_BATCH_ROWS integer)];
```

リモート推論の CREATE MODEL パラメータ

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

FUNCTION fn_name ([data_type] [, ...])

関数の名前と入力引数のデータ型。サポートされているすべてのデータ型については、「データ型」を参照してください。`Geography`、`geometry`、`hllsketch` はサポートされていません。

また、`myschema.myfunction` などの 2 つの部分からなる表記を使用して、スキーマ内に関数名を指定することもできます。

RETURNS data_type

関数によって返される値のデータ型。サポートされているすべてのデータ型については、「データ型」を参照してください。`Geography`、`geometry`、`hllsketch` はサポートされていません。

SAGEMAKER 'endpoint_name'[:'model_name']

Amazon SageMaker エンドポイントの名前。エンドポイント名がマルチモデルのエンドポイントを指している場合は、使用するモデルの名前を追加します。エンドポイントは、Amazon Redshift クラスターと同じ AWS リージョン内でホストされる必要があります。エンドポイントを見つけるには、Amazon SageMaker を起動します。[推論] ドロップダウンメニューで、[エンドポイント] を選択します。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}

デフォルトキーワードを使用して、CREATE MODEL コマンドの実行時にデフォルトとして設定され、同時にクラスターに関連付けられた IAM ロールを、Amazon Redshift が使用するようにします。または、IAM ロールの ARN を指定して、そのロールを使用することもできます。

MAX_BATCH_ROWS 整数

Amazon Redshift が 1 回の SageMaker 呼び出しに対して 1 回のバッチリクエストで送信する最大行数。リモート推論を使用する BYOM でのみサポートされます。バッチ内の実際の行数は入力サイズにも依存しますが、この値以下です。このパラメータの最小値は 1 です。最大値は `INT_MAX` または 2,147,483,647 です。このパラメータは、入力データ型と返されたデータ型の両方が `SUPER` である場合にのみ必要です。デフォルト値は `INT_MAX` または 2,147,483,647 です。

モデルが SageMaker エンドポイントにデプロイされると、SageMaker は Amazon Redshift でモデルの情報を作成します。その後、外部関数を介して推論を実行します。SHOW MODEL コマンドを使用して、Amazon Redshift クラスターのモデル情報を表示できます。

リモート推論の使用上の注意のための CREATE MODEL

リモート推論に CREATE MODEL を使用する前に、次の点を考慮してください。

- エンドポイントは、Amazon Redshift クラスターを所有するのと同じ AWS アカウントでホストされている必要があります。
- Amazon SageMaker エンドポイントに Amazon Redshift からの推論呼び出しに対応するのに十分なリソースがあること、または Amazon SageMaker エンドポイントを自動的にスケーリングできることを確認してください。
- `SUPER` データ型を入力として使用していない場合、モデルは SageMaker の `text/CSV` のコンテンツタイプに対応するカンマ区切り値 (CSV) 形式の入力のみを受け入れます。
- `SUPER` データ型を入力として使用していない場合、モデルの出力は、関数の作成時に指定された型の単一の値です。出力は、SageMaker の `text/CSV` のコンテンツタイプを介してカンマ区切り値 (CSV) の形式です。`VARCHAR` データ型は引用符で囲むことはできません。また、新しい行を含めることはできず、各出力は新しい行に含める必要があります。
- モデルは空の文字列として null を受け入れます。
- 入力データ型が `SUPER` の場合、サポートされる入力引数は 1 つだけです。
- 入力データ型が `SUPER` の場合、返されるデータ型も `SUPER` である必要があります。
- MAX_BATCH_ROWS は、入力データ型と返されたデータ型の両方が SUPER の場合に必要です。
- 入力データ型が `SUPER` のとき、エンドポイント呼び出しのコンテンツ型は、MAX_BATCH_ROWS が `1` の場合は `application/json`、それ以外の場合は `application/jsonlines` です。

- 返されるデータ型が SUPER のとき、エンドポイント呼び出しの受け入れ型は、MAX_BATCH_ROWS が 1 の場合は `application/json`、それ以外の場合は `application/jsonlines` です。

リモート推論のための CREATE MODEL 例

次の例では、SageMaker エンドポイントを使用して予測を行うモデルを作成します。エンドポイントが実行されていることを確認して予測を行い、CREATE MODEL コマンドでその名前を指定します。

```
CREATE MODEL remote_customer_churn
FUNCTION remote_fn_customer_churn_predict (varchar, int, float, float)
RETURNS int
SAGEMAKER 'customer-churn-endpoint'
IAM_ROLE default;
```

次の例では、大規模言語モデル (LLM) を使用してリモート推論で BYOM を作成します。Amazon SageMaker Jumpstart でホストされている LLM は、`application/json` コンテンツ型を受け入れて返し、呼び出しごとに単一の JSON をサポートします。入力データ型と返されるデータ型は SUPER で、MAX_BATCH_ROWS は 1 に設定する必要があります。

```
CREATE MODEL sample_super_data_model
FUNCTION sample_super_data_model_predict(super)
RETURNS super
SAGEMAKER 'sample_super_data_model_endpoint'
IAM_ROLE default
SETTINGS (MAX_BATCH_ROWS 1);
```

K-MEANS を使用した CREATE MODEL

Amazon Redshift は、ラベル付けされていないデータをグループ化する K-Means アルゴリズムをサポートしています。このアルゴリズムは、データ内でグループを検出する際にクラスタリングで発生する問題を解決します。未分類のデータについては、その類似点と相違点に基づいたグループ分けと分割が行われます。

K-MEANS 構文による CREATE MODEL

```
CREATE MODEL model_name
FROM { table_name | ( select_statement ) }
FUNCTION function_name
```

```
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS 'string'
HYPERPARAMETERS DEFAULT EXCEPT ( K 'val' [, ...] )
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  KMS_KEY_ID 'kms_string', |
    -- optional
  S3_GARBAGE_COLLECT on / off, |
    -- optional
  MAX_CELLS integer, |
    -- optional
  MAX_RUNTIME integer
    -- optional);
```

K-MEANS パラメータによる CREATE MODEL

AUTO OFF

プリプロセッサ、アルゴリズム、およびハイパーパラメータの選択での、CREATE MODEL による自動検出をオフにします。

MODEL_TYPE KMEANS

KMEANS を使用してモデルをトレーニングするよう指定します。

PREPROCESSORS 'string'

特定の列セットに対するプリプロセッサの特定の組み合わせを指定します。形式は、columnSets のリストであり、各列のセットに適用される適切な変換です。Amazon Redshift は、StandardScaler、MinMax、NumericPassthrough という、3 つの K-Means プリプロセッサをサポートしています。K-Means の前処理を適用しない場合は、トランスフォーマーとして明示的に NumericPassthrough を選択します。サポートされる変換の詳細については、「ユーザーガイダンスパラメータ付きの CREATE MODEL」を参照してください。

K-Means アルゴリズムは、ユークリッド距離を使用して類似度を計算します。データの前処理により、モデルの特徴が同じスケールに維持され、信頼性の高い結果が保証されます。

HYPERPARAMETERS DEFAULT EXCEPT (K 'val' [, ...])

K-Means パラメータを使用するかどうかを指定します。K-Means アルゴリズムを使用する際には、Kパラメータを指定する必要があります。詳細については、Amazon SageMaker デベロッパーガイドの「K-Means Hyperparameters」を参照してください。

次に、K-Means 用のデータを準備する際の例を示します。

```
CREATE MODEL customers_clusters
FROM customers
FUNCTION customers_cluster
IAM_ROLE default
AUTO OFF
MODEL_TYPE KMEANS
PREPROCESSORS '[
{
  "ColumnSet": [ "*" ],
  "Transformers": [ "NumericPassthrough" ]
}
]'
HYPERPARAMETERS DEFAULT EXCEPT ( K '5' )
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket');

select customer_id, customers_cluster(...) from customers;
customer_id | customers_cluster
--------------------
12345           1
12346           2
12347           4
12348
```

予測による CREATE MODEL

Redshift ML の予測モデルは Amazon Forecast を使用して正確な時系列予測を作成します。そうすることで、ある期間の履歴データを使用して将来のイベントを予測することができます。Amazon Forecast の一般的な使用例としては、小売製品データを使用して在庫の価格を決定する、製造数量データを使用して注文する商品の量を予測する、ウェブトラフィックデータを使用してウェブサーバーが受け取る可能性のあるトラフィック量を予測する、などが含まれます。

Amazon Redshift 予測モデルには、Amazon Forecast のクォータ制限が適用されます。例えば、予測の最大数は 100 ですが、調整可能です。予測モデルを削除しても、Amazon Forecast 内の関連リソースは自動的に削除されません。Redshift クラスターを削除すると、関連するモデルもすべて削除されます。

予測モデルは現在、以下のリージョンでのみ利用可能であることに注意してください。

- 米国東部 (オハイオ) (us-east-2)
- 米国東部 (バージニア北部) (us-east-1)

- 米国西部 (オレゴン) (us-west-2)
- アジアパシフィック (ムンバイ) (ap-south-1)
- アジアパシフィック (ソウル) (ap-northeast-2)
- アジアパシフィック (シンガポール) (ap-southeast-1)
- アジアパシフィック (シドニー) (ap-southeast-2)
- アジアパシフィック (東京) (ap-northeast-1)
- ヨーロッパ (フランクフルト) (eu-central-1)
- 欧州 (アイルランド) (eu-west-1)

予測構文による CREATE MODEL

```
CREATE [ OR REPLACE ] MODEL forecast_model_name
FROM { table_name | ( select_query ) }
TARGET column_name
IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>'}
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (
  S3_BUCKET 'amzn-s3-demo-bucket',
  HORIZON integer,
  FREQUENCY forecast_frequency
  [PERCENTILES '0.1', '0.5', '0.9']
  )
```

予測パラメータによる CREATE MODEL

forecast_model_name

モデルの名前です。モデル名は一意である必要があります。

FROM { table_name | (select_query) }

table_name またはトレーニングデータを指定するクエリ。これは、システム内の既存のテーブル、または丸かっこで囲まれた Amazon RedShift 互換の SELECT クエリのいずれかです。テーブルまたはクエリ結果には、少なくとも次の 3 つの列が必要です。(1) 時系列の名前を指定する varchar 列。各データセットには、複数の時系列を含めることができます。(2) datetime 列、(3) 予測する対象列。このターゲット列は int または float である必要があります。3 つ以上の列を含むデータセットを指定した場合、Amazon Redshift では、追加の列はすべて関連する時系列の一部であると見なします。関連する時系列は int 型または float 型である必要があることに注意して

ください。関連する時系列の詳細については、「関連する時系列データセットの使用」を参照してください。

TARGET column_name

予測対象となる列の名前。列は、FROM 句内に存在する必要があります。

IAM_ROLE { default | 'arn:aws:iam::<account-id>:role/<role-name>' }

デフォルトキーワードを使用して、CREAT MODEL コマンドの実行時にデフォルトとして設定され、クラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。または、IAM ロールの ARN を指定して、そのロールを使用することもできます。

AUTO ON

アルゴリズムおよびハイパーパラメータの選択での、CREATE MODEL による自動検出をオンにします。予測モデルを作成するときに on を指定すると、Forecast AutoPredictor を使用することを示します。ここで、Amazon Forecast は、データセット内の各時系列に最適なアルゴリズムの組み合わせを適用します。

MODEL_TYPE FORECAST

FORECAST を使用してモデルをトレーニングするよう指定します。

S3_BUCKET 'amzn-s3-demo-bucket'

以前に作成した Amazon Simple Storage Service バケットの名前で、これは Amazon Redshift と Amazon Forecast の間でトレーニングデータとアーティファクトを共有するために使用されます。Amazon Redshift は、トレーニングデータをアンロードする前に、このバケットにサブフォルダを作成します。トレーニングが完了すると、Amazon Redshift は作成したサブフォルダとその内容を削除します。

HORIZON 整数

予測モデルが返すことができる予測の最大数。モデルのトレーニングが完了すると、この整数は変更できません。

FREQUENCY forecast_frequency

予測の詳細度を指定します。使用できるオプションは、`Y | M | W | D | H | 30min | 15min | 10min | 5min | 1min` です。予測モデルをトレーニングする場合は必須です。

PERCENTILES 文字列

予測子のトレーニングに使用される予測タイプを指定するカンマ区切り文字列。予測タイプは、0.01 以上の増分で 0.01 から 0.99 までの分位数にすることができます。mean で平均予測を指定することもできます。最大 5 つの予測タイプを指定できます。

次の例は、簡単な予測モデルを作成する方法を示しています。

```
CREATE MODEL forecast_example
FROM forecast_electricity_
TARGET target
IAM_ROLE 'arn:aws:iam::<account-id>:role/<role-name>'
AUTO ON
MODEL_TYPE FORECAST
SETTINGS (S3_BUCKET 'amzn-s3-demo-bucket',
          HORIZON 24,
          FREQUENCY 'H',
          PERCENTILES '0.25,0.50,0.75,mean',
          S3_GARBAGE_COLLECT OFF);
```

予測モデルを作成したら、予測データを含む新しいテーブルを作成できます。

```
CREATE TABLE forecast_model_results as SELECT Forecast(forecast_example)
```

その後、新しいテーブルをクエリして予測を取得できます。

```
SELECT * FROM forecast_model_results
```

CREATE PROCEDURE

新しいストアドプロシージャを作成するか、現在のデータベースの既存のプロシージャを置き換えます。

詳細な説明と例については、「Amazon Redshift のストアドプロシージャの作成」を参照してください。

必要な権限

CREATE OR REPLACE PROCEDURE を実行するには、以下のいずれかの方法によるアクセス許可が必要です。

- CREATE PROCEDURE の場合:
  - スーパーユーザー
  - ストアドプロシージャの作成先のスキーマに対する CREATE 権限と USAGE 権限を持つユーザー

- REPLACE PROCEDURE の場合:
  - スーパーユーザー
  - プロシージャの所有者

構文

```
CREATE [ OR REPLACE ] PROCEDURE sp_procedure_name
  ( [ [ argname ] [ argmode ] argtype [, ...] ] )
[ NONATOMIC ]
AS $$
  procedure_body
$$ LANGUAGE plpgsql
[ { SECURITY INVOKER | SECURITY DEFINER } ]
[ SET configuration_parameter { TO value | = value } ]
```

パラメータ

OR REPLACE

プロシージャの名前と入力引数のデータタイプ (署名) が既存のプロシージャと同じである場合、既存のプロシージャを置き換えることを指定する句。プロシージャは、同じデータタイプセットを定義する新しいプロシージャにのみ置き換えることができます。

既存のプロシージャと名前は同じでも、署名が異なるプロシージャを定義する場合は、新しいプロシージャを作成することになります。つまり、プロシージャ名は重複されます。詳細については、「プロシージャ名の多重定義」を参照してください。

sp_procedure_name

プロシージャの名前。スキーマ名 (**myschema.myprocedure** など) を指定すると、プロシージャは指定したスキーマで作成されます。指定しない場合、プロシージャは現在のスキーマで作成されます。有効な名前の詳細については、「名前と識別子」を参照してください。

すべてのストアドプロシージャ名の前に sp_ を付けることをお勧めします。Amazon Redshift は、ストアドプロシージャ名用に sp_ プレフィックスを予約します。sp_ プレフィックスを使用すると、ストアドプロシージャ名は既存または将来の Amazon Redshift の組み込みストアドプロシージャ名や関数名と競合しません。詳細については、「ストアドプロシージャの名前付け」を参照してください。

入力引数のデータタイプ (署名) が異なる場合、同じ名前で複数のプロシージャを定義できます。つまり、この場合、プロシージャ名は重複されます。詳細については、「[プロシージャ名の多重定義]」を参照してください。

[argname] [argmode] argtype

引数の名前、モード、およびデータタイプのリスト。データタイプのみが必須です。名前とモードはオプションであり、両者の位置は交換できます。

引数のモードは IN、OUT、INOUT のいずれかです。デフォルトは IN です。

OUT 引数および INOUT 引数を使用して、プロシージャ呼び出しから 1 つ以上の値を返すことができます。OUT 引数または INOUT 引数がある場合、プロシージャ呼び出しは、n 列が含まれている 1 つの結果行を返します。n は OUT 引数または INOUT 引数の合計数です。

INOUT 引数は同時に入力引数であり、出力引数です。入力引数には IN 引数と INOUT 引数の両方が含まれます。出力引数には OUT 引数と INOUT 引数の両方が含まれます。

OUT 引数は CALL ステートメントの一部として指定されません。INOUT 引数は、ストアドプロシージャの CALL ステートメントに指定します。INOUT 引数は、ネストされた呼び出しとの間で値を受け渡すときと、`refcursor`を返すときに役立つ場合があります。`refcursor` タイプの詳細については、「[カーソル]」を参照してください。

引数のデータタイプには、Amazon Redshift のすべての標準データタイプを使用できます。さらに、引数のデータタイプとして `refcursor` も使用できます。

最大 32 個の入力引数と最大 32 個の出力引数を指定できます。

AS $$ procedure_body $$

実行するプロシージャを囲む構造。リテラルキーワードの AS $$ および $$ は必須です。

Amazon Redshift では、プロシージャのステートメントをドル引用符という形式を使用して囲む必要があります。囲まれた内容がそのまま渡されます。文字列の内容がそのまま書き込まれるため、特殊文字のエスケープは一切不要です。

ドル引用符付けでは、次の例に示すように、実行するステートメントの開始と終了を 2 つのドル記号のペア ($$) で指定します。

```
$$ my statement $$
```

必要に応じて、各ペアのドル記号間にステートメントの識別に役立つ文字列を指定できます。使用する文字列は、開始と終了の囲い文字のペアで同じにする必要があります。この文字列では大

文字と小文字が区別され、ドル記号を含めることができない点を除いては、引用符で囲まれていない識別子と同じ制約事項に従います。次の例では、文字列 test を使用しています。

```
$test$ my statement $test$
```

この構文は、ネストされたドル引用符付けにも役立ちます。ドル引用符付けの詳細については、PostgreSQL ドキュメントの「Lexical Structure」で「Dollar-quoted String Constants」を参照してください。

procedure_body

有効な PL/pgSQL ステートメントのセット。PL/pgSQL ステートメントは、SQL コマンドをプロシージャ構造体 (ループや条件式など) で強化し、論理フローを制御します。大半の SQL コマンドはプロシージャ本文で使用できます。これには、COPY、UNLOAD、INSERT などのデータ変更言語 (DML) と CREATE TABLE などのデータ定義言語 (DDL) が含まれます。詳細については、「PL/pgSQL 言語リファレンス」を参照してください。

LANGUAGE plpgsql

言語の値。`plpgsql` を指定します。`plpgsql` を使用するために言語の使用に関するアクセス許可が必要です。詳細については、「GRANT」を参照してください。

NONATOMIC

ストアドプロシージャを非アトミックトランザクションモードで作成します。NONATOMIC モードは、プロシージャ内のステートメントを自動的にコミットします。また、NONATOMIC プロシージャ内でエラーが発生しても、例外ブロックによって処理される場合、エラーは再スローされません。詳細については、「トランザクションの管理」および「RAISE」を参照してください。

ストアドプロシージャを `NONATOMIC` として定義する場合は、次の点を考慮してください。

- ストアドプロシージャコールをネストするときには、すべてのプロシージャを同じトランザクションモードで作成する必要があります。
- NONATOMIC モードでプロシージャを作成する場合、`SECURITY DEFINER` オプションと `SET configuration_parameter` オプションはサポートされません。
- すべての (明示的または暗黙的に) 開いているカーソルは、暗黙的なコミットが処理されるときに自動的に閉じられます。そのため、カーソルループを開始する前に明示的なトランザクションを開いて、ループの反復処理内の SQL が暗黙的にコミットされないようにする必要があります。

SECURITY INVOKER | SECURITY DEFINER

`SECURITY DEFINER` オプションは、`NONATOMIC` が指定されている場合はサポートされません。

プロシージャのセキュリティモードは、実行時のプロシージャのアクセス権限を決定します。プロシージャは、基礎となるデータベースオブジェクトにアクセスするためのアクセス許可を必要とします。

SECURITY INVOKER モードの場合、プロシージャはプロシージャを呼び出すユーザーの特権を使用します。ユーザーは、基礎となるデータベースオブジェクトに対する明示的なアクセス許可を必要とします。デフォルトは SECURITY INVOKER です。

SECURITY DEFINER モードの場合、このプロシージャは、プロシージャ所有者の権限を使用します。プロシージャ所有者の定義は、実行時にプロシージャを所有するユーザーであり、必ずしも最初にプロシージャを定義したユーザーではありません。プロシージャを呼び出すユーザーは、プロシージャに対する実行権限を必要としますが、基礎となるオブジェクトに対する権限は不要です。

SET configuration_parameter { TO value | = value }

これらのオプションは、`NONATOMIC` が指定されている場合はサポートされません。

SET 句は、プロシージャの開始時に、指定した `configuration_parameter` を指定した値に設定します。次に、プロシージャの終了時に、この句は `configuration_parameter` を以前の値に戻します。

使用に関する注意事項

SECURITY DEFINER オプションを使用してストアドプロシージャが作成された場合、そのストアドプロシージャ内から CURRENT_USER 関数を呼び出すと、Amazon Redshift はストアドプロシージャの所有者のユーザー名を返します。

例

> **Note**
>
> このような例を実行した場合、次のようなエラーが発生します。
>
> ```
> ERROR: 42601: [Amazon](500310) unterminated dollar-quoted string at or near "$$
> ```

「Amazon Redshift でのストアドプロシージャの概要」を参照してください。

次の例では、2 つの入力パラメータを使用するプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE test_sp1(f1 int, f2 varchar(20))
AS $$
DECLARE
  min_val int;
BEGIN
  DROP TABLE IF EXISTS tmp_tbl;
  CREATE TEMP TABLE tmp_tbl(id int);
  INSERT INTO tmp_tbl values (f1),(10001),(10002);
  SELECT INTO min_val MIN(id) FROM tmp_tbl;
  RAISE INFO 'min_val = %, f2 = %', min_val, f2;
END;
$$ LANGUAGE plpgsql;
```

> **Note**
>
> ストアドプロシージャを記述する場合は、機密の値を保護するためのベストプラクティスに従うことをお勧めします。
>
> ストアドプロシージャロジックに機密情報をハードコーディングしないでください。例えば、ストアドプロシージャの本文の CREATE USER ステートメントにユーザーパスワードを割り当てないでください。ハードコードした値は、カタログテーブルにスキーマメタデータとして記録される可能性があるため、セキュリティ上のリスクが生じます。代わりに、パスワードなどの機密の値は、パラメータを使用して引数として、ストアドプロシージャに渡します。
>
> ストアドプロシージャの詳細については、「CREATE PROCEDURE」と「Amazon Redshift のストアドプロシージャの作成」を参照してください。カタログテーブルの詳細については、「システムカタログテーブル」を参照してください。

次の例では、1 つの IN パラメータ、1 つの OUT パラメータ、および 1 つの INOUT パラメータを使用するプロシージャを作成します。

```
CREATE OR REPLACE PROCEDURE test_sp2(f1 IN int, f2 INOUT varchar(256), out_var OUT
 varchar(256))
```

```
AS $$
DECLARE
  loop_var int;
BEGIN
  IF f1 is null OR f2 is null THEN
    RAISE EXCEPTION 'input cannot be null';
  END IF;
  DROP TABLE if exists my_etl;
  CREATE TEMP TABLE my_etl(a int, b varchar);
    FOR loop_var IN 1..f1 LOOP
        insert into my_etl values (loop_var, f2);
        f2 := f2 || '+' || f2;
    END LOOP;
  SELECT INTO out_var count(*) from my_etl;
END;
$$ LANGUAGE plpgsql;
```

次の例では、SECURITY DEFINER パラメータを使用するプロシージャを作成します。このプロシージャは、プロシージャを所有するユーザーの権限を使用して実行します。

```
CREATE OR REPLACE PROCEDURE sp_get_current_user_definer()
AS $$
DECLARE curr_user varchar(250);
BEGIN
  SELECT current_user INTO curr_user;
  RAISE INFO '%', curr_user;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

次の例では、SECURITY INVOKER パラメータを使用するプロシージャを作成します。この手順は、プロシージャを実行するユーザーの権限を使用して実行します。

```
CREATE OR REPLACE PROCEDURE sp_get_current_user_invoker()
AS $$
DECLARE curr_user varchar(250);
BEGIN
  SELECT current_user INTO curr_user;
  RAISE INFO '%', curr_user;
END;
$$ LANGUAGE plpgsql
```

```
SECURITY INVOKER;
```

CREATE RLS POLICY

新しい行レベルのセキュリティポリシーを作成して、データベースオブジェクトへのきめ細かなアクセスを提供します。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、ポリシーを作成できます。

構文

```
CREATE RLS POLICY policy_name
[ WITH (column_name data_type [, ...]) [ [AS] relation_alias ] ]
USING ( using_predicate_exp )
```

パラメータ

policy_name

ポリシーの名前。

WITH (column_name data_type [, ...])

ポリシーがアタッチされているテーブルの列を参照対象の column_name と data_type を指定します。

RLS ポリシーがアタッチされているテーブルの列を参照対象としない場合に限り、WITH 句を省略できます。

AS relation_alias

RLS ポリシーをアタッチするテーブルにオプションのエイリアスを指定します。

USING (using_predicate_exp)

クエリの WHERE 句に適用されるフィルターを指定します。Amazon Redshift は、クエリレベルのユーザー述語より先にポリシー述語を適用します。例えば、**current_user = ‘joe’ and price > 10** は Joe に対して価格が 10 USD を超えるレコードのみを表示するように制限します。

使用に関する注意事項

CREATE RLS POLICY ステートメントを操作するとき、次の点に注意してください。

- Amazon Redshift は、クエリの WHERE 句の一部となるフィルタをサポートしています。
- テーブルにアタッチされるすべてのポリシーは、同じテーブルエイリアスで作成されている必要があります。
- ルックアップテーブルには SELECT 許可は必要ありません。ポリシーを作成するとき、Amazon Redshift はそれぞれのポリシーのルックアップテーブルに対する SELECT 許可を付与します。ルックアップテーブルは、ポリシー定義内で使用されるテーブルオブジェクトです。
- Amazon Redshift の行レベルセキュリティは、ポリシー定義内の以下のオブジェクトタイプをサポートしていません: カタログテーブル、データベース間の関係、外部テーブル、通常のビュー、遅延バインディングビュー、RLS ポリシーがオンになっているテーブル、一時テーブル。

例

次の SQL 文は、CREATE RLS POLICY の例に対してテーブル、ユーザー、ロールを作成します。

```
-- Create users and roles reference in the policy statements.
CREATE ROLE analyst;

CREATE ROLE consumer;

CREATE USER bob WITH PASSWORD 'Name_is_bob_1';

CREATE USER alice WITH PASSWORD 'Name_is_alice_1';

CREATE USER joe WITH PASSWORD 'Name_is_joe_1';

GRANT ROLE sys:secadmin TO bob;

GRANT ROLE analyst TO alice;

GRANT ROLE consumer TO joe;

GRANT ALL ON TABLE tickit_category_redshift TO PUBLIC;
```

次の例では、policy_concerts というポリシーを作成します。

```
CREATE RLS POLICY policy_concerts
```

```
WITH (catgroup VARCHAR(10))
USING (catgroup = 'Concerts');
```

CREATE ROLE

アクセス許可のコレクションである新しいカスタムロールを作成します。Amazon Redshift でのシステム定義のロールのリストについては、「the section called “Amazon Redshift でのシステム定義のロール”」を参照してください。クエリ SVV_ROLES を実行して、クラスターまたはワークグループに現在作成されているロールを表示します。

作成できるロールの数にはクォータがあります。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクォータと制限」を参照してください。

必要なアクセス許可

以下に、CREATE ROLE に必要な権限を示します。

- スーパーユーザー
- CREATE ROLE の権限を持つユーザー

構文

```
CREATE ROLE role_name
[ EXTERNALID external_id ]
```

パラメータ

role_name

ロールの名前。ロールの名前は一意にする必要があり、他のユーザー名と同じにすることはできません。ロールの名前に予約語は使用できません。

スーパーユーザー、または CREATE ROLE の権限を持つ通常のユーザーがロールを作成できます。スーパーユーザーでなくとも、ロールでの WITH GRANT OPTION および ALTER の権限に対する USAGE が付与されているユーザーは、そのロールを任意のユーザーに付与できます。

EXTERNALID external_id

ID プロバイダーに関連付けられているロールの識別子。詳細については、「Amazon Redshift のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

例

次の例では、ロール sample_role1 を作成しています。

```
CREATE ROLE sample_role1;
```

次の例では、ID プロバイダーに関連付けられている外部 ID を持つロール sample_role1 を作成します。

```
CREATE ROLE sample_role1 EXTERNALID "ABC123";
```

CREATE SCHEMA

現在のデータベースの新しいスキーマを定義します。

必要な権限

以下に、CREATE SCHEMA に必要な権限を示します。

- スーパーユーザー
- CREATE SCHEMA の権限を持つユーザー

構文

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name [ AUTHORIZATION username ]
           [ QUOTA {quota [MB | GB | TB] | UNLIMITED} ] [ schema_element [ ... ]

CREATE SCHEMA AUTHORIZATION username[ QUOTA {quota [MB | GB | TB] | UNLIMITED} ]
 [ schema_element [ ... ] ]
```

パラメータ

IF NOT EXISTS

指定されたスキーマが既に存在する場合、コマンドはエラーで終了するのではなく、何も変更しないで、スキーマが存在するというメッセージを返すことを示す句。

この句は、CREATE SCHEMA で既存のスキーマを作成しようとしてもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

schema_name

新しいスキーマの名前。スキーマ名を PUBLIC にすることはできません。有効な名前の詳細については、「名前と識別子」を参照してください。

> **Note**
>
> search_path 設定パラメータのスキーマリストによって、スキーマ名を指定せずに同じ名前のオブジェクトが参照されたときに、優先するオブジェクトが決まります。

AUTHORIZATION

指定されたユーザーに所有権を付与する句。

username

スキーマ所有者の名前。

schema_element

スキーマ内に作成する 1 つまたは複数のオブジェクトの定義。

QUOTA

指定したスキーマが使用できる最大ディスク容量。この容量は総ディスク使用量です。これには、各コンピューティングノードのすべてのディストリビューションのすべての永続テーブル、指定したスキーマのマテリアライズドビュー、すべてのテーブルの複製されたコピーが含まれます。スキーマクォータでは、一時的な名前空間またはスキーマの一部として作成された一時テーブルは考慮されません。

構成済みのスキーマクォータを表示するには、「SVV_SCHEMA_QUOTA_STATE」を参照してください。

スキーマクォータを超過したレコードを表示するには、「STL_SCHEMA_QUOTA_VIOLATIONS」を参照してください。

Amazon Redshift は選択した値をメガバイトに変換します。値を指定しない場合、デフォルトの測定単位はギガバイトです。

スキーマのクォータを設定および変更するには、データベースのスーパーユーザーである必要があります。スーパーユーザーではなくても、CREATE SCHEMA 権限を持つユーザーは、クォー

タを定義してスキーマを作成できます。クォータを定義せずにスキーマを作成すると、スキーマのクォータは無制限になります。クォータをスキーマで現在使用されている値よりも低く設定した場合、Amazon Redshift ではディスク容量を解放するまでそれ以上の取り込みを行うことはできません。DELETE ステートメントはテーブルからデータを削除するものであり、ディスク容量は VACUUM が実行された場合にのみ解放されます。

Amazon Redshift は、トランザクションをコミットする前に、各トランザクションのクォータ違反をチェックします。Amazon Redshift は、変更された各スキーマのサイズ (スキーマ内のすべてのテーブルで使用されるディスク容量) を、設定されたクォータに対してチェックします。クォータ違反のチェックはトランザクションの最後に行われるため、トランザクションをコミットする前にトランザクションでクォータを一時的に超過する場合があります。トランザクションがクォータを超過した場合、Amazon Redshift はそのトランザクションを中止し、後続の取り込みを禁止した上で、ディスク容量が解放されるまでの間はすべての変更を元に戻します。バックグラウンドでの VACUUM と内部のクリーンアップが原因で、トランザクションをキャンセルした後にスキーマをチェックするまでの間に、スキーマのサイズに空きが発生する場合があります。

例外として、Amazon Redshift はクォータ違反を無視し、特定の場合にトランザクションをコミットします。Amazon Redshift は、同じトランザクション内に INSERT または COPY の取り込みステートメントが存在しない、次の 1 つ以上のステートメントのみで構成されるトランザクションに対してこれを行います。

- DELETE
- TRUNCATE
- VACUUM
- DROP TABLE
- ALTER TABLE APPEND (サイズに空きのないスキーマから空きのあるスキーマにデータを移動する場合のみ)

UNLIMITED

Amazon Redshift は、スキーマの合計サイズの増加に制限を課しません。

制限

Amazon Redshift では、スキーマに次の制限があります。

- スキーマの数は 1 つのデータベースにつき最大 9900 個です。

例

次の例では、US_SALES というスキーマを作成し、DWUSER というユーザーに所有権を付与します。

```
create schema us_sales authorization dwuser;
```

次の例では、US_SALES というスキーマを作成し、DWUSER というユーザーに所有権を付与し、クォータを 50 GB に設定します。

```
create schema us_sales authorization dwuser QUOTA 50 GB;
```

新しいスキーマを確認するには、次に示すように PG_NAMESPACE カタログテーブルに対してクエリを実行します。

```
select nspname as schema, usename as owner
from pg_namespace, pg_user
where pg_namespace.nspowner = pg_user.usesysid
and pg_user.usename ='dwuser';

   schema |  owner
----------+----------
 us_sales | dwuser
(1 row)
```

次の例では、US_SALES スキーマを作成します。そのスキーマが既に存在する場合は、何もしないでメッセージを返します。

```
create schema if not exists us_sales;
```

CREATE TABLE

現在のデータベースに新しいテーブルを作成します。各列が異なる型のデータを保持する列のリストを定義します。テーブルの所有者は CREATE TABLE コマンドの発行者です。

必要な権限

以下に、CREATE TABLE に必要な権限を示します。

- スーパーユーザー
- CREATE TABLE の権限を持つユーザー

構文

```
CREATE [ [LOCAL ] { TEMPORARY | TEMP } ] TABLE
[ IF NOT EXISTS ] table_name
( { column_name data_type [column_attributes] [ column_constraints ]
  | table_constraints
  | LIKE parent_table [ { INCLUDING | EXCLUDING } DEFAULTS ] }
  [, ... ]  )
[ BACKUP { YES | NO } ]
[table_attributes]

where column_attributes are:
  [ DEFAULT default_expr ]
  [ IDENTITY ( seed, step ) ]
  [ GENERATED BY DEFAULT AS IDENTITY ( seed, step ) ]
  [ ENCODE encoding ]
  [ DISTKEY ]
  [ SORTKEY ]
  [ COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE  ]

and column_constraints are:
  [ { NOT NULL | NULL } ]
  [ { UNIQUE  |  PRIMARY KEY } ]
  [ REFERENCES reftable [ ( refcolumn ) ] ]

and table_constraints  are:
  [ UNIQUE ( column_name [, ... ] ) ]
  [ PRIMARY KEY ( column_name [, ... ] )  ]
  [ FOREIGN KEY (column_name [, ... ] ) REFERENCES reftable [ ( refcolumn ) ]


and table_attributes are:
  [ DISTSTYLE { AUTO | EVEN | KEY | ALL } ]
  [ DISTKEY ( column_name ) ]
  [ [COMPOUND | INTERLEAVED ] SORTKEY ( column_name [,...]) |  [ SORTKEY AUTO ] ]
  [ ENCODE AUTO ]
```

パラメータ

LOCAL

省略可能。このキーワードは、ステートメントに指定できますが、Amazon Redshift では効果がありません。

TEMPORARY | TEMP

現在のセッション内でのみ表示可能な一時テーブルを作成するキーワード。このテーブルは、作成したセッションの終了時に自動的に削除されます。一時テーブルには永続テーブルと同じ名前を付けることができます。一時テーブルは、別のセッション固有のスキーマ内に作成されます (このスキーマの名前を指定することはできません)。この一時スキーマは検索パスの最初のスキーマになるため、永続テーブルにアクセスするようにスキーマ名でテーブル名を修飾しない限り、一時テーブルは永続テーブルよりも優先されます。スキーマと優先順位の詳細については、「search_path」を参照してください。

> **Note**
>
> デフォルトでは、データベースユーザーは、PUBLIC グループの自動メンバーシップにより、一時テーブルを作成するアクセス許可を持ちます。あるユーザーにこの権限が付与されないようにするには、PUBLIC グループから TEMP 権限を取り消して、特定のユーザーまたはユーザーグループにのみ TEMP 権限を明示的に付与します。

IF NOT EXISTS

指定されたテーブルが既に存在する場合、コマンドはエラーで終了するのではなく、何も変更せずに、テーブルが存在するというメッセージを返すことを示す句。既存のテーブルの内容が、作成予定のテーブルとはまったく異なる可能性があることに注意してください。比較されるのはテーブル名のみです。

この句は、CREATE TABLE で既存のテーブルを作成しようとしてもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

table_name

作成するテーブルの名前。

> **Important**
>
> 「#」で始まるテーブル名を指定すると、そのテーブルは一時テーブルとして作成されます。次に例を示します。
>
> ```
> create table #newtable (id int);
> ```
>
> また、「#」を使用してテーブルを参照します。例:
>
> ```
> select * from #newtable;
> ```

テーブル名の最大長は 127 バイトです。それより長い名前は 127 バイトまで切り詰められます。最大 4 バイトまで UTF-8 マルチバイト文字を使用できます。Amazon Redshift は、ユーザー定義の一時テーブルや、クエリ処理またはシステムメンテナンス中に Amazon Redshift によって作成された一時テーブルを含む、各ノードタイプのクラスターあたりのテーブル数のクォータを適用します。オプションで、テーブル名は、データベース名およびスキーマ名で修飾することができます。次の例では、データベース名は `tickit`、スキーマ名は `public`、テーブル名は `test` です。

```
create table tickit.public.test (c1 int);
```

データベースまたはスキーマが存在し、テーブルが作成されていない場合、このステートメントはエラーを返します。システムデータベース `template0`、`template1`、`padb_harvest`、または `sys:internal` にテーブルまたはビューを作成することはできません。

スキーマ名を指定すると、新しいテーブルはそのスキーマ内に作成されます (作成者がスキーマにアクセス権を持っている場合)。テーブル名は、そのスキーマで一意の名前にする必要があります。スキーマを指定しない場合、現在のデータベーススキーマを使用してテーブルが作成されます。一時テーブルを作成する場合、一時テーブルは特殊なスキーマに作成されるので、スキーマ名を指定することはできません。

同じ名前の一時テーブルでも、別のセッションで作成される場合は同じデータベース内に同時に存在することができます。これは、テーブルに割り当てられるスキーマが異なるためです。有効な名前の詳細については、「名前と識別子」を参照してください。

column_name

新しいテーブルで作成される列の名前。列名の最大長は 127 バイトです。それより長い名前は 127 バイトまで切り詰められます。最大 4 バイトまで UTF-8 マルチバイト文字を使用できます。1 つのテーブルで定義できる列の最大数は 1,600 です。有効な名前の詳細については、「名前と識別子」を参照してください。

> **Note**
>
> 「横長のテーブル」を作成する場合は、ロードとクエリ処理の結果が即時に表示されるように、列リストが行幅の限度を超えないように注意します。詳細については、「使用に関する注意事項」を参照してください。

data_type

作成する列のデータ型。CHAR および VARCHAR の列の場合、最大長を宣言する代わりに MAX キーワードを使用できます。MAX は、最大長を CHAR では 4096 バイト、VARCHAR では 65535 バイトに設定します。GEOMETRY オブジェクトの最大サイズは 1,048,447 バイトです。

Amazon Redshift でサポートされているデータ型の詳細については、「データ型」を参照してください。

DEFAULT default_expr

列のデフォルトのデータ値を割り当てる句。default_expr のデータ型は列のデータ型に一致する必要があります。DEFAULT 値は、変数を使用しない式にする必要があります。サブクエリ、現在のテーブルに含まれる他の列の相互参照、およびユーザー定義の関数は使用できません。

default_expr 式は、列の値を指定しないすべての INSERT 操作で使用されます。デフォルト値を指定しなかった場合、列のデフォルト値は null です。

定義済み列リストの COPY 操作で、DEFAULT 値が含まれている列を省略すると、COPY コマンドで default_expr の値が挿入されます。

IDENTITY(seed, step)

列が IDENTITY 列であることを指定する句。IDENTITY 列には、一意の自動生成値が含まれます。IDENTITY 列のデータ型は、INT または BIGINT にする必要があります。

INSERT または INSERT INTO [tablename] VALUES() ステートメントを使用して行を追加する場合、これらの値は、seed として指定された値で始まり、step として指定された数が増分されます。

INSERT INTO [tablename] SELECT * FROM または COPY ステートメントを使用してテーブルをロードすると、データはパラレルにロードされ、ノードスライスに分散されます。ID 値が一意であることを確認するために、Amazon Redshift は ID 値を作成するときにいくつかの値をスキップします。ID 値は一意ですが、順序はソースファイルの順序と一致しない場合があります。

GENERATED BY DEFAULT AS IDENTITY (seed、step)

列がデフォルトの IDENTITY 列であることを指定し、一意の値を列に自動的に割り当てることができるようにする句。IDENTITY 列のデータ型は、INT または BIGINT にする必要があります。値のない行を追加する場合、これらの値は、seed として指定された値で始まり、step として指定された数が増分されます。値の生成方法については、「IDENTITY」を参照してください。

また、INSERT、UPDATE、または COPY の間に、EXPLICIT_IDS なしで値を指定できます。Amazon Redshift は、システムで生成された値を使用する代わりに、その値を使用して ID 列に挿入します。値は、重複、seed より小さい値、または step 値間の値にすることができます。Amazon Redshift は、列値の一意性を確認しません。値を指定しても、システムが生成する次の値には影響しません。

> **Note**
>
> 列に一意性が必要な場合は、重複する値を追加しないでください。代わりに、seed より小さい値または step 値の間にある一意の値を追加します。

デフォルトの identity 列については、次のことに注意してください。

- デフォルトの identity 列は NOT NULL です。NULL は挿入できません。
- 生成された値をデフォルトの identity 列に挿入するには、キーワード DEFAULT を使用します。

```
INSERT INTO tablename (identity-column-name) VALUES (DEFAULT);
```

- デフォルトの identity 列の値をオーバーライドしても、次に生成される値には影響しません。
- ALTER TABLE ADD COLUMN ステートメントでデフォルトの identity 列を追加することはできません。
- ALTER TABLE APPEND ステートメントを使用して、デフォルトの IDENTITY 列を追加できます。

ENCODE encoding

列の圧縮エンコード。ENCODE AUTO は、テーブルのデフォルトです。Amazon Redshift は、テーブル内のすべての列の圧縮エンコードを自動的に管理します。テーブル内のいずれかの列に圧縮エンコードを指定すると、テーブルは ENCODE AUTO に設定されなくなります。Amazon Redshift は、テーブルにあるすべての列の圧縮エンコードを自動的に管理しないようになりました。テーブルに ENCODE AUTO オプションを指定して、Amazon Redshift がテーブルにあるすべての列の圧縮エンコードを自動的に管理できるようになります。

Amazon Redshift は、圧縮エンコードを指定していない列に、次のように初期圧縮エンコードを自動的に割り当てます。

- 一時テーブルのすべての列には RAW 圧縮がデフォルトで割り当てられます。
- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY、もしくは GEOGRAPHY の各データ型で定義されている列には、RAW 圧縮が割り当てられます。
- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ として定義された列には AZ64 圧縮が割り当てられます。
- CHAR、VARCHAR、または VARBYTE として定義されている列には、LZO 圧縮が割り当てられます。

> Note
>
> 列を圧縮しない場合は、明示的に RAW エンコードを指定します。

次の compression encodings (p. 65) がサポートされています。

- AZ64
- BYTEDICT
- DELTA
- DELTA32K
- LZO
- MOSTLY8
- MOSTLY16
- MOSTLY32

- RAW (非圧縮)
- RUNLENGTH
- TEXT255
- TEXT32K
- ZSTD

DISTKEY

列がテーブルの分散キーであることを指定するキーワード。テーブル内の 1 つの列のみを分散キーに指定できます。DISTKEY キーワードは列名の後に使用するか、DISTKEY (column_name) 構文を使用してテーブル定義の一部として使用できます。どちらの方法でも同じ結果が得られます。詳細については、このトピックで後述する DISTSTYLE パラメータを参照してください。

ディストリビューションキー列のデータ型には、BOOLEAN、REAL、DOUBLE PRECISION、SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ、CHAR、または VARCHAR のいずれかを使用できます。

SORTKEY

列がテーブルのソートキーであることを指定するキーワード。データをテーブルにロードすると、データはソートキーとして指定された 1 つまたは複数の列に従ってソートされます。列名の後に SORTKEY キーワードを使用して 1 列のソートキーを指定することができます。または、SORTKEY (column_name [, ...]) 構文を使用して、テーブルのソートキーとして 1 つまたは数の列を指定することができます。この構文では複合ソートキーのみが作成されます。

1 つのテーブルで、最大 400 の SORTKEY 列を定義できます。

ソートキー列のデータ型には、BOOLEAN、REAL、DOUBLE PRECISION、SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ、CHAR、または VARCHAR のいずれかを使用できます。

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

列での文字列検索または比較が、CASE_SENSITIVE か CASE_INSENSITIVE かを指定する句。デフォルト値は、大文字と小文字を区別する、現行のデータベースの設定と同じです。

データベース照合に関する情報を検索するには、次のコマンドを使用します。

```
SELECT db_collation();
```

```
db_collation
----------------
 case_sensitive
(1 row)
```

NOT NULL | NULL

NOT NULL は、列に null 値を使用できないことを指定します。NULL はデフォルトであり、列で null 値を使用できることを指定します。IDENTITY 列はデフォルトで NOT NULL として宣言されます。

UNIQUE

列に一意の値のみを含めることができることを指定するキーワード。一意のテーブル制約の動作は、列制約の動作と同じですが、さらに複数の列に適用される点が異なります。一意のテーブル制約を定義するには、UNIQUE (column_name [, ...]) 構文を使用します。

> **Important**
> 一意の制約は情報提供に使用され、システムで強制されることはありません。

PRIMARY KEY

列がテーブルのプライマリキーであることを指定するキーワード。列定義を使用してプライマリキーとして定義できるのは 1 列だけです。複数列のプライマリキーを使用してテーブル制約を定義するには、PRIMARY KEY (column_name [, ...]) 構文を使用します。

列をプライマリキーに指定すると、スキーマの設計に関するメタデータが提供されます。プライマリキーは、他のテーブルが、行の一意の識別子としてこの列セットに依存している可能性があることを示します。列制約かテーブル制約かにかかわらず、テーブルに指定できるプライマリキーは 1 つです。プライマリキーの制約は、同じテーブルに定義されている一意の制約で指定された列セットとは異なる列セットを指定する必要があります。

PRIMARY KEY 列は NOT NULL としても定義されます。

> **Important**
> プライマリキーの制約は情報提供にのみ使用されます。プライマリキーの制約がシステムに強制されることはありませんが、プランナーによって使用されます。

References reftable [(refcolumn)]

参照テーブルの行の参照列に含まれている値と一致する値を列に格納する必要があることを意味する、外部キー制約を指定する句。参照列は、参照テーブルの一意の制約またはプライマリキーの制約の列にする必要があります。

> **Important**
> 外部キーの制約は情報提供にのみ使用されます。プライマリキーの制約がシステムに強制されることはありませんが、プランナーによって使用されます。

LIKE parent_table [{ INCLUDING | EXCLUDING } DEFAULTS]

新しいテーブルで、列名、データ型、NOT NULL 制約を自動的にコピーする既存のテーブルを指定する句。新しいテーブルと親テーブル間に関連付けはなく、親テーブルを変更しても、新しいテーブルに変更は適用されません。コピーした列定義のデフォルトの式は、INCLUDING DEFAULTS を指定した場合にのみコピーされます。デフォルトの動作では、デフォルトの式が除外されるため、新しいテーブルのすべての列は NULL のデフォルト値になります。

LIKE オプションを指定して作成したテーブルは、プライマリキーと外部キーの制約を継承しません。分散スタイル、ソートキー、BACKUP、NULL のプロパティは LIKE テーブルで継承されますが、CREATE TABLE ...。LIKE ステートメントで明示的に設定することはできません。

BACKUP { YES | NO }

テーブルを自動および手動クラスター スナップショットに含めるかどうかを指定する句。

重要なデータを含まないステージングテーブルなどのテーブルについては、スナップショットの作成やスナップショットからの復元にかかる時間を節約し、Amazon Simple Storage Service のストレージスペースを節約するため、BACKUP NO を指定します。BACKUP NO の設定は、クラスター内の別ノードへのデータの自動レプリケーションには影響しません。そのため、BACKUP NO が指定されたテーブルはノードの障害時に回復されます。デフォルトは BACKUP YES です。

DISTSTYLE { AUTO | EVEN | KEY | ALL }

テーブル全体のデータディストリビューションスタイルを定義するキーワード。Amazon Redshift は、テーブルに指定されたディストリビューションスタイルに従って、テーブルの行をコンピューティングノードに分散します。デフォルトは AUTO です。

テーブルにどの分散スタイルを選択するかによって、データベースの全体的なパフォーマンスが左右されます。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。可能な分散スタイルは次のとおりです。

- AUTO: Amazon Redshift がテーブルデータに基づいて最適な分散スタイルを割り当てます。例えば、AUTO 分散スタイルが指定された場合、Amazon Redshift ではまず、ALL 分散スタイルを小さなテーブルに割り当てます。テーブルが大きくなると、Amazon Redshift は分散スタイルを KEY に変更し、プライマリキー (または複合プライマリキーの列) を DISTKEY として選択する場合があります。テーブルが大きくなり、DISTKEY に適した列がない場合、Amazon Redshift は分散スタイルを EVEN に変更します。ディストリビューションスタイルの変更は、ユーザークエリへの影響を最小限に抑え、バックグラウンドで発生します。

  テーブルに適用された分散スタイルを表示するには、PG_CLASS システムカタログテーブルに対してクエリを実行します。詳細については、「分散スタイルの表示」を参照してください。
- EVEN: テーブルのデータは、ラウンドロビン分散方式で、クラスター内のノード全体に均等に分散されます。行 ID は分散を決定するために使用され、およそ同じ行数が各ノードに分散されます。
- KEY: データは、DISTKEY 列の値で分散されます。結合するテーブルの結合列を分散キーとして設定すると、両方のテーブルの結合列がコンピューティングノードによりコロケーションされます。データがコロケーションされることで、オプティマイザーはより効率的に結合を実行できます。DISTSTYLE KEY を指定する場合は、テーブルに対して、または列定義の一部として、DISTKEY 列に名前を付ける必要があります。詳細については、このトピックで先述した DISTKEY パラメータを参照してください。
- ALL: テーブル全体のコピーがすべてのノードに分散されます。この分散方式により、結合に必要なすべての列が確実にすべてのノードで使用可能になりますが、ストレージ要件が倍増し、テーブルに関する負荷とメンテナンス時間が増大します。ALL 分散を指定する場合、KEY 分散が適さない特定のディメンションテーブルで使用する場合には実行時間が改善される可能性がありますが、パフォーマンスの改善とメンテナンスコストを比較検討する必要があります。

DISTKEY (column_name)

テーブルの分散キーとして使用する列を指定する制約。DISTKEY キーワードは列名の後に使用するか、DISTKEY (column_name) 構文を使用してテーブル定義の一部として使用できます。どちらの方法でも同じ結果が得られます。詳細については、このトピックで先述した DISTSTYLE パラメータを参照してください。

[COMPOUND | INTERLEAVED] SORTKEY (column_name [,...]) | [SORTKEY AUTO]

テーブルに対して 1 つ以上のソートキーを指定します。データをテーブルにロードすると、データはソートキーとして指定された列に従ってソートされます。列名の後に SORTKEY キーワー

ドを使用して 1 列のソートキーを指定することができます。または、SORTKEY (column_name [, ...])構文を使用して、テーブルのソートキーとして 1 つまたは数の列を指定することができます。

オプションでソート方式として COMPOUND または INTERLEAVED を指定できます。列とともに SORTKEY を指定した場合、デフォルトは COMPOUND です。詳細については、「ソートキー」を参照してください。

ソートキーオプションを指定しない場合、デフォルトは AUTO です。

1 つのテーブルで最大 400 の COMPOUND SORTKEY 列または 8 の INTERLEAVED SORTKEY 列を定義できます。

AUTO

Amazon Redshift がテーブルデータに基づいて最適なソートキーを割り当てることを指定します。例えば、AUTO ソートキーが指定されている場合、Amazon Redshift は最初にテーブルにソートキーを割り当てません。ソートキーによってクエリのパフォーマンスが向上するとAmazon Redshift が判断した場合、Amazon Redshift はテーブルのソートキーを変更する可能性があります。テーブルの実際のソートは、自動テーブルソートによって行われます。詳細については、「自動テーブルソート」を参照してください。

Amazon Redshift は、既存のソートキーまたはディストリビューションキーを持つテーブルを変更しません。1 つの例外を除き、テーブルに JOIN で使用されたことがないディストリビューションキーがある場合、Amazon Redshift がより適切なキーがあると判断すると、キーが変更される可能性があります。

テーブルのソートキーを表示するには、SVV_TABLE_INFO システムカタログビューに対してクエリを実行します。詳細については、「SVV_TABLE_INFO」を参照してください。テーブルの Amazon Redshift Advisor のレコメンデーションを表示するには、SVV_ALTER_TABLE_RECOMMENDATIONS システムカタログビューをクエリします。詳細については、「SVV_ALTER_TABLE_RECOMMENDATIONS」を参照してください。Amazon Redshift が実行したアクションを表示するには、SVL_AUTO_WORKER_ACTION システムカタログビューにクエリを実行します。詳細については、「SVL_AUTO_WORKER_ACTION」を参照してください。

COMPOUND

リストされているすべての列から成る複合キーを使用して、データがリスト順にソートされるように指定します。複合ソートキーは、クエリがソート列の順序に従って行をスキャンする場合に最も便利です。複合キーを使用したソートのパフォーマンス上のメリットは、クエリがセ

カンダリソート列に依存する状況が減少する点にあります。1 つのテーブルで、最大 400 の COMPOUND SORTKEY 列を定義できます。

INTERLEAVED

インターリーブソートキーを使用してデータがソートされるように指定します。インターリーブソートキーには、最大 8 つの列を指定できます。

インターリーブソートは、ソートキーの各列または列のサブセットに等しい重みを割り当てるため、クエリはソートキーの列の順序に依存しません。クエリが 1 つ以上のセカンダリソート列を使用すると、インターリーブソートによってクエリのパフォーマンスは大幅に向上します。インターリーブソートでは、データのロード操作とバキューム操作にわずかなオーバーヘッドコストがかかります。

> **Important**
>
> ID 列、日付、タイムスタンプなど、一定間隔で増加する属性を持つ列で、インターリーブソートキーを使用しないでください。

ENCODE AUTO

Amazon Redshift がテーブル内のすべての列のエンコードタイプを自動的に調整して、クエリのパフォーマンスを最適化できるようにします。ENCODE AUTO は、テーブルの作成時に指定した初期エンコードタイプを保持します。その後、Amazon Redshift が新しいエンコードタイプによってクエリのパフォーマンスが向上すると判断した場合、Amazon Redshift はテーブル列のエンコードタイプを変更できます。テーブルのどの列にもエンコード型を指定しない場合、デフォルトは ENCODE AUTO です。

UNIQUE (column_name [,...])

テーブルの 1 つまたは複数の列のグループに、一意の値のみを含めることができることを指定する制約。一意のテーブル制約の動作は、列制約の動作と同じですが、さらに複数の列に適用される点が異なります。一意の制約がある場合、NULL 値は等値と見なされません。各一意のテーブル制約は、テーブルに定義されている他の一意のキーまたはプライマリキーで指定された列セットとは異なる列セットを指定する必要があります。

> **Important**
>
> 一意の制約は情報提供に使用され、システムで強制されることはありません。

PRIMARY KEY (column_name [,...])

テーブルの 1 つまたは複数の列のグループに、一意の (重複しない) NULL 以外の値のみを含めることができることを指定する制約。列セットをプライマリキーに指定すると、スキーマの設計に関するメタデータも提供されます。プライマリキーは、他のテーブルが、行の一意の識別子としてこの列セットに依存している可能性があることを示します。単一の列制約かテーブル制約かにかかわらず、テーブルに指定できるプライマリキーは 1 つです。プライマリキーの制約は、同じテーブルに定義されている一意の制約で指定された列セットとは異なる列セットを指定する必要があります。

> **Important**
>
> プライマリキーの制約は情報提供にのみ使用されます。プライマリキーの制約がシステムに強制されることはありませんが、プランナーによって使用されます。

FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn)]

外部キーの制約を指定する制約。この制約では、新しいテーブルの 1 列以上のグループに、参照テーブルのいずれかの行の参照列の値と一致する値のみを含める必要があります。refcolumn を省略すると、reftable のプライマリキーが使用されます。参照列は、参照テーブルの一意の制約またはプライマリキーの制約の列にする必要があります。

> **Important**
>
> 外部キーの制約は情報提供にのみ使用されます。プライマリキーの制約がシステムに強制されることはありませんが、プランナーによって使用されます。

使用に関する注意事項

一意性、プライマリキー、および外部キーの制約は情報提供のみを目的としており、テーブルに値を入れるときに Amazon Redshift によって強要されるわけではありません。例えば、依存関係のあるテーブルにデータを挿入する場合、制約に違反していても挿入は成功します。ただし、プライマリキーと外部キーはプランニング時のヒントとして使用されます。アプリケーションの ETL プロセスまたは他の何らかのプロセスによってこれらのキーの整合性が強要される場合は、これらのキーを宣言する必要があります。依存関係のあるテーブルを削除する方法については、「DROP TABLE」を参照してください。

制限とクォータ

テーブルを作成するときは、次の制限を考慮してください。

- ノードタイプごとにクラスター内のテーブルの最大数には制限があります。詳細については、「Amazon Redshift 管理ガイド」の「制限」を参照してください。
- テーブルの最大文字数は 127 です。
- 1 つのテーブルで定義できる列の最大数は 1,600 です。
- 1 つのテーブルで定義できる SORTKEY 列の最大数は 400 です。

列レベルの設定とテーブルレベルの設定の概要

列レベルまたはテーブルレベルで設定できる属性と設定値がいくつかあります。属性または制約を列レベルまたはテーブルレベルで設定して同じ結果が得られる場合もあれば、異なる結果が得られる場合もあります。

次のリストは、列レベルとテーブルレベルの設定値の概要を示したものです。

DISTKEY

列レベルで設定されるかテーブルレベルで設定されるかにかかわらず、結果に違いはありません。

列レベルかテーブルレベルかにかかわらず、DISTKEY が設定されている場合は、DISTSTYLE を KEY に設定するか、まったく設定しない必要があります。DISTSTYLE はテーブルレベルでしか設定できません。

SORTKEY

列レベルで設定されている場合、SORTKEY は 1 列でなければなりません。SORTKEY がテーブルレベルで設定されている場合は、1 つまたは複数の列で複合ソートキーまたはインターリーブコンポジットソートキーを構成できます。

COLLATE CASE_SENSITIVE | COLLATE CASE_INSENSITIVE

Amazon Redshift では、列での大文字と小文字の区別に関する設定の変更はサポートされていません。新しい列をテーブルに追加すると、Amazon Redshift は、大文字と小文字の区別に関する設定にデフォルト値を適用します。Amazon Redshift は、新しい列を追加するときに、COLLATE キーワードをサポートしません。

データベース照合を使用してデータベースを作成する方法については、「CREATE DATABASE」を参照してください。

COLLATE 関数の詳細については、「COLLATE 関数」を参照してください。

UNIQUE

列レベルでは、1 つまたは複数のキーを UNIQUE に設定できます。UNIQUE 制約は各列に個別に適用されます。UNIQUE がテーブルレベルで設定されている場合は、1 つまたは複数の列で複合 UNIQUE 制約を構成できます。

PRIMARY KEY

列レベルで設定されている場合、PRIMARY KEY は 1 列でなければなりません。PRIMARY KEY がテーブルレベルで設定されている場合は、1 つまたは複数の列で複合プライマリキーを構成できます。

外部キー

FOREIGN KEY が列レベルで設定されるかテーブルレベルで設定されるかにかかわらず、結果に違いはありません。列レベルでは、構文は単に REFERENCES reftable [(refcolumn)] になります。

受信データの分散

受信データのハッシュ分散スキームが、ターゲットテーブルのスキームと同じ場合、データをロードするときに、データを物理的に分散させる必要はありません。例えば、新しいテーブルに分散キーが設定されており、同じキー列で分散されている別のテーブルからデータが挿入される場合、同じノードとスライスを使用してデータが所定の位置にロードされます。ただし、ソーステーブルとターゲットテーブルの両方が EVEN 分散に設定されている場合、データはターゲットテーブルで再分散されます。

横長のテーブル

非常に横長のテーブルは、作成できても、そのテーブルに対して INSERT や SELECT などのクエリ処理を実行できないことがあります。CHAR のように固定幅の列を持つテーブルの最大幅は、64KB マイナス 1 (つまり 65535 バイト) です。テーブルに VARCHAR 列がある場合は、エラーを返さないで、より大きな幅を宣言できます。VARCHAR 列は、宣言した幅がクエリ処理の制限の算出に関係しないためです。VARCHAR 列に関する有効なクエリ処理の制限は、いくつかの要因に応じて変わります。

テーブルの幅が広すぎて挿入や選択ができない場合は、次のエラーが発生します。

```
ERROR:  8001
```

```
DETAIL:  The combined length of columns processed in the SQL statement
exceeded the query-processing limit of 65535 characters (pid:7627)
```

例

ALTER TABLE コマンドの使用方法を示す例については、「例」のトピックを参照してください。

例

次の例は、Amazon Redshift CREATE TABLE ステートメントのさまざまな列とテーブルの属性を示しています。パラメータ定義を含め、CREATE TABLE の詳細については、「CREATE TABLE」を参照してください。

例の多くは、TICKIT サンプルデータセットのテーブルとデータを使用しています。詳細については、「サンプルデータベース」を参照してください。

CREATE TABLE コマンドでは、テーブル名の前にデータベース名とスキーマ名を付けることができます。例えば、`dev_database.public.sales`。データベース名は、接続しているデータベースでなければなりません。別のデータベースにデータベースオブジェクトを作成しようとすると、無効な操作エラーで失敗します。

分散キー、複合ソートキー、圧縮を使用してテーブルを作成する

次の例では、複数の列に圧縮が定義された SALES テーブルを TICKIT データベースに作成します。LISTID は分散キーとして宣言され、LISTID と SELLERID は複数列の複合ソートキーとして宣言されています。このテーブルでは、プライマリキーと外部キーの制約も定義されています。この例のテーブルを作成する前に、制約が存在しない場合は、外部キーによって参照される各列に UNIQUE 制約を追加する必要がある場合があります。

```
create table sales(
salesid integer not null,
listid integer not null,
sellerid integer not null,
buyerid integer not null,
eventid integer not null encode mostly16,
dateid smallint not null,
qtysold smallint not null encode mostly8,
pricepaid decimal(8,2) encode delta32k,
commission decimal(8,2) encode delta32k,
saletime timestamp,
```

```
primary key(salesid),
foreign key(listid) references listing(listid),
foreign key(sellerid) references users(userid),
foreign key(buyerid) references users(userid),
foreign key(dateid) references date(dateid))
distkey(listid)
compound sortkey(listid,sellerid);
```

結果は以下のとおりです。

```
schemaname | tablename | column     | type                        | encoding | distkey
 | sortkey | notnull
-----------+-----------+------------+-----------------------------+----------+---------
+---------+--------
public     | sales     | salesid    | integer                     | lzo      | false
 |       0 | true
public     | sales     | listid     | integer                     | none     | true
 |       1 | true
public     | sales     | sellerid   | integer                     | none     | false
 |       2 | true
public     | sales     | buyerid    | integer                     | lzo      | false
 |       0 | true
public     | sales     | eventid    | integer                     | mostly16 | false
 |       0 | true
public     | sales     | dateid     | smallint                    | lzo      | false
 |       0 | true
public     | sales     | qtysold    | smallint                    | mostly8  | false
 |       0 | true
public     | sales     | pricepaid  | numeric(8,2)                | delta32k | false
 |       0 | false
public     | sales     | commission | numeric(8,2)                | delta32k | false
 |       0 | false
public     | sales     | saletime   | timestamp without time zone | lzo      | false
 |       0 | false
```

次の例では、大文字と小文字が区別されない列 col1 を持つテーブル t1 を作成します。

```
create table T1 (
  col1 Varchar(20) collate case_insensitive
 );

insert into T1 values ('bob'), ('john'), ('Tom'), ('JOHN'), ('Bob');
```

テーブルに対してクエリを実行します。

```
select * from T1 where col1 = 'John';

col1
------
 john
 JOHN
(2 rows)
```

インターリーブソートキーを使用してテーブルを作成する

次の例では、インターリーブソートキーを使用して CUSTOMER テーブルを作成しています。

```
create table customer_interleaved (
  c_custkey      integer        not null,
  c_name         varchar(25)    not null,
  c_address      varchar(25)    not null,
  c_city         varchar(10)    not null,
  c_nation       varchar(15)    not null,
  c_region       varchar(12)    not null,
  c_phone        varchar(15)    not null,
  c_mktsegment      varchar(10)    not null)
diststyle all
interleaved sortkey (c_custkey, c_city, c_mktsegment);
```

IF NOT EXISTS を使用したテーブルの作成

次の例では、CITIES テーブルを作成します。そのテーブルが既に存在する場合は、何もしないでメッセージを返します。

```
create table if not exists cities(
cityid integer not null,
city varchar(100) not null,
state char(2) not null);
```

ALL 分散を指定したテーブルの作成

次の例では、ALL 分散を指定して VENUE テーブルを作成します。

```
create table venue(
```

```
venueid smallint not null,
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid))
diststyle all;
```

EVEN 分散を指定したテーブルの作成

次の例では、3 つの列を持つ MYEVENT というテーブルを作成します。

```
create table myevent(
eventid int,
eventname varchar(200),
eventcity varchar(30))
diststyle even;
```

テーブルは均等に分散され、ソートされません。このテーブルに宣言された DISTKEY 列または SORTKEY 列はありません。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'myevent';

  column   |          type          | encoding | distkey | sortkey
-----------+------------------------+----------+---------+---------
 eventid   | integer                | lzo      | f       |       0
 eventname | character varying(200) | lzo      | f       |       0
 eventcity | character varying(30)  | lzo      | f       |       0
(3 rows)
```

別のテーブルの LIKE である一時テーブルの作成

次の例では、TEMPEVENT という一時テーブルを作成します。このテーブルは EVENT テーブルの列を継承します。

```
create temp table tempevent(like event);
```

また、このテーブルは、親テーブルの DISTKEY 属性と SORTKEY 属性も継承します。

```
select "column", type, encoding, distkey, sortkey
```

```
from pg_table_def where tablename = 'tempevent';

  column   |            type             | encoding | distkey | sortkey
-----------+-----------------------------+----------+---------+---------
 eventid   | integer                     | none     | t       |       1
 venueid   | smallint                    | none     | f       |       0
 catid     | smallint                    | none     | f       |       0
 dateid    | smallint                    | none     | f       |       0
 eventname | character varying(200)      | lzo      | f       |       0
 starttime | timestamp without time zone | bytedict | f       |       0
(6 rows)
```

IDENTITY 列があるテーブルの作成

次の例では、VENUE_IDENT というテーブルを作成します。このテーブルには、VENUEID という IDENTITY 列があります。この列は 0 から始まり、レコードごとに 1 ずつ増分します。VENUEID は、テーブルのプライマリキーとしても宣言されています。

```
create table venue_ident(venueid bigint identity(0, 1),
venuename varchar(100),
venuecity varchar(30),
venuestate char(2),
venueseats integer,
primary key(venueid));
```

デフォルトの IDENTITY 列があるテーブルの作成

次の例では、t1という名前のテーブルを作成します。このテーブルには、hist_idという名前の IDENTITY 列と、base_idという名前のデフォルトの IDENTITY 列があります。

```
CREATE TABLE t1(
  hist_id BIGINT IDENTITY NOT NULL, /* Cannot be overridden */
  base_id BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL, /* Can be overridden */
  business_key varchar(10) ,
  some_field varchar(10)
);
```

テーブルに行を挿入すると、hist_idと base_id の両方が生成されることがわかります。

```
INSERT INTO T1 (business_key, some_field) values ('A','MM');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
```

2 番目の行を挿入すると、base_idのデフォルト値が生成されることがわかります。

```
INSERT INTO T1 (base_id, business_key, some_field) values (DEFAULT, 'B','MNOP');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
```

3 番目の行を挿入すると、base_idの値が一意である必要がないことがわかります。

```
INSERT INTO T1 (base_id, business_key, some_field) values (2,'B','MNNN');
```

```
SELECT * FROM t1;

 hist_id | base_id | business_key | some_field
---------+---------+--------------+------------
       1 |       1 | A            | MM
       2 |       2 | B            | MNOP
       3 |       2 | B            | MNNN
```

DEFAULT 列値を指定したテーブルの作成

次の例では、各列のデフォルト値を宣言した CATEGORYDEF テーブルを作成します。

```
create table categorydef(
catid smallint not null default 0,
catgroup varchar(10) default 'Special',
catname varchar(10) default 'Other',
catdesc varchar(50) default 'Special events',
primary key(catid));
```

```
insert into categorydef values(default,default,default,default);
```

```
select * from categorydef;

 catid | catgroup | catname |    catdesc
-------+----------+---------+----------------
     0 | Special  | Other   | Special events
(1 row)
```

DISTSTYLE、DISTKEY、SORTKEY オプション

次の例は、DISTKEY、SORTKEY、DISTSTYLE オプションがどのように機能するかを示しています。この例で、COL1 は分散キーであるため、分散スタイルを KEY に設定するか、何も設定しない必要があります。デフォルトで、テーブルにはソートキーがないため、ソートされません。

```
create table t1(col1 int distkey, col2 int) diststyle key;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't1';

column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | az64     | t       | 0
col2   | integer | az64     | f       | 0
```

次の例では、同じ列が分散キーおよびソートキーとして定義されています。ここでも、分散スタイルを KEY に設定するか、何も設定しない必要があります。

```
create table t2(col1 int distkey sortkey, col2 int);
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't2';

column |  type   | encoding | distkey | sortkey
-------+---------+----------+---------+---------
col1   | integer | none     | t       | 1
col2   | integer | az64     | f       | 0
```

次の例では、分散キーに設定されている列はありません。また、COL2 はソートキーに設定され、分散スタイルは ALL に設定されています。

```
create table t3(col1 int, col2 int sortkey) diststyle all;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't3';

Column |  Type   | Encoding | DistKey | SortKey
-------+---------+----------+---------+--------
col1   | integer | az64     | f       | 0
col2   | integer | none     | f       | 1
```

次の例では、分散スタイルは EVEN に設定され、ソートキーは明示的に定義されていません。そのため、テーブルは均等に分散されますが、ソートされません。

```
create table t4(col1 int, col2 int) diststyle even;
```

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 't4';

             column |  type   |encoding | distkey | sortkey
--------+---------+---------+---------+--------
col1    | integer | az64    | f       | 0
col2    | integer | az64    | f       | 0
```

ENCODE AUTO オプションを使用してテーブルを作成する

次の例では、自動圧縮エンコードを使用してテーブル t1 を作成します。ENCODE AUTO は、どの列にもエンコード型を指定しない場合のテーブルのデフォルトです。

```
create table t1(c0 int, c1 varchar);
```

次の例では、ENCODE AUTO を指定して、自動圧縮エンコードを使用してテーブル t2 を作成します。

```
create table t2(c0 int, c1 varchar) encode auto;
```

次の例では、ENCODE AUTO を指定して、自動圧縮エンコードを使用してテーブル t3 を作成します。列 c0 は、初期エンコードタイプが DELTA で定義されています。別のエンコードがより良いクエリパフォーマンスを提供する場合、Amazon Redshift はエンコードを変更できます。

```
create table t3(c0 int encode delta, c1 varchar) encode auto;
```

次の例では、ENCODE AUTO を指定して、自動圧縮エンコードを使用してテーブル t4 を作成します。列 c0 は DELTA の初期エンコードで定義され、列 c1 は LZO の初期エンコードで定義されます。他のエンコードがより優れたクエリパフォーマンスを提供する場合、Amazon Redshift はこれらのエンコードを変更できます。

```
create table t4(c0 int encode delta, c1 varchar encode lzo) encode auto;
```

CREATE TABLE AS

トピック

- 構文
- パラメータ
- CTAS の使用に関する注意事項
- CTAS の例

クエリに基づいて新しいテーブルを作成します。このテーブルの所有者は、このコマンドを発行したユーザーになります。

新しいテーブルが作成され、コマンドのクエリで定義されたデータがロードされます。テーブルの列には、クエリの出力列に関連付けられた名前とデータ型が指定されます。CREATE TABLE AS (CTAS) コマンドを実行すると、新しいテーブルが作成され、新しいテーブルをロードするクエリが評価されます。

構文

```
CREATE [ [ LOCAL ] { TEMPORARY | TEMP } ]
TABLE table_name
[ ( column_name [, ... ] ) ]
[ BACKUP { YES | NO } ]
[ table_attributes ]
AS query

where table_attributes are:
[ DISTSTYLE { AUTO | EVEN | ALL | KEY } ]
[ DISTKEY( distkey_identifier ) ]
[ [ COMPOUND | INTERLEAVED ] SORTKEY( column_name [, ...] ) ]
```

パラメータ

LOCAL

このオプションのキーワードは、ステートメントに指定できますが、Amazon Redshift では効果がありません。

TEMPORARY | TEMP

一時テーブルを作成します。一時テーブルは、作成したセッションの終了時に削除されます。

table_name

作成するテーブルの名前。

> **Important**
>
> 「#」で始まるテーブル名を指定すると、そのテーブルは一時テーブルとして作成されます。次に例を示します。
>
> ```
> create table #newtable (id) as select * from oldtable;
> ```

テーブル名の最大長は 127 バイトです。それより長い名前は 127 文字まで切り詰められます。Amazon Redshift は、ノードタイプごとにクラスターあたりのテーブル数のクォータを適用します。次の表で示すように、テーブル名はデータベース名およびスキーマ名で修飾することができます。

```
create table tickit.public.test (c1) as select * from oldtable;
```

この例では、tickitはデータベース名、publicはスキーマ名です。このデータベースまたはスキーマが存在しない場合、ステートメントはエラーを返します。

スキーマ名を指定すると、新しいテーブルはそのスキーマ内に作成されます (作成者がスキーマにアクセス権を持っている場合)。テーブル名は、そのスキーマで一意の名前にする必要があります。スキーマを指定しない場合、現在のデータベーススキーマを使用してテーブルが作成されます。一時テーブルを作成する場合、一時テーブルは特殊なスキーマにあるのでスキーマ名を指定することはできません。

同じ名前の一時テーブルでも、別のセッションで作成される場合、同じデータベース内に同時に存在することができます。このようなテーブルは別のスキーマに割り当てられます。

column_name

新しいテーブルの列の名前。列名を指定しない場合、列名には、クエリの出力列名が使用されます。式にはデフォルトの列名が使用されます。有効な名前の詳細については、「名前と識別子」を参照してください。

BACKUP { YES | NO }

テーブルを自動および手動クラスター スナップショットに含めるかどうかを指定する句。

重要なデータを含まないステージングテーブルなどのテーブルについては、スナップショットの作成やスナップショットからの復元にかかる時間を節約し、Amazon Simple Storage Service のストレージスペースを節約するため、BACKUP NO を指定します。BACKUP NO の設定は、クラスター内の別ノードへのデータの自動レプリケーションには影響しません。そのため、BACKUP NO が指定されたテーブルはノードの障害時に回復されます。デフォルトは BACKUP YES です。

DISTSTYLE { AUTO | EVEN | KEY | ALL }

テーブル全体のデータディストリビューションスタイルを定義するキーワード。Amazon Redshift は、テーブルに指定されたディストリビューションスタイルに従って、テーブルの行をコンピューティングノードに分散します。デフォルトは DISTSTYLE AUTO です。

テーブルにどの分散スタイルを選択するかによって、データベースの全体的なパフォーマンスが左右されます。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

- AUTO: Amazon Redshift がテーブルデータに基づいて最適な分散スタイルを割り当てます。テーブルに適用された分散スタイルを表示するには、PG_CLASS システムカタログテーブルに対してクエリを実行します。詳細については、「分散スタイルの表示」を参照してください。
- EVEN: テーブルのデータは、ラウンドロビン分散方式で、クラスター内のノード全体に均等に分散されます。行 ID は分散を決定するために使用され、およそ同じ行数が各ノードに分散されます。これはデフォルトの分散方法です。
- KEY: データは、DISTKEY 列の値で分散されます。結合するテーブルの結合列を分散キーとして設定すると、両方のテーブルの結合列がコンピューティングノードによりコロケーションされます。データがコロケーションされることで、オプティマイザーはより効率的に結合を実行できます。DISTSTYLE KEY を指定する場合、DISTKEY 列を指定する必要があります。
- ALL: テーブル全体のコピーがすべてのノードに分散されます。この分散方式により、結合に必要なすべての列が確実にすべてのノードで使用可能になりますが、ストレージ要件が倍増し、テーブルに関する負荷とメンテナンス時間が増大します。ALL 分散を指定する場合、KEY 分散

が適さない特定のディメンションテーブルで使用する場合には実行時間が改善される可能性がありますが、パフォーマンスの改善とメンテナンスコストを比較検討する必要があります。

DISTKEY (column)

分散キーの列名または位置番号を指定します。テーブルのオプションの列リストまたはクエリの選択したリストで指定された名前を使用します。または、位置番号を使用します。この番号は、最初に選択された列は 1、2 番目は 2 などと続きます。テーブル内の 1 つの列のみを分散キーに指定できます。

- 列を DISTKEY 列として宣言する場合、DISTSTYLE を KEY に設定するか、まったく設定しない必要があります。
- DISTKEY 列を宣言しない場合、DISTSTYLE を EVEN に設定することができます。
- DISTKEY または DISTSTYLE を指定しない場合、CTAS は SELECT 句のクエリプランに基づいて新しいテーブルの分散スタイルを決定します。詳細については、「列属性とテーブル属性の継承」を参照してください。

同じ列を分散キーおよびソートキーとして定義できます。この方法の場合、該当する列をクエリで結合する列にすると、結合が高速になる傾向があります。

[COMPOUND | INTERLEAVED] SORTKEY (column_name [, ...])

テーブルに対して 1 つ以上のソートキーを指定します。データをテーブルにロードすると、データはソートキーとして指定された列に従ってソートされます。

オプションでソート方式として COMPOUND または INTERLEAVED を指定できます。デフォルトは COMPOUND です。詳細については、「ソートキー」を参照してください。

1 つのテーブルで最大 400 の COMPOUND SORTKEY 列または 8 の INTERLEAVED SORTKEY 列を定義できます。

SORTKEY を指定しない場合、CTAS は SELECT 句のクエリプランに基づいて新しいテーブルのソートキーを決定します。詳細については、「列属性とテーブル属性の継承」を参照してください。

COMPOUND

リストされているすべての列から成る複合キーを使用して、データがリスト順にソートされるように指定します。複合ソートキーは、クエリがソート列の順序に従って行をスキャンする場合に最も便利です。複合キーを使用したソートのパフォーマンス上のメリットは、クエリがセカンダリソート列に依存する状況が減少する点にあります。1 つのテーブルで、最大 400 の COMPOUND SORTKEY 列を定義できます。

INTERLEAVED

インターリーブソートキーを使用してデータがソートされるように指定します。インターリーブソートキーには、最大 8 つの列を指定できます。

インターリーブソートは、ソートキーの各列または列のサブセットに等しい重みを割り当てるため、クエリはソートキーの列の順序に依存しません。クエリが 1 つ以上のセカンダリソート列を使用すると、インターリーブソートによってクエリのパフォーマンスは大幅に向上します。インターリーブソートでは、データのロード操作とバキューム操作にわずかなオーバーヘッドコストがかかります。

AS query

Amazon Redshift がサポートするすべてのクエリ (SELECT ステートメント)。

CTAS の使用に関する注意事項

制限

Amazon Redshift は、ノードタイプごとにクラスターあたりのテーブル数のクォータを適用します。

テーブルの最大文字数は 127 です。

1 つのテーブルで定義できる列の最大数は 1,600 です。

列属性とテーブル属性の継承

CREATE TABLE AS (CTAS) テーブルは、作成元のテーブルから制約、ID 列、デフォルトの列値、またはプライマリキーを継承しません。

CTAS テーブルの列圧縮エンコードは指定できません。Amazon Redshift では、次のように圧縮エンコードが自動的に割り当てられます。

- ソートキーとして定義されている列には、RAW 圧縮が割り当てられます。
- BOOLEAN、REAL、DOUBLE PRECISION、GEOMETRY、もしくは GEOGRAPHY の各データ型で定義されている列には、RAW 圧縮が割り当てられます。
- SMALLINT、INTEGER、BIGINT、DECIMAL、DATE、TIME、TIMETZ、TIMESTAMP、または TIMESTAMPTZ として定義された列には AZ64 圧縮が割り当てられます。
- CHAR、VARCHAR、または VARBYTE として定義されている列には、LZO 圧縮が割り当てられます。

詳細については、「圧縮エンコード」および「データ型」を参照してください。

列エンコーディングを明示的に割り当てるには、CREATE TABLE を使用します。

CTAS は、SELECT 句のクエリプランに基づいて新しいテーブルの分散スタイルとソートキーを決定します。

結合、集計、ORDER BY 句、LIMIT 句を含むクエリなど、複雑なクエリの場合、CTAS はクエリプランに基づいてベストエフォートで最適な分散スタイルとソートキーを選択します。

> **Note**
>
> 大きいデータセットや複雑なクエリで最良のパフォーマンスを得るために、テストの際には、通常のデータセットを使用することをお勧めします。

多くの場合、クエリプランを調べて、クエリオプティマイザがデータのソートと分散に使用する列 (ある場合) を確認することにより、CTAS が選択する分散キーとソートキーを予測できます。クエリプランの最上位ノードが単一のテーブルからのシンプルなシーケンシャルスキャンである場合 (XN Seq Scan)、CTAS は通常ソーステーブルの分散スタイルとソートキーを使用します。クエリプランの最上位ノードがシーケンシャルスキャン以外である場合 (XN Limit、XN Sort、XN HashAggregate など)、CTAS はクエリプランに基づいてベストエフォートで最適な分散スタイルとソートキーを選択します。

例えば、次のタイプの SELECT 句を使用して 5 つのテーブルを作成するとします。

- シンプルな SELECT ステートメント
- Limit 句
- LISTID を使用した ORDER BY 句
- QTYSOLD を使用した ORDER BY 句
- GROUP BY 句を使用した SUM 集計関数です。

次の例は、各 CTAS ステートメントのクエリプランを示しています。

```
explain create table sales1_simple as select listid, dateid, qtysold from sales;
                           QUERY PLAN
----------------------------------------------------------------
 XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(1 row)
```

```
explain create table sales2_limit as select listid, dateid, qtysold from sales limit
 100;
                              QUERY PLAN
----------------------------------------------------------------------
 XN Limit  (cost=0.00..1.00 rows=100 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)


explain create table sales3_orderbylistid as select listid, dateid, qtysold from sales
 order by listid;
                               QUERY PLAN
------------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: listid
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)


explain create table sales4_orderbyqty as select listid, dateid, qtysold from sales
 order by qtysold;
                               QUERY PLAN
------------------------------------------------------------------------
 XN Sort  (cost=1000000016724.67..1000000017155.81 rows=172456 width=8)
   Sort Key: qtysold
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(3 rows)


explain create table sales5_groupby as select listid, dateid, sum(qtysold) from sales
 group by listid, dateid;
                              QUERY PLAN
----------------------------------------------------------------------
 XN HashAggregate  (cost=3017.98..3226.75 rows=83509 width=8)
   ->  XN Seq Scan on sales  (cost=0.00..1724.56 rows=172456 width=8)
(2 rows)
```

各テーブルのディストリビューションキーとソートキーを表示するには、次に示すように PG_TABLE_DEF システムカタログテーブルのクエリを実行します。

```
select * from pg_table_def where tablename like 'sales%';
```

```
      tablename       |   column   | distkey | sortkey
----------------------+------------+---------+---------
 sales                | salesid    | f       |       0
 sales                | listid     | t       |       0
 sales                | sellerid   | f       |       0
 sales                | buyerid    | f       |       0
 sales                | eventid    | f       |       0
 sales                | dateid     | f       |       1
 sales                | qtysold    | f       |       0
 sales                | pricepaid  | f       |       0
 sales                | commission | f       |       0
 sales                | saletime   | f       |       0
 sales1_simple        | listid     | t       |       0
 sales1_simple        | dateid     | f       |       1
 sales1_simple        | qtysold    | f       |       0
 sales2_limit         | listid     | f       |       0
 sales2_limit         | dateid     | f       |       0
 sales2_limit         | qtysold    | f       |       0
 sales3_orderbylistid | listid     | t       |       1
 sales3_orderbylistid | dateid     | f       |       0
 sales3_orderbylistid | qtysold    | f       |       0
 sales4_orderbyqty    | listid     | t       |       0
 sales4_orderbyqty    | dateid     | f       |       0
 sales4_orderbyqty    | qtysold    | f       |       1
 sales5_groupby       | listid     | f       |       0
 sales5_groupby       | dateid     | f       |       0
 sales5_groupby       | sum        | f       |       0
```

結果の概要を次の表に示します。分かりやすいように、説明プランからコスト、行、幅の詳細を省略しています。

表	CTAS SELECT ステートメント	EXPLAIN プランの最上位ノード	分散キー	ソートキー
S1_SIMPLE	`select listid, dateid, qtysold from sales`	`XN Seq Scan on sales ...`	LISTID	DATEID

表	CTAS SELECT ステートメント	EXPLAIN プランの最上位ノード	分散キー	ソートキー
S2_LIMIT	`select listid, dateid, qtysold from sales limit 100`	`XN Limit ...`	なし (EVEN)	なし
S3_ORDER_BY_LISTID	`select listid, dateid, qtysold from sales order by listid`	`XN Sort ...` `Sort Key: listid`	LISTID	LISTID
S4_ORDER_BY_QTY	`select listid, dateid, qtysold from sales order by qtysold`	`XN Sort ...` `Sort Key: qtysold`	LISTID	QTYSOLD
S5_GROUP_BY	`select listid, dateid, sum(qtysold) from sales group by listid, dateid`	`XN HashAggregate ...`	なし (EVEN)	なし

CTAS ステートメントでは、分散スタイルとソートキーを明示的に指定できます。例えば、次のステートメントは、EVEN 分散を使用してテーブルを作成し、ソートキーとして SALESID を指定します。

```
create table sales_disteven
diststyle even
sortkey (salesid)
as
select eventid, venueid, dateid, eventname
from event;
```

圧縮エンコード

ENCODE AUTO は、テーブルのデフォルトになっています。Amazon Redshift は、テーブル内のすべての列の圧縮エンコードを自動的に管理します。

受信データの分散

受信データのハッシュ分散スキームが、ターゲットテーブルのスキームと同じ場合、データをロードするときに、データを物理的に分散させる必要はありません。例えば、新しいテーブルに分散キーが設定されており、同じキー列で分散されている別のテーブルからデータが挿入される場合、同じノードとスライスを使用してデータが所定の位置にロードされます。ただし、ソーステーブルとターゲットテーブルの両方が EVEN 分散に設定されている場合、データはターゲットテーブルで再分散されます。

自動 ANALYZE 操作

Amazon Redshift は、CTAS コマンドで作成したテーブルを自動的に分析します。最初に作成したとき、これらのテーブルに ANALYZE コマンドを実行する必要はありません。変更する場合は、他のテーブルと同じように分析する必要があります。

CTAS の例

次の例では、EVENT テーブルに対して EVENT_BACKUP というテーブルを作成します。

```
create table event_backup as select * from event;
```

結果のテーブルは、EVENT テーブルから分散キーとソートキーを継承します。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'event_backup';

column    | type                        | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+--------
catid     | smallint                    | none     | false   |       0
dateid    | smallint                    | none     | false   |       1
eventid   | integer                     | none     | true    |       0
eventname | character varying(200)      | none     | false   |       0
starttime | timestamp without time zone | none     | false   |       0
venueid   | smallint                    | none     | false   |       0
```

次のコマンドでは、EVENT テーブルから 4 つの列を選択して、EVENTDISTSORT という新しいテーブルを作成します。新しいテーブルは EVENTID によって分散され、EVENTID と DATEID によってソートされます。

```
create table eventdistsort
distkey (1)
```

```
sortkey (1,3)
as
select eventid, venueid, dateid, eventname
from event;
```

結果は次のようになります。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistsort';

column    |          type          | encoding | distkey | sortkey
---------+-----------------------+----------+---------+-------
eventid   | integer                | none     | t       | 1
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 2
eventname | character varying(200)| none     | f       | 0
```

分散キーとソートキーの列名を使用することで、まったく同じテーブルを作成できます。次に例を示します。

```
create table eventdistsort1
distkey (eventid)
sortkey (eventid, dateid)
as
select eventid, venueid, dateid, eventname
from event;
```

次のステートメントは、テーブルに均等に分配適用されますが、明示的なソートキーを定義しません。

```
create table eventdisteven
diststyle even
as
select eventid, venueid, dateid, eventname
from event;
```

EVEN 分散は新しいテーブルで指定されるため、このテーブルは EVENT テーブル (EVENTID) からソートキーを継承しません。新しいテーブルにはソートキーと分散キーがありません。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdisteven';
```

```
column    |          type          | encoding | distkey | sortkey
----------+------------------------+----------+---------+---------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 0
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

次のステートメントでは、均等分散を適用し、ソートキーを定義します。

```
create table eventdistevensort diststyle even sortkey (venueid)
as select eventid, venueid, dateid, eventname from event;
```

結果のテーブルにはソートキーがありますが、分散キーはありません。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'eventdistevensort';

column    |          type          | encoding | distkey | sortkey
----------+------------------------+----------+---------+-------
eventid   | integer                | none     | f       | 0
venueid   | smallint               | none     | f       | 1
dateid    | smallint               | none     | f       | 0
eventname | character varying(200) | none     | f       | 0
```

次のステートメントでは、受信データに基づいて別のキー列で EVENT テーブルを再分散します。データは EVENTID 列に基づいてソートされており、SORTKEY 列は定義されません。そのため、テーブルはソートされません。

```
create table venuedistevent distkey(venueid)
as select * from event;
```

結果は次のようになります。

```
select "column", type, encoding, distkey, sortkey
from pg_table_def where tablename = 'venuedistevent';

 column   |            type             | encoding | distkey | sortkey
----------+-----------------------------+----------+---------+-------
eventid   | integer                     | none     | f       | 0
venueid   | smallint                    | none     | t       | 0
```

```
catid     | smallint                    | none     | f        | 0
dateid    | smallint                    | none     | f        | 0
eventname | character varying(200)      | none     | f        | 0
starttime | timestamp without time zone | none     | f        | 0
```

CREATE USER

新しいデータベースユーザーを作成します。データベースユーザーは、権限とロールに応じて、データベースでデータを取得したり、コマンドを実行したり、その他のアクションを実行したりできます。このコマンドを実行するには、データベースのスーパーユーザー権限を持つ必要があります。

必要な権限

以下に、CREATE USER に必要な権限を示します。

- スーパーユーザー
- CREATE USER の権限を持つユーザー

構文

```
CREATE USER name [ WITH ]
PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }
[ option [ ... ] ]

where option can be:

CREATEDB | NOCREATEDB
| CREATEUSER | NOCREATEUSER
| SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }
| IN GROUP groupname [, ... ]
| VALID UNTIL 'abstime'
| CONNECTION LIMIT { limit | UNLIMITED }
| SESSION TIMEOUT limit
| EXTERNALID external_id
```

パラメータ

name

作成するユーザーの名前。ユーザー名を PUBLIC にすることはできません。有効な名前の詳細については、「名前と識別子」を参照してください。

WiTH

オプションキーワード Amazon Redshift では WITH は無視されます

PASSWORD { 'password' | 'md5hash' | 'sha256hash' | DISABLE }

ユーザーのパスワードを設定します。

デフォルトでは、ユーザーはパスワードが無効になっていない限り、自分のパスワードを変更できます。ユーザーのパスワードを無効にするには、DISABLE を指定します。ユーザーのパスワードが無効になると、パスワードはシステムから削除され、ユーザーは AWS Identity and Access Management (IAM) ユーザーの一時的認証情報を使用してのみログオンできます。詳細については、「IAM 認証を使用したデータベースユーザー認証情報の生成」を参照してください。スーパーユーザーのみが、パスワードを有効または無効にできます。スーパーユーザーのパスワードを無効にすることはできません。パスワードを有効にするには、ALTER USERを実行し、パスワードを指定します。

パスワードの指定は、クリアテキスト、MD5 ハッシュ文字列、もしくは SHA256 ハッシュ文字列の形式で行うことができます。

> **Note**
>
> AWS Management Console、AWS CLI、または Amazon Redshift API を使用して新しいクラスターを起動する場合、初期データベースユーザーのパスワードはクリアテキストで指定する必要があります。ALTER USER を使用して、パスワードを後で変更できます。

クリアテキストでは、パスワードは以下の制約を満たす必要があります:

- 8 から 64 文字の長さにする必要があります。
- 少なくとも 1 つの大文字、1 つの小文字、および 1 つの数字を使用する必要があります。
- ASCII 文字 (ASCII コード 33～126) のうち、' (一重引用符)、" (二重引用符)、\、/、@ を除く任意の文字を使用できます。

クリアテキストで CREATE USER パスワード パラメータを通過するための安全な代替方法として、パスワードとユーザー名を含む文字列の MD5 ハッシュを指定できます。

> **Note**
>
> MD5 ハッシュ文字列を指定すると、CREATE USER コマンドが MD5 ハッシュ文字列の有効性を確認しますが、パスワード部分の文字列は検証されません。この場合、データ

ベースにログインするために利用できないパスワード (空の文字列など) を作成することができます。

MD5 パスワードを指定するには、以下のステップに従います:

1. パスワードとユーザー名を連結します。

   例えば、パスワードが `ez`、ユーザーが `user1` の場合、連結した文字列は `ezuser1` です。
2. 連結した文字列を 32 文字の MD5 ハッシュ文字列に変換します。ハッシュ文字列を作成するために任意の MD5 ユーティリティを使用できます。次の例では、Amazon Redshift MD5 関数 と連結演算子 (||) を使用し、32 文字の MD5 ハッシュ文字列を使用しています。

   ```
   select md5('ez' || 'user1');

   md5
   --------------------------------
   153c434b4b77c89e6b94f12c5393af5b
   ```
3. MD5 ハッシュ文字列の前に「md5」を連結し、md5hash 引数として連結した文字列を指定します。

   ```
   create user user1 password 'md5153c434b4b77c89e6b94f12c5393af5b';
   ```
4. サインイン認証情報を使用してデータベースにログオンします。

   この例では、パスワード `ez` を使用して `user1` としてログオンします。

もう 1 つの安全な方法は、パスワード文字列の SHA-256 ハッシュを指定することです。または、ダイジェストの作成に使用された、独自の有効な SHA-256 ダイジェストと 256 ビットのソルトを指定することもできます。

- Digest – ハッシュ関数の出力。
- Salt – ハッシュ関数の出力のパターンを減らすために、パスワードと組み合わされて、ランダムに生成されたデータ。

```
'sha256|Mypassword'
```

```
'sha256|digest|256-bit-salt'
```

次の例では、Amazon Redshift によりソルトの生成および管理を行っています。

```
CREATE USER admin PASSWORD 'sha256|Mypassword1';
```

次の例では、ダイジェストの作成に使用された有効な SHA-256 ダイジェストと 256 ビットのソルトが提供されます。

パスワードを指定して独自のソルトでハッシュするには、次の手順に従います。

1. 256 ビットのソルトを作成します。ソルトを取得するには、任意の 16 進文字列ジェネレータを使用して 64 文字の文字列を生成します。この例で、ソルトは `c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6` です。
2. FROM_HEX 関数を使用して、ソルトをバイナリに変換します。SHA2 関数にはソルトのバイナリ表現が必要であるためです。次のステートメントをご覧ください。

```
SELECT
 FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6');
```

3. CONCAT 関数を使用して、パスワードにソルトを追加します。この例で、パスワードは `Mypassword1` です。次のステートメントをご覧ください

```
SELECT
 CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038
```

4. SHA2 関数を使用して、パスワードとソルトの組み合わせからダイジェストを作成します。次のステートメントをご覧ください。

```
SELECT
 SHA2(CONCAT('Mypassword1',FROM_HEX('c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f95
 0);
```

5. これまでの手順のダイジェストとソルトを使用して、ユーザーを作成します。次のステートメントをご覧ください。

```
CREATE USER admin PASSWORD 'sha256|
821708135fcc42eb3afda85286dee0ed15c2c461d000291609f77eb113073ec2|
c721bff5d9042cf541ff7b9d48fa8a6e545c19a763e3710151f9513038b0f6c6';
```

6. サインイン認証情報を使用してデータベースにログオンします。

   この例では、パスワード `Mypassword1` を使用して `admin` としてログオンします。

ハッシュ関数を指定せずにプレーンテキストによりパスワードを設定した場合は、ユーザーネームをソルトとして使用して MD5 ダイジェストの生成が行われます。

CREATEDB | NOCREATEDB

CREATEDB オプションを使用すると、新規ユーザーはデータベースを作成できます。デフォルトは NOCREATEDB です。

CREATEUSER | NOCREATEUSER

CREATEUSER オプションを使用すると、CREATE USER を含め、データベースに関するすべての権限を持つスーパーユーザーが作成されます。デフォルトは NOCREATEUSER です。詳細については、「superuser」を参照してください。

SYSLOG ACCESS { RESTRICTED | UNRESTRICTED }

Amazon Redshift のシステムテーブルとビューに対するユーザーのアクセスレベルを指定する句です。

SYSLOG ACCESS RESTRICTED アクセス許可を持つ通常のユーザーは、ユーザーが表示できるシステムテーブルとビューで自分が生成した行のみを表示できます。デフォルトは RESTRICTED です。

SYSLOG ACCESS UNRESTRICTED アクセス許可を持つ通常のユーザーは、ユーザーが表示できるシステムテーブルとビューのすべての行 (別のユーザーが生成した行を含む) を表示できます。UNRESTRICTED を指定しても、スーパーユーザーが表示可能なテーブルへのアクセス権が、通常のユーザーに付与されるわけではありません。スーパーユーザーが表示可能なテーブルを表示できるのはスーパーユーザーだけです。

> **Note**
>
> システムテーブルに対する無制限のアクセス権限を付与されたユーザーは、別のユーザーが生成したデータへの可視性が提供されます。例えば、STL_QUERY と STL_QUERYTEXT には INSERT、UPDATE、および DELETE ステートメントのフルテキストが含まれており、ユーザーが生成した機密データがこれらに含まれている可能性があります。

SVV_TRANSACTIONS のすべての行は、すべてのユーザーが表示可能です。

詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

IN GROUP groupname

ユーザーが属する既存のグループ名を指定します。複数のグループ名を指定できます。

VALID UNTIL abstime

VALID UNTIL オプションでは、ユーザーのパスワードが無効になるまでの絶対時間を設定します。デフォルトでは、パスワードには期限がありません。

CONNECTION LIMIT { limit | UNLIMITED }

ユーザーが同時に開けるデータベース接続の最大数。この制限はスーパーユーザーには適用されません。同時接続の最大数を許可するには、UNLIMITED キーワードを使用します。データベースごとの接続数の制限が適用される場合もあります。詳細については、「CREATE DATABASE」を参照してください。デフォルトは UNLIMITED です。現在の接続を確認するには、STV_SESSIONSシステムビューに対してクエリを実行します。

> **Note**
>
> ユーザーとデータベースの両方の接続制限が適用される場合は、ユーザーが接続しようとしたときに、両方の制限内に未使用の接続スロットがなければなりません。

SESSION TIMEOUT limit

セッションが非アクティブまたはアイドル状態を維持する最大時間 (秒) です。指定できる範囲は 60 秒 (1 分) から 1,728,000 秒 (20 日) です。ユーザーに対しセッションタイムアウトが設定されていない場合は、クラスターの設定値が適用されます。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクォータと制限」を参照してください。

設定されたセッションタイムアウトは、新しいセッションにのみ適用されます。

開始時刻、ユーザー名、セッションタイムアウトなど、アクティブなユーザーセッションに関する情報を表示するには、STV_SESSIONSシステムビューを参照します。ユーザーセッションの履歴に関する情報を表示するには、STL_SESSIONSビューを参照します。セッションタイムアウト値など、データベースユーザーに関する情報を取得するには、SVL_USER_INFOビューを参照します。

EXTERNALID external_id

ID プロバイダーに関連付けられているユーザーの識別子。ユーザーはパスワードを無効にする必要があります。詳細については、「Amazon Redshift 用のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

使用に関する注意事項

デフォルトでは、すべてのユーザーは PUBLIC スキーマに対して、CREATE 権限と USAGE 権限を所有しています。ユーザーがデータベースの PUBLIC スキーマにオブジェクトを作成できないようにするには、REVOKE コマンドを使用してその権限を削除します。

IAM 認証情報を使用してデータベースのユーザー認証情報を作成する場合、一時的認証情報を使用してのみログオンできるスーパーユーザーを作成することをお勧めします。スーパーユーザーのパスワードを無効にすることはできませんが、ランダムに生成された MD5 ハッシュ文字列を使って不明なパスワードを作成することはできます。

```
create user iam_superuser password 'md5A1234567890123456780123456789012' createuser;
```

二重引用符で囲まれたユーザーネームの大文字と小文字は、enable_case_sensitive_identifier設定オプションの設定にかかわらず、常に保持されます。詳細については、「enable_case_sensitive_identifier」を参照してください。

例

次のコマンドでは、パスワード (abcD1234)、データベース作成権限、接続制限 (30) の dbuser という名前のユーザーを作成します。

```
create user dbuser with password 'abcD1234' createdb connection limit 30;
```

PG_USER_INFO カタログテーブルに対してクエリを実行し、データベースユーザーに関する詳細を表示します。

```
select * from pg_user_info;

 usename   | usesysid | usecreatedb | usesuper | usecatupd | passwd   | valuntil |
 useconfig | useconnlimit
-----------+----------+-------------+----------+-----------+----------+----------
+-----------+-------------
 rdsdb     |        1 | true        | true     | true      | ******** | infinity |
      |
 adminuser |      100 | true        | true     | false     | ******** |          |
      | UNLIMITED
 dbuser    |      102 | true        | false    | false     | ******** |          |
      | 30
```

次の例では、アカウントのパスワードは 2017 年 6 月 10 日まで有効です。

```
create user dbuser with password 'abcD1234' valid until '2017-06-10';
```

次の例では、特殊文字を含む、大文字と小文字が区別されるパスワードを持つユーザーを作成します。

```
create user newman with password '@AbC4321!';
```

MD5 パスワードにバックスラッシュ (「\」) を使用するには、ソースとなる文字列のバックスラッシュをバックスラッシュでエスケープします。次の例では、`slashpass`という名前のユーザーを作成し、バックスラッシュ一つ ('\') をパスワードとして使用しています。

```
select md5('\\'||'slashpass');

md5
--------------------------------
0c983d1a624280812631c5389e60d48c
```

md5 パスワードでユーザーを作成します。

```
create user slashpass password 'md50c983d1a624280812631c5389e60d48c';
```

次の例では、アイドルセッションのタイムアウトを 120 秒に設定しながら、`dbuser`という名前のユーザーを作成します。

```
CREATE USER dbuser password 'abcD1234' SESSION TIMEOUT 120;
```

次の例では、bob という名前のユーザーを作成します。名前空間は myco_aad です。これはサンプルのみです。コマンドを正常に実行するには、ID プロバイダーが登録されている必要があります。詳細については、「Amazon Redshift 用のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

```
CREATE USER myco_aad:bob EXTERNALID "ABC123" PASSWORD DISABLE;
```

CREATE VIEW

データベースにビューを作成します。このビューは物理的にマテリアライズされません。ビューを定義するクエリは、ビューがクエリで参照されるたびに実行されます。外部テーブルでビューを作成するには、WITH NO SCHEMA BINDING 句を含めます。

標準ビューを作成するには、基礎となるテーブルまたは基礎となるビューへのアクセスが必要です。標準ビューにクエリを実行するには、ビュー自体に対する選択のアクセス許可が必要ですが、基礎となるテーブルに対する選択のアクセス許可は必要ありません。別のスキーマのテーブルまたはビューを参照するビューを作成する場合や、マテリアライズドビューを参照するビューを作成する場合は、使用許可が必要です。遅延バインドビューにクエリを実行するには、遅延バインドビュー自体に対する選択のアクセス許可が必要です。また、遅延バインドビューの所有者が、参照先のオブジェクト (テーブル、ビュー、またはユーザー定義関数) に対する選択のアクセス許可を持っていることも確認します。遅延バインドビューの詳細については、「使用に関する注意事項」を参照してください。

必要なアクセス許可

CREATE VIEW を使用するには、次のいずれかのアクセス許可が必要です。

- CREATE [OR REPLACE] VIEW を使用してビューを作成するには
  - スーパーユーザー
  - CREATE [REPLACE] VIEW アクセス許可のあるユーザー
- CREATE OR REPLACE VIEW を使用して既存のビューを置き換えるには
  - スーパーユーザー
  - CREATE [OR REPLACE] VIEW アクセス許可のあるユーザー
  - ビューの所有者

ユーザーがユーザー定義関数を組み込んだビューにアクセスしたい場合は、その関数に対する EXECUTE アクセス許可が必要です。

構文

```
CREATE [ OR REPLACE ] VIEW name [ ( column_name [, ...] ) ] AS query
[ WITH NO SCHEMA BINDING ]
```

パラメータ

OR REPLACE

同じ名前のビューが既に存在する場合、ビューは置換されます。同じ列名をデータタイプを使用して同一の列セットを生成する新しいクエリでは、ビューの置換のみが可能です。CREATE OR REPLACE VIEW は、操作が完了するまで、読み書きのためにビューをロックします。

ビューが置き換えられると、所有権や付与された権限などの他のプロパティが保持されます。

name

ビューの名前。スキーマ名を指定すると (`myschema.myview` など)、指定したスキーマを使用してビューが作成されます。指定しない場合、現在のスキーマにビューが作成されます。ビュー名は、同じスキーマ内の他のビューやテーブルと異なる名前にする必要があります。

「#」で始まるビュー名を指定したビューは、現在のセッションでのみ表示される一時ビューとして作成されます。

有効な名前の詳細については、「名前と識別子」を参照してください。システムデータベース template0、template1、padb_harvest、または sys:internal にテーブルまたはビューを作成することはできません。

column_name

ビューの列に使用されるオプションの名前リスト。列名を指定しない場合、列名はクエリから取得されます。1 つの画面で定義できる列の最大数は 1,600 です。

query

テーブルに評価されるクエリ (SELECT ステートメントのフォーム)。このテーブルでは、ビューの列と行を定義します。

WITH NO SCHEMA BINDING

テーブルやユーザー定義関数など、基盤となるデータベースオブジェクトにバインドされていないことを示す句。その結果、ビューと参照先のオブジェクト間には依存関係がありません。参照先のオブジェクトが存在しない場合でも、ビューを作成できます。依存関係がないため、ビューに影響を与えることなく参照先のオブジェクトを削除または変更できます。Amazon Redshift は、ビューがクエリされるまで依存関係をチェックしません。遅延バインドビューの詳細を表示するには、PG_GET_LATE_BINDING_VIEW_COLS 関数を実行します。

WITH NO SCHEMA BINDING 句を含める場合、SELECT ステートメントで参照されるテーブルとビューは、スキーマ名で修飾する必要があります。スキーマは、参照されるテーブルが存在しない場合でも、ビューを作成するときに存在している必要があります。例えば、次のステートメントはエラーを返します。

```
create view myevent as select eventname from event
with no schema binding;
```

次のステートメントは正常に実行されます。

```
create view myevent as select eventname from public.event
with no schema binding;
```

> **Note**
>
> ビューの更新、ビューへの挿入、ビューからの削除を行うことはできません。

使用に関する注意事項

遅延バインドビュー

遅延バインドビューは、ビューのクエリが行われるまで、基礎となるデータベースオブジェクト (テーブルや他のビュー) などをチェックしません。その結果、ビューを削除して再作成することなく、基礎となるオブジェクトを変更または削除できます。基礎となるオブジェクトを削除した場合、遅延バインドビューへのクエリは失敗します。遅延バインドビューへのクエリで、存在しない基盤となるオブジェクトの列を参照している場合、クエリは失敗します。

遅延バインドビューの基礎となるテーブルまたはビューを削除し、再作成した場合、デフォルトのアクセス権限を持つ新しいオブジェクトが作成されます。ビューにクエリを実行するユーザー用に、基盤オブジェクトへのアクセス許可の付与が必要になる場合があります。

遅延バインドビューを作成するには、WITH NO SCHEMA BINDING 句を含めます。次の例では、スキーマバインドなしでビューを作成します。

```
create view event_vw as select * from public.event
with no schema binding;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | eventname     | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

次の例は、ビューを再作成せずに基礎となるテーブルを変更できることを示しています。

```
alter table event rename column eventname to title;
```

```
select * from event_vw limit 1;

eventid | venueid | catid | dateid | title          | starttime
--------+---------+-------+--------+---------------+--------------------
      2 |     306 |     8 |   2114 | Boris Godunov | 2008-10-15 20:00:00
```

遅延バインドビューでのみ Amazon Redshift Spectrum の外部テーブルを参照できます。遅延バインドビューの 1 つのアプリケーションは、Amazon Redshift と Redshift Spectrum テーブルの両方をクエリできます。例えば、UNLOAD コマンドを使用して古いデータを Amazon S3 にアーカイブすることができます。次に、Amazon S3 のデータを参照する Redshift Spectrum 外部テーブルを作成し、両方のテーブルをクエリするビューを作成します。次の例では、UNION ALL 句を使用して、Amazon Redshift SALES テーブルと Redshift Spectrum SPECTRUM.SALES テーブルを結合します。

```
create view sales_vw as
select * from public.sales
union all
select * from spectrum.sales
with no schema binding;
```

SPECTRUM.SALES など、Redshift Spectrum 外部テーブルの作成の詳細については、Amazon Redshift Spectrum の開始方法参照してください。

遅延バインドビューから標準ビューを作成すると、標準ビューの定義には、標準ビューが作成された時点での遅延バインドビューの定義が含まれます。遅延バインドビューの依存関係は追跡されないため、遅延バインドビューへの変更は標準ビューでは追跡されません。

遅延バインドビューの最新の定義を参照するように標準ビューを更新するには、標準ビューの作成に使用した最初のビュー定義で CREATE OR REPLACE VIEW を実行します。

次の例では、遅延バインドビューから標準ビューを作成します。

```
create view sales_vw_lbv as
select * from public.sales
with no schema binding;

show view sales_vw_lbv;
                            Show View DDL statement
--------------------------------------------------------------------------------
 create view sales_vw_lbv as select * from public.sales with no schema binding;
```

```
(1 row)

create view sales_vw as
select * from sales_vw_lbv;

show view sales_vw;
                                                Show View DDL statement
-----------------------------------------------------------------------------------------------
 SELECT sales_vw_lbv.price, sales_vw_lbv."region" FROM (SELECT sales.price,
 sales."region" FROM sales) sales_vw_lbv;
(1 row)
```

標準ビューの DDL ステートメントに示されている遅延バインドビューは、標準ビューの作成時に定義され、後で遅延バインドビューに変更を加えても更新されないことに注意してください。

例

サンプルコマンドでは、TICKIT データベースと呼ばれるオブジェクトとデータのサンプルセットを使用しています。詳細については、「サンプルデータベース」を参照してください。

次のコマンドでは、EVENT というテーブルから myevent というビューを作成します。

```
create view myevent as select eventname from event
where eventname = 'LeAnn Rimes';
```

次のコマンドでは、USERS というテーブルから myuser というビューを作成します。

```
create view myuser as select lastname from users;
```

次のコマンドでは、USERS というテーブルから myuser というビューを作成または置換します。

```
create or replace view myuser as select lastname from users;
```

次の例では、スキーマバインドなしでビューを作成します。

```
create view myevent as select eventname from public.event
with no schema binding;
```

DEALLOCATE

準備済みステートメントの割り当てを解除します。

構文

```
DEALLOCATE [PREPARE] plan_name
```

パラメータ

PREPARE

このキーワードはオプションであり、無視されます。

plan_name

割り当てを解除する準備済みステートメントの名前。

使用に関する注意事項

DEALLOCATE は、以前に準備した SQL ステートメントの割り当てを解除するために使用されます。準備済みステートメントの割り当てを明示的に解除しない場合、現在のセッションの終了時に割り当てが解除されます。準備済みステートメントの詳細については、「PREPARE」を参照してください。

以下の資料も参照してください。

EXECUTE, PREPARE

DECLARE

新しいカーソルを定義します。カーソルを使用して、大きなクエリセットの結果から、一度で数行を取得します。

カーソルの最初の行が取得されると、必要に応じて、結果セット全体がリーダーノード、メモリ内、またはディスク上にマテリアライズされます。大きな結果セットにカーソルを使用すると、パフォーマンスが低下する可能性があるため、可能な限り、別の方法を使用することをお勧めします。詳細については、「カーソルを使用するときのパフォーマンスに関する考慮事項」を参照してください。

カーソルは、トランザクションブロック内で宣言する必要があります。1 つのセッションで、同時に開くことができるカーソルは 1 つのみです。

詳細については、「FETCH」、「CLOSE」を参照してください。

構文

```
DECLARE cursor_name CURSOR FOR query
```

パラメータ

cursor_name

新しいカーソルの名前。

query

カーソルを作成する SELECT ステートメント。

DECLARE CURSOR の使用に関する注意事項

クライアントアプリケーションが ODBC 接続を使用し、クエリで作成される結果セットが大きすぎてメモリが足りなくなる場合、カーソルを使用して、結果セットをクライアントアプリケーションに渡すことができます。カーソルを使用すると、結果セット全体がリーダーノードでマテリアライズされ、クライアントが少しずつ結果を取得できるようになります。

> **Note**
>
> ODBC for Microsoft Windows でカーソルを有効にするには、Amazon Redshift で使用する ODBC DSN で [Use Declare/Fetch (宣言/フェッチを使用)] オプションを有効にします。マルチノードクラスターでは、ラウンドトリップを最小限に抑えるために、ODBC キャッシュサイズを設定し、ODBC DSN オプションダイアログの [Cache Size (キャッシュサイズ)] フィールドを 4,000 以上に設定することをお勧めします。単一ノードクラスターでは、キャッシュサイズを 1,000 に設定します。

カーソルを使用すると、パフォーマンスが低下する可能性があるため、可能な限り、別の方法を使用することをお勧めします。詳細については、「カーソルを使用するときのパフォーマンスに関する考慮事項」を参照してください。

Amazon Redshift のカーソルは、次の制限付きでサポートされています。

- 1 つのセッションで、同時に開くことができるカーソルは 1 つのみです。
- カーソルはトランザクション内 (BEGIN ... END) で使用する必要があります。

- すべてのカーソルの累積結果セットの最大サイズは、クラスターノードタイプに基づいて制限されます。より大きな結果セットが必要な場合は、XL または 8XL ノード構成にサイズ変更できます。

  詳細については、「カーソルの制約」を参照してください。

カーソルの制約

カーソルの最初の行が取得されると、結果セット全体がリーダーノードにマテリアライズされます。結果セットをメモリに格納できない場合、必要に応じてディスクに書き込まれます。リーダーノードの整合性を保護するために、Amazon Redshift はクラスターのノードタイプに基づいてすべてのカーソルの結果セットのサイズに制約を適用します。

次の表に、各タイプのクラスターノードの結果セットの最大合計サイズを示します。結果セットの最大サイズの単位は、メガバイトです。

ノードの種類	クラスターあたりの最大結果セット (MB)
DC2 Large 複数ノード	192,000
DC2 Large 単一ノード	8,000
DC2 8XL 複数ノード	3,200,000
RA3 16XL 複数ノード	14,400,000
RA3 4XL 複数ノード	3,200,000
RA3 XLPLUS 複数ノード	1,000,000
RA3 XLPLUS シングルノード	64,000
RA3 LARGE 複数ノード	240,000
RA3 LARGE 単一ノード	8,000
Amazon Redshift Serverless	150,000

クラスターのアクティブなカーソル設定を表示するには、スーパーユーザー権限で STV_CURSOR_CONFIGURATION システムテーブルに対してクエリを実行します。アクティブな

カーソルの状態を表示するには、STV_ACTIVE_CURSORSシステムテーブルに対してクエリを実行します。ユーザーは自分のカーソルの行のみを表示できますが、スーパーユーザーはすべてのカーソルを表示できます。

カーソルを使用するときのパフォーマンスに関する考慮事項

カーソルによって結果セット全体がリーダーノードでマテリアライズされてから、結果をクライアントに返す処理が始まるため、非常に大きな結果セットにカーソルを使用すると、パフォーマンスが低下する可能性があります。非常に大きな結果セットには、カーソルを使用しないことを強くお勧めします。アプリケーションが ODBC 接続を使用する場合など、状況によっては、カーソルのみが実行可能な解決策の場合があります。ただし、可能な限り、次の代替方法を使用することをお勧めします。

- UNLOAD を使用して大きなテーブルをエクスポートします。UNLOAD を使用すると、複数のコンピューティングノードが同時に機能し、Amazon Simple Storage Service のデータファイルに直接データを転送します。詳細については、「Amazon Redshift でのデータのアンロード」を参照してください。
- クライアントアプリケーションで JDBC の fetch size パラメータを設定します。JDBC 接続を使用し、クライアント側のメモリ不足エラーが発生する場合、JDBC の fetch size パラメータを設定することで、ユーザーが少量の結果セットを取得するように指定できます。詳細については、「JDBC フェッチサイズパラメータの設定」を参照してください。

DECLARE CURSOR の例

次の例では、LOLLAPALOOZA というカーソルを宣言し、Lollapalooza イベントの売り上げ情報を選択した後、カーソルを使用して結果セットから行を取得します。

```
-- Begin a transaction

begin;

-- Declare a cursor

declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';
```

```
-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

次の例では、テーブルのすべての結果を refcursor でループ処理しています。

```
CREATE TABLE tbl_1 (a int, b int);
INSERT INTO tbl_1 values (1, 2),(3, 4);

CREATE OR REPLACE PROCEDURE sp_cursor_loop() AS $$
DECLARE
    target record;
    curs1 cursor for select * from tbl_1;
BEGIN
    OPEN curs1;
    LOOP
        fetch curs1 into target;
        exit when not found;
        RAISE INFO 'a %', target.a;
    END LOOP;
    CLOSE curs1;
```

```
END;
$$ LANGUAGE plpgsql;

CALL sp_cursor_loop();

SELECT message
   from svl_stored_proc_messages
   where querytxt like 'CALL sp_cursor_loop()%';

  message
----------
      a 1
      a 3
```

DELETE

テーブルから行を削除します。

> **Note**
>
> 単一 SQL ステートメントの最大サイズは 16 MB です。

構文

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
DELETE [ FROM ] { table_name | materialized_view_name }
    [ { USING } table_name, ... ]
    [ WHERE condition ]
```

パラメータ

WITH 句

1 つ以上の common-table-expressions を指定する任意の句。「WITH 句」を参照してください。

FROM

FROM キーワードは、USING 句が指定されている場合を除き、オプションです。ステートメント `delete from event;` と `delete event;` は、EVENT テーブルからすべての行を削除する操作と同じです。

> **Note**
>
> テーブルからすべての行を削除するには、テーブルに対して TRUNCATE を実行します。TRUNCATE は DELETE よりもはるかに効率的であり、VACUUM および ANALYZE を必要としません。ただし、TRUNCATE では、その操作を実行するトランザクションがコミットされることに注意してください。

table_name

一時テーブルまたは永続的テーブル テーブルの所有者またはテーブルで DELETE 権限を持つユーザーのみが、テーブルから行を削除できます。

大きなテーブルで制限のない削除操作を実行するには、TRUNCATE コマンドを使用します。「TRUNCATE」を参照してください。

> **Note**
>
> テーブルから多数の行を削除した後:
>
> - ストレージ容量を再利用し、行を再ソートするため、テーブルにバキューム処理を実行します。
> - テーブルを分析して、クエリプランナーの統計情報を更新します。

materialized_view_name

マテリアライズドビュー。DELETE ステートメントは、マテリアライズドビューへのストリーミング取り込みに使用されるマテリアライズドビューで機能します。マテリアライズドビューの所有者またはマテリアライズドビューに対する DELETE 権限を持つユーザーのみが、マテリアライズドビューから行を削除できます。

ユーザーに IGNORE RLS 権限が付与されていない行レベルセキュリティ (RLS) ポリシーでは、ストリーミング取り込みに使用されるマテリアライズドビューで DELETE を実行することはできません。ただし、例外として、DELETE を実行するユーザーに IGNORE RLS が付与されていれば、DELETE は正常に実行されます。詳細については、「RLS ポリシーの所有権と管理」を参照してください。

USING table_name, ...。

USING キーワードは、WHERE 句の条件で追加のテーブルを参照するときに、テーブルリストを導入するために使用されます。例えば、次のステートメントでは、EVENT テーブルと SALES テーブルに対する結合条件を満たす EVENT テーブルから、すべての行を削除します。FROM リストで、SALES テーブル名を明示的に指定する必要があります。

```
delete from event using sales where event.eventid=sales.eventid;
```

USING 句でターゲットテーブル名を繰り返すと、DELETE 操作が自己結合を実行します。USING 構文で同じクエリを書く代わりに、WHERE 句でサブクエリを使用することもできます。

WHERE condition

削除対象を、条件を満たす行に制限するオプションの句。例えば、列に対する制限条件、結合条件、クエリ結果に基づく条件などがあります。クエリでは、DELETE コマンドのターゲットではないテーブルを参照できます。次に例を示します。

```
delete from t1
where col1 in(select col2 from t2);
```

条件を指定しない場合、テーブルのすべての行が削除されます。

例

CATEGORY テーブルからすべての行を削除します。

```
delete from category;
```

CATEGORY テーブルから CATID 値が 0～9 の行を削除します。

```
delete from category
where catid between 0 and 9;
```

LISTING テーブルから、SELLERID 値が SALES テーブルに存在しない行を削除します。

```
delete from listing
where listing.sellerid not in(select sales.sellerid from sales);
```

次の 2 つのクエリはいずれも、EVENT テーブルへの結合と CATID に対する追加の制限に基づいて、CATEGORY テーブルから 1 行を削除します。

```
delete from category
using event
where event.catid=category.catid and category.catid=9;
```

```
delete from category
where catid in
(select category.catid from category, event
where category.catid=event.catid and category.catid=9);
```

次のクエリは、mv_cities マテリアライズドビューからすべての行を削除します。この例で使用しているマテリアライズドビュー名はサンプルです。

```
delete from mv_cities;
```

DESC DATASHARE

ALTER DATASHARE を使用して追加されたデータ共有内にあるデータベースオブジェクトのリストを表示します。Amazon Redshift には、名前、データベース、スキーマ、テーブル、ビュー、関数のタイプが表示されます。

データ共有オブジェクトに関する追加情報は、システムビューを使用して確認できます。詳細については、「SVV_DATASHARE_OBJECTS」および「SVV_DATASHARES」を参照してください。

構文

```
DESC DATASHARE datashare_name [ OF [ ACCOUNT account_id ] NAMESPACE namespace_guid ]
```

パラメータ

datashare_name

データ共有の名前。

NAMESPACE namespace_guid

データ共有が使用する名前空間を指定する値。DESC DATAHSARE をコンシューマークラスターの管理者として実行する場合は、NAMESPACE パラメータを指定して、インバウンドのデータ共有を表示します。

ACCOUNT account_id

データ共有が属しているアカウントを指定する値。

使用に関する注意事項

コンシューマーマアカウントの管理者として、DESC DATASHARE を実行して AWS アカウント内でインバウンドのデータ共有を確認する場合は、NAMESPACE オプションを指定します。DESC DATASHARE を実行して AWS アカウント全体でインバウンドのデータ共有を確認する場合は、ACCOUNT オプションと NAMESPACE オプションを指定します。

例

次の例では、プロデューサクラスターのアウトバウンドデータ共有の情報を表示します。

```
DESC DATASHARE salesshare;

producer_account |          producer_namespace           | share_type  | share_name   |
 object_type |         object_name           |  include_new
-----------------+---------------------------------------+------------+-------------
+-------------+-----------------------------+--------------
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 TABLE       | public.tickit_sales_redshift |
 123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d  | OUTBOUND    |  salesshare  |
 SCHEMA      | public                        |   t
```

次の例では、コンシューマークラスターのインバウンドデータ共有の情報を表示します。

```
DESC DATASHARE salesshare of ACCOUNT '123456789012' NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';

 producer_account |          producer_namespace          | share_type | share_name |
 object_type |          object_name          |  include_new
------------------+--------------------------------------+-----------+------------
+-------------+-----------------------------+--------------
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 table       | public.tickit_sales_redshift |
 123456789012     | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d | INBOUND    | salesshare |
 schema      | public                        |
(2 rows)
```

DESC ID プロバイダー

ID プロバイダーに関する情報を表示します。スーパーユーザーのみが ID プロバイダーを記述できます。

構文

```
DESC IDENTITY PROVIDER identity_provider_name
```

パラメータ

identity_provider_name

ID プロバイダーの名前。

例

以下の例では、ID プロバイダーに関する情報を表示します。

```
DESC IDENTITY PROVIDER azure_idp;
```

サンプル出力。

```
  uid   |   name    | type  |              instanceid              | namespc |

                                                     params

                              | enabled
--------+-----------+-------+--------------------------------------+---------
+------------------------------------------------------------------------------------
+---------
 126692 | azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | aad     |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":'',
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

DETACH MASKING POLICY

既にアタッチされている動的データマスキングポリシーを列からデタッチします。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、マスキングポリシーをデタッチできます。

構文

```
DETACH MASKING POLICY policy_name
   ON { table_name }
   ( output_column_names )
   FROM { user_name | ROLE role_name | PUBLIC };
```

パラメータ

policy_name

デタッチするマスキングポリシーの名前。

table_name

マスキングポリシーをデタッチするテーブルの名前。

output_column_names

マスキングポリシーがアタッチされた列の名前。

user_name

マスキングポリシーがアタッチしされたユーザーの名前。

1 回の DETACH MASKING POLICY ステートメントで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

role_name

マスキングポリシーがアタッチされたロールの名前。

1 回の DETACH MASKING POLICY ステートメントで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

PUBLIC

ポリシーがテーブル内のすべてのユーザーにアタッチされたことを示します。

1 回の DETACH MASKING POLICY ステートメントで設定できるのは、user_name、role_name、PUBLIC のいずれか 1 つのみです。

DETACH RLS POLICY

テーブルの行レベルのセキュリティポリシーをテーブルで 1 つ以上のユーザーまたはロールからデタッチします。

スーパーユーザーとユーザー、または `sys:secadmin` ロールを持つロールはポリシーをデタッチできます。

構文

```
DETACH RLS POLICY policy_name ON [TABLE] table_name [, ...]
FROM { user_name | ROLE role_name | PUBLIC } [, ...]
```

パラメータ

policy_name

ポリシーの名前。

ON [TABLE] table_name [, ...]

行レベルのセキュリティポリシーがデタッチされたテーブルまたはビュー。

FROM { user_name | ROLE role_name | PUBLIC} [, ...]

ポリシーを指定した 1 つ以上のユーザーまたはロールからデタッチするかどうか指定します。

使用に関する注意事項

DETACH RLS POLICY ステートメントを使用するとき、次の点に注意してください。

- ポリシーを、関係、ユーザー、ロール、パブリックからデタッチできます。

例

次の例では、ロールからテーブルのポリシーをデタッチします。

```
DETACH RLS POLICY policy_concerts ON tickit_category_redshift FROM ROLE analyst, ROLE
 dbadmin;
```

DROP DATABASE

データベースを削除します。

トランザクションブロック (BEGIN ... END) 内で DROP DATABASE を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

構文

```
DROP DATABASE database_name
```

パラメータ

database_name

削除するデータベースの名前。dev、padb_harvest、template0、template1、sys:internal の各データベースを削除することはできません。また、現在のデータベースも削除できません。

外部データベースを削除するには、外部スキーマを削除します。詳細については、「DROP SCHEMA」を参照してください。

DROP DATABASE 使用に関する注意事項

DROP DATABASE ステートメントを使用する際には、次の点を考慮してください。

- 一般的に、AWS Data Exchangeデータ共有が含まれるデータベースを DROP DATABASE ステートメントを使用してドロップすることはお勧めしません。この操作を行うと、データ共有に対する AWS アカウント のアクセス権が失われます。このタイプの変更を実行すると、AWS Data Exchangeのデータ製品での使用条件に違反する可能性があります。

  次に、AWS Data Exchangeデータ共有を含むデータベースが削除された場合に発生するエラーの例を示します。

  ```
  DROP DATABASE test_db;
  ```

```
ERROR:   Drop of database test_db that contains ADX-managed datashare(s)
 requires session variable datashare_break_glass_session_var to be set to value
 'ce8d280c10ad41'
```

データベースの削除を許可するには、次の変数を設定した後に、DROP DATABASE 文を再度実行します。

```
SET datashare_break_glass_session_var to 'ce8d280c10ad41';
```

```
DROP DATABASE test_db;
```

この場合、Amazon Redshift は 1 回限り有効なランダム値を生成し、この値によりセッション変数の設定を行うことで、AWS Data Exchangeデータ共有を含むデータベースでの DROP DATABASE の使用を許可します。

例

次の例では、TICKIT_TEST という名前のデータベースを削除します。

```
drop database tickit_test;
```

DROP DATASHARE

データ共有を削除します。このコマンドを元に戻すことはできません。

データ共有を削除できるのは、スーパーユーザーまたはデータ共有の所有者だけです。

必要な権限

DROP DATASHAREに必要な権限を以下に示します。

- スーパーユーザー
- DROP DATASHARE の権限を持つユーザー
- データ共有の所有者

構文

```
DROP DATASHARE datashare_name;
```

パラメータ

datashare_name

削除するデータ共有の名前。

DROP DATASHARE の使用に関する注意事項

DROP DATASHARE ステートメントを使用する際には、次の点を考慮してください。

- 一般的に、AWS Data Exchangeデータ共有を DROP DATASHARE ステートメントを使用してドロップすることはお勧めしません。この操作を行うと、データ共有に対する AWS アカウント のアクセス権が失われます。このタイプの変更を実行すると、AWS Data Exchangeのデータ製品での使用条件に違反する可能性があります。

  次に、AWS Data Exchangeデータ共有を削除した際に発生するエラーの例を示します。

  ```
  DROP DATASHARE salesshare;
  ERROR:  Drop of ADX-managed datashare salesshare requires session variable
   datashare_break_glass_session_var to be set to value '620c871f890c49'
  ```

  AWS Data Exchange データ共有の削除を許可するには、次の変数を設定した後、DROP DATASHARE ステートメントを再度実行します。

  ```
  SET datashare_break_glass_session_var to '620c871f890c49';
  ```

  ```
  DROP DATASHARE salesshare;
  ```

  この場合、Amazon Redshift は 1 回限り有効なランダム値を生成し、この値でセッション変数を設定することで、AWS Data Exchangeデータ共有での DROP DATASHARE 使用を許可します。

例

次の例では、salesshareという名前のデータ共有を削除します。

```
DROP DATASHARE salesshare;
```

DROP EXTERNAL VIEW (プレビュー)

これは、プレビュー版の Amazon Redshift のデータカタログについて記載した暫定版ドキュメントです。ドキュメントと機能はどちらも変更されることがあります。この機能は、本番環境ではなくテストクラスターでのみ使用することをお勧めします。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

Amazon Redshift クラスターを [プレビュー] で作成して、Amazon Redshift の新機能をテストできます。これらの機能を本番稼働で使用したり、[プレビュー] クラスターを本稼働クラスターや別のトラックのクラスターに移動したりすることはできません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

[Preview] (プレビュー) でクラスターを作成するには

1. AWS Management Console にサインインして、https://console.aws.amazon.com/redshiftv2/ で Amazon Redshift コンソールを開きます。
2. ナビゲーションメニューで [Provisioned clusters dashboard] (プロビジョニングされたクラスターダッシュボード) を選択し、[Clusters] (クラスター) を選択します。現在の AWS リージョンにあるアカウントのクラスターがリストされています。各クラスターのプロパティのサブセットが、リストの列に表示されます。
3. [Clusters] (クラスター) リストページに、プレビューを紹介するバナーが表示されます。[Create preview cluster] (プレビュークラスターの作成) ボタンを選択して、クラスターの作成ページを開きます。
4. クラスターのプロパティを入力します。テストしたい機能を含む [プレビュートラック] を選択します。プレビュートラックにあることを示すクラスターの名前を入力することをお勧めします。テストする機能について、-preview というラベルの付いたオプションを含む、クラスターのオプションを選択します。クラスター作成の詳細については、「Amazon Redshift 管理ガイド」の「クラスターの作成」を参照してください。
5. [クラスターを作成] を選択して、プレビューのクラスターを作成します。

> **Note**
>
> `preview_2023` トラックは、利用可能な最新のプレビュートラックです。このトラックは RA3 ノードタイプのクラスターの作成のみをサポートしています。ノードタイプ DC2 以前のノードタイプはサポートされていません。

6. プレビュークラスターが使用可能になったら、SQL クライアントを使用してデータをロードし、クエリを実行します。

データカタログビューのプレビュー機能は以下のリージョンでのみ利用できます。

- 米国東部 (オハイオ) (us-east-2)
- 米国東部 (バージニア北部) (us-east-1)
- 米国西部 (北カリフォルニア) (us-west-1)
- アジアパシフィック (東京) (ap-northeast-1)
- 欧州 (アイルランド) (eu-west-1)
- 欧州 (ストックホルム) (eu-north-1)

データカタログビューをテストするためのプレビューワークグループを作成することもできます。これらの機能を本番稼働で使用したり、ワークグループを別のワークグループに移動したりすることはできません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。プレビューワークグループの作成方法については、「https://docs.aws.amazon.com/redshift/latest/mgmt/serverless-workgroup-preview.html」を参照してください。

データベースから外部ビューを削除します。外部ビューを削除すると、そのビューが関連付けられているすべての SQL エンジン (Amazon Athena や Amazon EMR Spark など) からそのビューが削除されます。このコマンドを元に戻すことはできません。データカタログビューの詳細については、「データカタログビューの作成 (プレビュー)」を参照してください。

構文

```
DROP EXTERNAL VIEW schema_name.view_name [ IF EXISTS ]
{catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name |
 external_schema_name.view_name}
```

パラメータ

schema_name.view_name

AWS Glue データベースにアタッチされているスキーマ。その後にビューの名前が続きます。

IF EXISTS

ビューが存在する場合にのみ、ビューを削除します。

catalog_name.schema_name.view_name | awsdatacatalog.dbname.view_name | external_schema_name.view_name

ビューを削除するときに使用するスキーマの表記法。AWS Glue Data Catalog、作成した Glue データベース、または作成した外部スキーマを使用するように指定できます。詳細については、「CREATE DATABASE」と「CREATE EXTERNAL SCHEMA 」を参照してください。

query_definition

Amazon Redshift がビューを変更するために実行する SQL クエリの定義。

例

次の例では、sample_schema.glue_data_catalog_view という名前のデータカタログビューを削除します。

```
DROP EXTERNAL VIEW sample_schema.glue_data_catalog_view IF EXISTS
```

DROP FUNCTION

データベースからユーザー定義関数 (UDF) を削除します。複数の関数が同じ名前でありながら異なる署名を持つことがあるため、関数の署名またはデータタイプの引数のリストを指定する必要があります。Amazon Redshift ビルトイン関数は削除できません。

このコマンドを元に戻すことはできません。

必要な権限

以下に、DROP FUNCTION に必要な権限を示します。

- スーパーユーザー
- DROP FUNCTION の権限を持つユーザー

- 関数の所有者

構文

```
DROP FUNCTION name
( [arg_name] arg_type   [, ...] )
[ CASCADE | RESTRICT ]
```

パラメータ

name

削除する関数の名前。

arg_name

入力引数の名前。引数データタイプのみが関数の識別を決定するために必要となるため、DROP FUNCTION は引数名を無視します。

arg_type

入力引数のデータタイプ。最大 32 までのデータタイプのカンマ区切りリストを指定できます。

CASCADE

ビューなど、関数に依存するオブジェクトを自動的に削除することを指定するキーワード。

関数に依存しないビューを作成するには、ビュー定義に WITH NO SCHEMA BINDING 句を含めます。詳細については、「CREATE VIEW」を参照してください。

RESTRICT

関数に依存するオブジェクトが、その関数を削除せず、メッセージを返すように指定するキーワード。この動作がデフォルトです。

例

次の例では、`f_sqrt`名の関数を削除します。

```
drop function f_sqrt(int);
```

依存性がある関数を削除するためには、次の例に示すように、CASCADE オプションを使用します。

```
drop function f_sqrt(int)cascade;
```

DROP GROUP

ユーザーグループを削除します。このコマンドを元に戻すことはできません。このコマンドでは、グループ内の個々のユーザーは削除されません。

個々のユーザーの削除については、DROP USER を参照してください。

構文

```
DROP GROUP name
```

パラメータ

name

削除するユーザーグループの名前。

例

次の例では、guests ユーザーグループを削除します。

```
DROP GROUP guests;
```

グループがオブジェクトに対して特権を持っている場合、そのグループを削除することはできません。このようなグループを削除しようとすると、以下のエラーが発生します。

```
ERROR: group "guests" can't be dropped because the group has a privilege on some object
```

グループが特定のオブジェクトに対して特権を持っている場合、グループを削除する前に、その特権を削除する必要があります。guests グループが権限を持っているオブジェクトを見つけるには、次の例を使用します。この例で使用されているメタデータビューの詳細については、「SVV_RELATION_PRIVILEGES」を参照してください。

```
SELECT DISTINCT namespace_name, relation_name, identity_name, identity_type
FROM svv_relation_privileges
WHERE identity_type='group' AND identity_name='guests';
```

```
+----------------+---------------+---------------+---------------+
| namespace_name | relation_name | identity_name | identity_type |
+----------------+---------------+---------------+---------------+
| public         | table1        | guests        | group         |
+----------------+---------------+---------------+---------------+
| public         | table2        | guests        | group         |
+----------------+---------------+---------------+---------------+
```

次の例は、publicユーザーグループから guests スキーマ内のすべてのテーブルに対するすべての特権を削除してから、グループを削除します。

```
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM GROUP guests;
DROP GROUP guests;
```

DROP ID プロバイダー

ID プロバイダーを削除する。このコマンドを元に戻すことはできません。スーパーユーザーのみが ID プロバイダーを削除できます。

構文

```
DROP IDENTITY PROVIDER identity_provider_name [ CASCADE ]
```

パラメータ

identity_provider_name

削除する ID プロバイダーの名前。

CASCADE

削除すると、ID プロバイダーにアタッチされているユーザーおよびロールを削除されます。

例

次の例では、oauth_provider ID プロバイダーを削除します。

```
DROP IDENTITY PROVIDER oauth_provider;
```

ID プロバイダーを削除すると、一部のユーザーはログインできないか、ID プロバイダーを使用するように構成されたクライアントツールを使用できない場合があります。

DROP LIBRARY

データベースからカスタム Python ライブラリを削除します。ライブラリの所有者またはスーパーユーザーのみがライブラリを削除できます。

DROP LIBRARY は、トランザクションブロック内で実行することはできません (BEGIN… END)。トランザクションの詳細については、「直列化可能分離」を参照してください。

このコマンドを元に戻すことはできません。DROP LIBRARY コマンドは、即座にコミットします。ライブラリに依存する UDF が同時に実行されている場合、UDF がトランザクション内で実行されていても、UDF は失敗する場合があります。

詳細については、「ライブラリを作成する」を参照してください。

必要な権限

以下に、DROP LIBRARY に必要な権限を示します。

- スーパーユーザー
- DROP LIBRARY の権限を持つユーザー
- ライブラリの所有者

構文

```
DROP LIBRARY library_name
```

パラメータ

library_name

ライブラリの名前。

DROP MASKING POLICY

すべてのデータベースから動的データマスキングポリシーを削除します。1 つ以上のテーブルにアタッチされているマスキングポリシーを削除することはできません。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールは、マスキングポリシーを削除できます。

構文

```
DROP MASKING POLICY policy_name;
```

パラメータ

policy_name

削除するマスキングポリシーの名前。

DROP MODEL

データベースからモデルを削除します。モデルの所有者またはスーパーユーザーのみがモデルを削除できます。

DROP MODEL は、このモデルから派生したすべての関連する予測関数、モデルに関連するすべての Amazon Redshift アーティファクト、およびモデルに関連するすべての Amazon S3 データも削除します。モデルが Amazon SageMaker でトレーニングを受けている間、DROP MODEL はこれらのオペレーションをキャンセルします。

このコマンドを元に戻すことはできません。DROP MODEL コマンドは、即座にコミットします。

必要なアクセス許可

DROP MODEL に必要なアクセス許可を以下に示します。

- スーパーユーザー
- DROP MODEL のアクセス許可を持つユーザー
- モデルの所有者
- スキーマの所有者

構文

```
DROP MODEL [ IF EXISTS ] model_name
```

パラメータ

IF EXISTS

指定されたスキーマが既に存在する場合、コマンドは何も変更しないで、スキーマが存在するというメッセージを返す必要があることを示す句。

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

例

次の例では、モデル demo_ml.customer_churn を削除します。

```
DROP MODEL demo_ml.customer_churn
```

DROP MATERIALIZED VIEW

マテリアライズドビューを削除します。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

構文

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [ CASCADE | RESTRICT ]
```

パラメータ

IF EXISTS

名前付きマテリアライズドビューが存在するかどうかをチェックすることを指定する句。マテリアライズドビューが存在しない場合、DROP MATERIALIZED VIEWコマンドは、エラーメッセージを返します。この句は、スクリプトの記述中に、存在しないマテリアライズドビューを削除した場合にスクリプトが失敗しないようにするために有用です。

mv_name

削除するマテリアライズドビューの名前。

CASCADE

その他のビューなど、マテリアライズドビューが依存するオブジェクトを自動的に削除することを示す句。

RESTRICT

ビューに依存するオブジェクトがある場合、マテリアライズドビューを削除しないことを示す句。これがデフォルトです。

使用に関する注意事項

マテリアライズドビューの所有者だけが、そのビューで `DROP MATERIALIZED VIEW` を使用できます。スーパーユーザーや DROP 権限を特別に付与されたユーザーはこの例外となる場合があります。

マテリアライズドビューの drop ステートメントを記述し、一致する名前のビューが存在すると、DROP VIEW を使用するように指示するエラーが発生します。`DROP MATERIALIZED VIEW IF EXISTS` を使用する場合でもエラーが発生します。

例

次の例では、マテリアライズドビュー `tickets_mv` を削除します。

```
DROP MATERIALIZED VIEW tickets_mv;
```

DROP PROCEDURE

プロシージャを削除します。プロシージャを削除するには、プロシージャ名と入力引数のデータタイプ (署名) の両方が必要です。必要に応じて、OUT 引数も含めて、完全な引数のデータタイプを指定できます。プロシージャの署名を検索するには、SHOW PROCEDURE コマンドを使用します。プロシージャの署名の詳細については、「PG_PROC_INFO」を参照してください。

必要な権限

以下に、DROP PROCEDURE に必要な権限を示します。

- スーパーユーザー
- DROP PROCEDURE の権限を持つユーザー
- プロシージャの所有者

構文

```
DROP PROCEDURE sp_name ( [ [ argname ] [ argmode ] argtype [, ...] ] )
```

パラメータ

sp_name

削除するプロシージャの名前。

argname

入力引数の名前。プロシージャの識別に必要なのは引数データタイプのみであるため、DROP PROCEDURE は引数名を無視します。

argmode

引数のモード。IN、OUT、INOUT のいずれかです。OUT 引数は、ストアドプロシージャの識別に使用されないため、省略可能です。

argtype

入力引数のデータタイプ。サポートされているデータ型のリストについては、「データ型」を参照してください。

例

次の例では、`quarterly_revenue`というストアドプロシージャを削除します。

```
DROP PROCEDURE quarterly_revenue(volume INOUT bigint, at_price IN numeric,result OUT
 int);
```

DROP RLS POLICY

すべてのデータベースにあるすべてのテーブルの行レベルのセキュリティポリシーを削除します。

スーパーユーザーと sys:secadmin ロールを持つユーザーまたはロールはポリシーを削除できます。

構文

```
DROP RLS POLICY [ IF EXISTS ] policy_name [ CASCADE | RESTRICT ]
```

パラメータ

IF EXISTS

指定されたポリシーが既に存在するかどうか示す句。

policy_name

ポリシーの名前。

CASCADE

ポリシーを削除する前に、アタッチされているすべてのテーブルからポリシーを自動的にデタッチすることを示す句。

RESTRICT

ポリシーが一部のテーブルにアタッチされている場合、ポリシーを削除しないことを示す句。これがデフォルトです。

例

次の例では、行レベルのセキュリティポリシーを削除します。

```
DROP RLS POLICY policy_concerts;
```

DROP ROLE

データベースからロールを削除します。ロールを削除できるのは、そのロールを作成した所有者、WITH ADMIN オプションを使用しているユーザー、またはスーパーユーザーのみです。

ユーザーに付与されているロールや、別のロールが依存しているロールを削除することはできません。

必要な権限

以下に、DROP ROLE に必要な権限を示します。

- スーパーユーザー
- ロールを作成した所有者ユーザー、または WITH ADMIN OPTION の権限があるロールを付与された所有者ユーザー。

構文

```
DROP ROLE role_name [ FORCE | RESTRICT ]
```

パラメータ

role_name

ロールの名前。

[FORCE | RESTRICT]

デフォルトの設定は RESTRICT です。削除しようとしたロールが別のロールを継承している場合、Amazon Redshift はエラーをスローします。ロールの割り当てが存在する場合、その割り当てをすべて削除するには FORCE を使用します。

例

次の例では、ロール `sample_role` を削除しています。

```
DROP ROLE sample_role FORCE;
```

次の例では、デフォルトの RESTRICT オプションを使用してユーザーに付与されたロール sample_role1 の削除を試みます。

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
DROP ROLE sample_role1;
ERROR:  cannot drop this role since it has been granted on a user
```

このようにユーザーに付与された sample_role1 を正常に削除するには、FORCE オプションを使用します。

```
DROP ROLE sample_role1 FORCE;
```

次の例では、別の (デフォルトの RESTRICT オプションを使用している) ロールからの依存関係がある、ロール sample_role2 を削除しようと試みます。

```
CREATE ROLE sample_role1;
```

```
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO sample_role2;
DROP ROLE sample_role2;
ERROR:  cannot drop this role since it depends on another role
```

依存する別のロールを持つ sample_role2 を正常に削除するには、FORCE オプションを使用します。

```
DROP ROLE sample_role2 FORCE;
```

DROP SCHEMA

スキーマを削除します。外部スキーマの場合は、そのスキーマに関連付けられている外部データベースも削除できます。このコマンドを元に戻すことはできません。

必要な権限

DROP SCHEMA に必要な権限を以下に示します。

- スーパーユーザー
- スキーマの所有者
- DROP SCHEMA の権限を持つユーザー

構文

```
DROP SCHEMA [ IF EXISTS ] name [, ...]
[ DROP EXTERNAL DATABASE ]
[ CASCADE | RESTRICT ]
```

パラメータ

IF EXISTS

指定されたスキーマが存在しない場合、コマンドはエラーで終了するのではなく、何も変更しないで、スキーマが存在しないというメッセージを返すことを示す句。

この句は、存在しないスキーマに対して DROP SCHEMA を実行してもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

name

削除するスキーマの名前。複数のスキーマ名をカンマで区切って指定できます。

DROP EXTERNAL DATABASE

外部スキーマが削除された場合、その外部スキーマに関連付けられている外部データベースがあれば、それを削除することを示す句。外部データベースが存在しない場合、コマンドは外部データベースが存在しないことを示すメッセージを返します。複数の外部スキーマが削除されると、指定されたスキーマに関連付けられているすべてのデータベースが削除されます。

外部データベースにテーブルなどの依存オブジェクトが含まれている場合は、CASCADE オプションを指定して、その依存オブジェクトも削除します。

外部データベースを削除すると、そのデータベースに関連付けられている他の外部スキーマにおいても、そのデータベースが削除されます。そのデータベースを使用する他の外部スキーマで定義されているテーブルも削除されます。

DROP EXTERNAL DATABASE は、HIVE メタストアに格納されている外部データベースをサポートしません。

CASCADE

スキーマ内のすべてのオブジェクトを自動的に削除することを示すキーワード。DROP EXTERNAL DATABASE が指定されると、外部データベース内のすべてのオブジェクトも削除されます。

RESTRICT

スキーマまたは外部データベースにオブジェクトが含まれている場合は、スキーマまたは外部データベースを削除しないことを示すキーワード。この動作がデフォルトです。

例

次の例では、S_SALES というスキーマを削除します。この例では、オブジェクトを含むスキーマが削除されないように安全策として RESTRICT を使用しています。この場合、スキーマを削除する前にスキーマオブジェクトを削除する必要があります。

```
drop schema s_sales restrict;
```

次の例では、S_SALES というスキーマと、そのスキーマに依存するすべてのオブジェクトを削除します。

```
drop schema s_sales cascade;
```

次の例では、S_SALES スキーマが存在する場合は削除し、存在しない場合は何もせずにメッセージを返します。

```
drop schema if exists s_sales;
```

次の例では、S_SPECTRUM という名前の外部スキーマとそれに関連付けられている外部データベースを削除します。この例では RESTRICT を使用しているため、スキーマとデータベースにオブジェクトが含まれている場合は、スキーマとデータベースが削除されません。この場合、スキーマとデータベースを削除する前に依存オブジェクトを削除する必要があります。

```
drop schema s_spectrum drop external database restrict;
```

次の例では、複数のスキーマとそれらに関連付けられている外部データベースを、依存オブジェクトとともに削除します。

```
drop schema s_sales, s_profit, s_revenue drop external database cascade;
```

DROP TABLE

データベースからテーブルを削除します。

テーブルを削除せずに、テーブルの行を空にする場合、DELETE または TRUNCATE コマンドを使用します。

DROP TABLE を使用すると、ターゲットテーブルに存在する制約が削除されます。1 つの DROP TABLE コマンドで複数のテーブルを削除できます。

DROP TABLE と外部テーブルは、トランザクション内 (BEGIN … END) で実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

DROP 権限がグループに付与される例を見つけるには、「GRANT 例」を参照してください。

必要な権限

DROP TABLE に必要な権限を以下に示します。

- スーパーユーザー
- DROP TABLE の権限を持つユーザー

- スキーマに対する USAGE 権限を持つテーブル所有者

構文

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

パラメータ

IF EXISTS

指定されたテーブルが存在しない場合、コマンドはエラーで終了するのではなく、何も変更しないで、テーブルが存在しないというメッセージを返すことを示す句。

この句は、存在しないテーブルに対して DROP TABLE を実行してもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

name

削除するテーブルの名前。

CASCADE

ビューなどのテーブルに依存するオブジェクトを自動的に削除することを示す句。

ビューやテーブルなど、他のデータベースオブジェクトに依存しないビューを作成するには、ビュー定義に WITH NO SCHEMA BINDING 句を含めます。詳細については、「CREATE VIEW」を参照してください。

RESTRICT

テーブルに依存するオブジェクトがある場合、テーブルを削除しないことを示す句。この動作がデフォルトです。

例

依存するオブジェクトがないテーブルの削除

次の例では、依存するオブジェクトがない FEEDBACK というテーブルを作成して、削除します。

```
create table feedback(a int);

drop table feedback;
```

ビューまたは他のテーブルによって参照される列がテーブルに含まれている場合は、次のようなメッセージが Amazon Redshift に表示されます。

```
Invalid operation: cannot drop table feedback because other objects depend on it
```

2 つのテーブルの同時削除

次のコマンドセットでは、FEEDBACK テーブルと BUYERS テーブルを作成し、1 つのコマンドで両方のテーブルを削除します。

```
create table feedback(a int);

create table buyers(a int);

drop table feedback, buyers;
```

依存関係を持つテーブルの削除

次の手順では、CASCADE スイッチを使用して、FEEDBACK というテーブルを削除する方法について説明します。

まず、CREATE TABLE コマンドを使用して、FEEDBACK という単純なテーブルを作成します。

```
create table feedback(a int);
```

次に、CREATE VIEW コマンドを使用して、テーブル FEEDBACK に依存する FEEDBACK_VIEW というビューを作成します。

```
create view feedback_view as select * from feedback;
```

次の例では、テーブル FEEDBACK を削除し、ビュー FEEDBACK_VIEW も削除します。これは、FEEDBACK_VIEW がテーブル FEEDBACK に依存しているためです。

```
drop table feedback cascade;
```

テーブルの依存関係の表示

テーブルの依存関係を返すには、次の例を使用します。*my_schema* と *my_table* を独自のスキーマとテーブルに置き換えます。

```
SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
LIMIT 10;
```

my_table とその依存関係を削除するには、次の例を使用します。この例では、削除されたテーブルのすべての依存関係も返されます。

```
DROP TABLE my_table CASCADE;

SELECT dependent_ns.nspname as dependent_schema
, dependent_view.relname as dependent_view
, source_ns.nspname as source_schema
, source_table.relname as source_table
, pg_attribute.attname as column_name
FROM pg_depend
JOIN pg_rewrite ON pg_depend.objid = pg_rewrite.oid
JOIN pg_class as dependent_view ON pg_rewrite.ev_class = dependent_view.oid
JOIN pg_class as source_table ON pg_depend.refobjid = source_table.oid
JOIN pg_attribute ON pg_depend.refobjid = pg_attribute.attrelid
    AND pg_depend.refobjsubid = pg_attribute.attnum
JOIN pg_namespace dependent_ns ON dependent_ns.oid = dependent_view.relnamespace
JOIN pg_namespace source_ns ON source_ns.oid = source_table.relnamespace
WHERE
source_ns.nspname = 'my_schema'
AND source_table.relname = 'my_table'
AND pg_attribute.attnum > 0
ORDER BY 1,2
```

```
LIMIT 10;

+------------------+----------------+--------------+--------------+-------------+
| dependent_schema | dependent_view | source_schema | source_table | column_name |
+------------------+----------------+--------------+--------------+-------------+
```

IF EXISTS を使用したテーブルの削除

次の例では、FEEDBACK テーブルが存在する場合は削除し、存在しない場合は何もしないでメッセージを返します。

```
drop table if exists feedback;
```

DROP USER

データベースからユーザーを削除します。1 つの DROP USER コマンドで複数のユーザーを削除できます。このコマンドを実行するには、データベースのスーパーユーザーであるか、DROP USER アクセス許可が必要です。

構文

```
DROP USER [ IF EXISTS ] name [, ... ]
```

パラメータ

IF EXISTS

指定したユーザーが存在しない場合、コマンドはエラーで終了せずに、何も変更しないで、ユーザーが存在しない旨のメッセージを返すことを示す句。

この句は、存在しないユーザーに対して DROP USER を実行してもスクリプトがエラーにならないため、スクリプトを実行する際に便利です。

name

削除するユーザーの名前。各ユーザー名をコンマで区切って、複数のユーザーを指定できます。

使用に関する注意事項

`rdsdb` という名前のユーザーまたはデータベースの管理者ユーザー (通常は、`awsuser` または `admin` という名前) は削除できません。

スキーマ、データベース、テーブル、ビューなどのデータベースオブジェクトを所有するユーザーと、データベース、テーブル、列、またはグループに対して権限を持っているユーザーは削除できません。このようなユーザーを削除しようとすると、以下のいずれかのエラーが発生します。

```
ERROR: user "username" can't be dropped because the user owns some object [SQL
 State=55006]

ERROR: user "username" can't be dropped because the user has a privilege on some object
 [SQL State=55006]
```

データベースユーザーが所有するオブジェクトを検索する方法の詳細については、ナレッジセンターの「How do I resolve the "user cannot be dropped" error in Amazon Redshift?」(Amazon Redshift の「ユーザーを削除できません」エラーを解決する方法を教えてください) を参照してください。

> **Note**
>
> Amazon Redshift は、ユーザーを削除する前に、現在のデータベースのみを確認します。ユーザーがデータベースオブジェクトを所有しているか、別のデータベースのオブジェクトの権限を持っている場合、DROP USER はエラーを返しません。別のデータベースのオブジェクトを所有しているユーザーを削除すると、それらのオブジェクトの所有者は「不明」に変更されます。

ユーザーがオブジェクトを所有する場合は、元のユーザーを削除する前に、まずオブジェクトを削除するか、所有者を別のユーザーに変更します。ユーザーがオブジェクトに対する権限を持つ場合は、ユーザーを削除する前に、まずその権限を削除します。次の例は、ユーザーを削除する前にオブジェクトの削除、所有者の変更、および権限の取り消しを行います。

```
drop database dwdatabase;
alter schema dw owner to dwadmin;
revoke all on table dwtable from dwuser;
drop user dwuser;
```

例

次の例では、paulo というユーザーを削除します。

```
drop user paulo;
```

次の例では、paulo と martha という 2 人のユーザーを削除します。

```
drop user paulo, martha;
```

次の例では、paulo というユーザーが存在する場合はそれを削除し、存在しない場合は何もしないでメッセージを返します。

```
drop user if exists paulo;
```

DROP VIEW

データベースからビューを削除します。1 つの DROP VIEW コマンドで複数のビューを削除できます。このコマンドを元に戻すことはできません。

必要な権限

DROP VIEW に必要な権限を以下に示します。

- スーパーユーザー
- DROP VIEW の権限を持つユーザー
- ビューの所有者

構文

```
DROP VIEW [ IF EXISTS ] name [, ... ] [ CASCADE | RESTRICT ]
```

パラメータ

IF EXISTS

指定されたビューが存在しない場合、コマンドはエラーで終了するのではなく、何も変更しないで、ビューが存在しないというメッセージを返すことを示す句。

この句は、存在しないビューに対して DROP VIEW を実行してもスクリプトが失敗しないため、スクリプトを作成する際に便利です。

name

削除するビューの名前。

CASCADE

その他のビューなど、ビューに依存するオブジェクトを自動的に削除することを示す句。

ビューやテーブルなど、他のデータベースオブジェクトに依存しないビューを作成するには、ビュー定義に WITH NO SCHEMA BINDING 句を含めます。詳細については、「CREATE VIEW」を参照してください。

CASCADE を含めることで、データベースオブジェクトの削除数が 10 以上になった場合、データベースクライアントはすべての削除されたオブジェクトをサマリー結果に一覧表示しない可能性があることに注意してください。この原因は、通常、SQL クライアントツールには、返される結果に対するデフォルトの制限があるためです。

RESTRICT

ビューに依存するオブジェクトがある場合、ビューを削除しないことを示す句。この動作がデフォルトです。

例

次の例では、event というビューを削除します。

```
drop view event;
```

依存するオブジェクトがあるビューを削除するには、CASCADE オプションを使用します。例えば、EVENT というテーブルを使用するとします。次に、以下の例に示すように、CREATE VIEW コマンドを使用して、EVENT テーブルの eventview ビューを作成します。

```
create view eventview as
select dateid, eventname, catid
from event where catid = 1;
```

ここで myeventview という 2 つ目のビューを作成します。このビューは、最初のビュー eventview に基づいています。

```
create view myeventview as
select eventname, catid
from eventview where eventname <> ' ';
```

この時点で、eventview および myeventview という 2 つのビューが作成されています。

myeventview ビューは、親が eventview である子ビューです。

eventview ビューを削除するために使用するコマンドは、次のコマンドであることは明白です。

```
drop view eventview;
```

しかしこの場合、このコマンドを実行すると、次のエラーが返されます。

```
drop view eventview;
ERROR: can't drop view eventview because other objects depend on it
HINT: Use DROP ... CASCADE to drop the dependent objects too.
```

この問題を回避するために、(エラーメッセージの指示に従って) 次のコマンドを実行します。

```
drop view eventview cascade;
```

今度は eventview と myeventview の両方が正常に削除されました。

次の例では、eventview というビューが存在する場合はそれを削除し、存在しない場合は何もせずにメッセージを返します。

```
drop view if exists eventview;
```

END

現在のトランザクションをコミットします。COMMIT コマンドと同じ機能です。

詳細については、「COMMIT」を参照してください。

構文

```
END [ WORK | TRANSACTION ]
```

パラメータ

WORK

　オプションキーワード

TRANSACTION

　オプションキーワード。WORK と TRANSACTION は同義語です。

例

次の例はすべて、トランザクションブロックを終了し、トランザクションをコミットします。

```
end;
```

```
end work;
```

```
end transaction;
```

Amazon Redshift では、次のどのコマンドを実行しても、トランザクションブロックが終了し、変更がコミットされます。

EXECUTE

事前に準備したステートメントを実行します。

構文

```
EXECUTE plan_name [ (parameter [, ...]) ]
```

パラメータ

plan_name

実行される準備済みステートメントの名前です。

parameter

準備済みステートメントに対するパラメータの実際の値。準備済みステートメントを作成した PREPARE コマンドで、このパラメータの位置に指定されているデータ型と互換性がある型の値に評価される式にする必要があります。

使用に関する注意事項

EXECUTE は、事前に準備したステートメントを実行するために使用されます。準備されたステートメントの存在期間は 1 つのセッションのみに限られるため、準備済みステートメントは、その時点のセッションより前に実行した PREPARE ステートメントで作成しておく必要があります。

前の PREPARE ステートメントでいくつかのパラメータを指定した場合、互換性のあるパラメータセットを EXECUTE ステートメントに渡す必要があります。そうしないと、Amazon Redshift からエラーが返されます。関数とは異なり、準備済みステートメントは、指定したパラメータの種類または数によって過負荷になることはありません。準備済みステートメントの名前は、データベースセッション内で一意にする必要があります。

準備済みステートメントに対して EXECUTE コマンドを発行すると、Amazon Redshift は (指定されたパラメータ値に基づいてパフォーマンスを改善するように) 必要に応じてクエリ実行計画を修正してから、その準備済みステートメントを実行することがあります。また、準備済みステートメントを新しく実行するたびに、EXECUTE ステートメントを使用して指定した異なるパラメータ値に基づいて、Amazon Redshift はクエリ実行計画を修正することがあります。Amazon Redshift が特定の EXECUTE ステートメントに対して選択したクエリ実行計画を確認するには、EXPLAIN コマンドを使用します。

準備済みステートメントの作成と使用の例と詳細については、「PREPARE」を参照してください。

以下の資料も参照してください。

DEALLOCATE, PREPARE

EXPLAIN

クエリを実行せずに、クエリステートメントの実行計画を表示します。クエリ分析ワークフローの詳細については、「クエリ分析ワークフロー」を参照してください。

構文

```
EXPLAIN [ VERBOSE ] query
```

パラメータ

VERBOSE

クエリプランの概要だけでなく、詳細情報を表示します。

query

説明を表示するクエリステートメント。SELECT、INSERT、CREATE TABLE AS、UPDATE、DELETE クエリステートメントを指定できます。

使用に関する注意事項

EXPLAIN のパフォーマンスは、一時テーブルの作成にかかる時間の影響を受けることがあります。例えば、共通のサブ式の最適化を使用するクエリでは、EXPLAIN の出力を返すために、一時テーブルを作成し、分析する必要があります。クエリプランは、一時テーブルのスキーマと統計情報に依存します。そのため、このようなクエリの EXPLAIN コマンドには、予測よりも長い実行時間がかかる可能性があります。

EXPLAIN は次のコマンドのみに使用できます。

- SELECT
- SELECT INTO
- CREATE TABLE AS
- INSERT
- UPDATE
- DELETE

データ定義言語 (DDL) やデータベース操作などのその他の SQL コマンドに対して使用した場合、EXPLAIN コマンドは失敗します。

Amazon Redshift は EXPLAIN 出力の相対単位コストを使用してクエリプランを選択します。Amazon Redshift は、さまざまなリソース見積もりのサイズを比較してプランを決定します。

クエリプランと実行ステップ

各 Amazon Redshift クエリステートメントの実行計画では、クエリの実行と計算を複数のステップとテーブル操作に分割し、クエリの最終的な結果セットを作成します。クエリの計画については、「クエリ処理」を参照してください。

次の表では、ユーザーが実行のために送信するクエリの実行計画を作成するときに、Amazon Redshift で使用できるステップの概要を説明します。

EXPLAIN の演算子	クエリ実行手順	説明
SCAN:		
Sequential Scan	scan	Amazon Redshift のリレーションスキャンまたはテーブルスキャンの演算子あるいはステッ

EXPLAIN の演算子	クエリ実行手順	説明
		プ。テーブル全体を最初から最後までスキャンします。また、WHERE 句で指定した場合は、各行についてのクエリの制約 (フィルタ) も評価します。また、INSERT、UPDATE、および DELETE ステートメントの実行にも使用されます。
JOINS: Amazon Redshift は、結合されるテーブルの物理的な設計、結合に必要なデータの場所、クエリ固有の属性に基づいて、異なる結合演算子を使用します。Subquery Scan -- Subquery scan と append は、UNION クエリの実行に使用されます。		
Nested Loop	nloop	最小限の最適な結合。主にクロス結合 (デカルト積、結合条件を使用しない) と一部の非等値結合に使用されます。
Hash Join	hjoin	内部結合と左右の外部結合にも使用され、通常、入れ子のループ結合よりも高速です。Hash Join では、外部テーブルを読み取り、結合する列をハッシュ処理し、内部ハッシュテーブルで一致を検索します。ディスクを使用するステップにすることもできます (hjoin の内部入力は、ディスクベースにすることができるハッシュステップです)。
Merge Join	mjoin	内部結合と外部結合にも使用されます (いずれの結合も、結合する列に基づいて分散とソートが行われます)。通常、他のコストを考慮しなければ、Amazon Redshift で最高速の結合アルゴリズムです。
AGGREGATION: 集計関数と GROUP BY 操作に関係するクエリに使用される演算子とステップ。		
Aggregate	aggr	スカラー集計関数の演算子とステップ。

EXPLAIN の演算子	クエリ実行手順	説明
HashAggregate	aggr	グループ化された集計関数の演算子とステップ。ハッシュテーブルの効力をディスクに拡張して、ディスクから操作できます。
GroupAggregate	aggr	force_hash_grouping 設定の Amazon Redshift の構成設定がオフの場合に、グループ化された集計クエリのために選択されることのある演算子。
SORT: クエリをソートする必要がある場合、または結果セットをマージする必要がある場合に使用される演算子とステップ。		
Sort	sort	Sort は、ORDER BY 句で指定されたソートと、UNION や結合などの操作を実行します。ディスクから操作できます。
Merge	merge	並行して実行された操作から派生した中間のソート結果に基づいて、クエリの最終的なソート結果を生成します。
EXCEPT、INTERSECT、UNION 操作:		
SetOp Except [Distinct]	hjoin	EXCEPT クエリに使用されます。入力ハッシュをディスクベースにすることができる機能に基づいて、ディスクから操作できます。
Hash Intersect [Distinct]	hjoin	INTERSECT クエリに使用されます。入力ハッシュをディスクベースにすることができる機能に基づいて、ディスクから操作できます。
Append [All |Distinct]	save	Append は、UNION および UNION ALL クエリを実装するために、Subquery Scan と共に使用されます。「save」の機能に基づいて、ディスクから操作できます。
その他:		

EXPLAIN の演算子	クエリ実行手順	説明
ハッシュ	hash	内部結合と左右の外部結合に使用されます (ハッシュ結合に入力を提供します)。Hash 演算子で、結合の内部テーブルのハッシュテーブルが作成されます (内部テーブルは、一致について確認されるテーブルであり、2 つのテーブルの結合で、通常は 2 つのうち小さい方です)。
制限	limit	LIMIT 句を評価します。
Materialize	save	入れ子のループ結合と一部のマージ結合への入力のために、行をマテリアライズします。ディスクから操作できます。
--	parse	ロード中にテキストの入力データを解析するために使用されます。
--	プロジェクト	列をソートし、式 (つまりプロジェクトデータ) を計算するために使用されます。
結果	--	テーブルへのアクセスを伴わないスカラー関数を実行します。
--	return	行をリーダーまたはクライアントに返します。
Subplan	--	特定のサブクエリに使用されます。
Unique	unique	SELECT DISTINCT および UNION クエリから重複が除外されます。
Window	window	集計およびランキングウィンドウ関数を計算します。ディスクから操作できます。
ネットワーク操作:		
Network (Broadcast)	bcast	Broadcast は、Join Explain 演算子とステップの属性でもあります。

EXPLAIN の演算子	クエリ実行手順	説明
Network (Distribute)	dist	データウェアハウスクラスターによる並行処理のために、行をコンピューティングノードに分散します。
Network (Send to Leader)	return	さらに詳細な処理のために、結果をリーダーに送り返します。
DML 操作 (データを変更する演算子):		
Insert (using Result)	insert	データを挿入します。
Delete (Scan + Filter)	delete	データを削除します。ディスクから操作できます。
Update (Scan + Filter)	delete、insert	delete と Insert として実装されます。

RLS に EXPLAIN を使う

クエリに行レベルセキュリティ (RLS) ポリシーの対象となるテーブルが含まれている場合、EXPLAIN は特別な RLS SecureScan ノードを表示します。Amazon Redshift は、同じノードタイプを STL_EXPLAIN システムテーブルにも記録します。EXPLAIN は、dim_tbl に適用される RLS 述語を明らかにしません。RLS SecureScan ノードタイプは、現在のユーザーには見えない追加の操作が実行プランに含まれていることを示す指標として機能します。

次の例は、RLS SecureScan ノードを示しています。

```
EXPLAIN
SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                               QUERY PLAN
------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   ->  *XN* *RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)*
         Filter: ((k_dim / 10) > 0)
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
```

```
                Filter: (("k" / 10) > 0)
```

RLS の対象となるクエリプランの完全な調査を可能にするため、Amazon Redshift は EXPLAIN RLS のシステム許可を提供します。この許可を付与されたユーザーは、RLS 述語も含む完全なクエリプランを検査できます。

次の例は、RLS SecureScan ノードの下に追加の Seq Scan にも RLS ポリシー述語 (k_dim > 1) が含まれることを示しています。

```
EXPLAIN SELECT D.cint
FROM fact_tbl F INNER JOIN dim_tbl D ON F.k_dim = D.k
WHERE F.k_dim / 10 > 0;
                                   QUERY PLAN
---------------------------------------------------------------------------------
 XN Hash Join DS_DIST_ALL_NONE  (cost=0.08..0.25 rows=1 width=4)
   Hash Cond: ("outer".k_dim = "inner"."k")
   *->  XN RLS SecureScan f  (cost=0.00..0.14 rows=2 width=4)
         Filter: ((k_dim / 10) > 0)*
         ->  *XN* *Seq Scan on fact_tbl rls_table  (cost=0.00..0.06 rows=5 width=8)
               Filter: (k_dim > 1)*
   ->  XN Hash  (cost=0.07..0.07 rows=2 width=8)
         ->  XN Seq Scan on dim_tbl d  (cost=0.00..0.07 rows=2 width=8)
               Filter: (("k" / 10) > 0)
```

EXPLAIN RLS 許可がユーザーに付与されている間、Amazon Redshift は RLS 述語を含む完全なクエリプランを STL_EXPLAIN システムテーブルに記録します。この許可が付与されていない間に実行されるクエリは、RLS 内部情報なしでログ記録されます。EXPLAIN RLS 許可を付与または削除しても、Amazon Redshift が以前のクエリで STL_EXPLAIN にログ記録した内容は変更されません。

AWS Lake Formation-RLS による Redshift の保護関係

次の例は、LF SecureScan ノードを示しています。このノードを使用して、Lake Formation と RLS の関係を表示できます。

```
EXPLAIN
SELECT *
FROM lf_db.public.t_share
WHERE a > 1;
QUERY PLAN
---------------------------------------------------------------
XN LF SecureScan t_share  (cost=0.00..0.02 rows=2 width=11)
```

```
(2 rows)
```

例

> **Note**
>
> これらの例では、出力例は Amazon Redshift の設定によって変わります。

次の例は、EVENT テーブルと VENUE テーブルから EVENTID、EVENTNAME、VENUEID、および VENUENAME を選択するクエリのクエリプランを返します。

```
explain
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                                QUERY PLAN
--------------------------------------------------------------------------
XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(5 rows)
```

次の例では、同じクエリで詳細な出力のクエリプランを返します。

```
explain verbose
select eventid, eventname, event.venueid, venuename
from event, venue
where event.venueid = venue.venueid;
```

```
                                QUERY PLAN
--------------------------------------------------------------------------
{HASHJOIN
:startup_cost 2.52
:total_cost 58653620.93
:plan_rows 8712
:plan_width 43
```

```
:best_pathkeys <>
:dist_info DS_DIST_OUTER
:dist_info.dist_keys (
TARGETENTRY
{
VAR
:varno 2
:varattno 1
...

XN Hash Join DS_DIST_OUTER  (cost=2.52..58653620.93 rows=8712 width=43)
Hash Cond: ("outer".venueid = "inner".venueid)
->  XN Seq Scan on event  (cost=0.00..87.98 rows=8798 width=23)
->  XN Hash  (cost=2.02..2.02 rows=202 width=22)
->  XN Seq Scan on venue  (cost=0.00..2.02 rows=202 width=22)
(519 rows)
```

次の例では、CREATE TABLE AS (CTAS) ステートメントのクエリプランを返します。

```
explain create table venue_nonulls as
select * from venue
where venueseats is not null;

QUERY PLAN
-----------------------------------------------------------
XN Seq Scan on venue  (cost=0.00..2.02 rows=187 width=45)
Filter: (venueseats IS NOT NULL)
(2 rows)
```

FETCH

カーソルを使用して行を取得します。カーソルの宣言について詳しくは、「DECLARE」を参照してください。

FETCH は、カーソル内の現在の位置に基づいて行を取得します。カーソルを作成すると、最初の行の前に位置が設定されます。FETCH 後は、最後に取得した行にカーソル位置が設定されます。使用できる最後の行まで FETCH を実行すると (FETCH ALL の後など)、カーソル位置は最後の行の後になります。

FORWARD 0 では、現在の行を取得し、カーソルを移動しません。つまり、最後に取得した行を取得します。カーソル位置が最初の行の前、または最後の行の後の場合、行は返されません。

カーソルの最初の行が取得されると、必要に応じて、結果セット全体がリーダーノード、メモリ内、またはディスク上にマテリアライズされます。大きな結果セットにカーソルを使用すると、パフォーマンスが低下する可能性があるため、可能な限り、別の方法を使用することをお勧めします。詳細については、「カーソルを使用するときのパフォーマンスに関する考慮事項」を参照してください。

詳細については、「DECLARE」、「CLOSE」を参照してください。

構文

```
FETCH [ NEXT | ALL | {FORWARD [ count | ALL ] } ] FROM cursor
```

パラメータ

NEXT

次の行を取得します。これがデフォルト値です。

ALL

残りのすべての行を取得します (FORWARD ALL と同じです)。単一ノードクラスターでは、ALL はサポートされません。

FORWARD [count | ALL]

次の count の行、または残りのすべての行を取得します。`FORWARD 0` は、現在の行を取得します。単一ノードクラスターでは、数値の最大値は `1000` です。単一ノードクラスターでは、FORWARD ALL はサポートされません。

cursor

新しいカーソルの名前。

FETCH の例

次の例では、LOLLAPALOOZA というカーソルを宣言し、Lollapalooza イベントの売り上げ情報を選択した後、カーソルを使用して結果セットから行を取得します。

```
-- Begin a transaction

begin;

-- Declare a cursor

```

```
declare lollapalooza cursor for
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Lollapalooza';

-- Fetch the first 5 rows in the cursor lollapalooza:

fetch forward 5 from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-05-01 19:00:00 |   92.00000000 |       3
 Lollapalooza | 2008-11-15 15:00:00 |  222.00000000 |       2
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       3
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       4
 Lollapalooza | 2008-04-17 15:00:00 |  239.00000000 |       1
(5 rows)

-- Fetch the next row:

fetch next from lollapalooza;

  eventname   |      starttime      | costperticket | qtysold
--------------+---------------------+---------------+---------
 Lollapalooza | 2008-10-06 14:00:00 |  114.00000000 |       2

-- Close the cursor and end the transaction:

close lollapalooza;
commit;
```

GRANT

ユーザーまたはユーザーグループのアクセス許可を定義します。

アクセス許可には、テーブルとビューのデータの読み取り、データの書き込み、テーブルの作成、テーブルの削除などのアクセスオプションが含まれます。このコマンドを使用して、テーブル、データベース、スキーマ、関数、プロシージャ、言語、または列に対する特定のアクセス許可を付与します。データベースオブジェクトからアクセス許可を削除するには、REVOKE コマンドを使用します。

アクセス許可には、以下のデータ共有プロデューサーのアクセスオプションも含まれます。

- コンシューマー名前空間とアカウントへのデータ共有のアクセス許可を付与する。
- データ共有にオブジェクトを追加したり、データ共有からオブジェクトを削除したりすることでデータ共有を変更するアクセス許可を付与する。
- コンシューマー名前空間をデータ共有に追加または削除することでデータ共有を共有するアクセス許可を付与する。

データ共有コンシューマーのアクセスオプションは次のとおりです。

- データ共有から作成されたデータベース、またはそのようなデータベースを指す外部スキーマへのフルアクセスをユーザーに付与する。
- ローカルデータベースオブジェクトと同様に、データ共有から作成されたデータベースに対するオブジェクトレベルのアクセス許可をユーザーに付与する。このレベルのアクセス許可を付与するには、データ共有からデータベースを作成するときに WITH PERMISSIONS 句を使用する必要があります。詳細については、「CREATE DATABASE」を参照してください。

データ共有のアクセス許可については、「クラスター内およびクラスター間のデータ共有」を参照してください。

また、ロールを付与してデータベースアクセス許可を管理したり、データに関連してユーザーが実行できる操作を制御したりすることもできます。ロールを定義し、ユーザーにロールを割り当てることで、ユーザーを CREATE TABLE コマンドと INSERT コマンドのみに制限するなど、ユーザーが実行できるアクションを制限できます。CREATE ROLE コマンドの詳細については、「the section called “CREATE ROLE”」を参照してください。Amazon Redshift にはシステム定義のロールがいくつかあり、これらを使用してユーザーに特定のアクセス許可を付与することもできます。詳細については、「the section called “Amazon Redshift でのシステム定義のロール”」を参照してください。

ON SCHEMA 構文を使用するデータベースユーザーおよびユーザーグループには、外部スキーマに対する USAGE 権限の GRANT (付与) または REVOKE (取り消し) のみを行うことができます。AWS Lake Formation で ON EXTERNAL SCHEMA を使用する場合は、AWS Identity and Access Management(IAM) ロールに対して、アクセス許可の GRANT (付与) および REVOKE (取り消し) のみを行うことができます。アクセス許可のリストについては、構文を参照してください。

ストアドプロシージャの場合、付与できるアクセス許可は EXCUTE のみです。

トランザクションブロック (BEGIN ... END) 内で GRANT を (外部リソースで) 実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

データベースに対してどのアクセス許可がユーザーに付与されているかを確認するには、HAS_DATABASE_PRIVILEGE を使用します。スキーマに対してどのアクセス許可がユーザーに付与されているかを確認するには、HAS_SCHEMA_PRIVILEGE を使用します。テーブルに対してどのアクセス許可がユーザーに付与されているかを確認するには、HAS_TABLE_PRIVILEGE を使用します。

構文

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE }
 [,...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE db_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...] | ALL
 PROCEDURES IN SCHEMA schema_name [, ...] }
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT USAGE
    ON LANGUAGE language_name [, ...]
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

テーブルの列レベルのアクセス許可の付与

Amazon Redshift テーブルとビューに対する列レベルのアクセス許可の構文を次に示します。

```
GRANT { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }

     TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

ASSUMEROLE アクセス許可の付与

指定されたロールを持つユーザーおよびグループに付与される ASSUMEROLE アクセス許可の構文を次に示します。ASSUMEROLE 権限の使用を開始する際は、「ASSUMEROLE アクセス許可を付与するための使用上の注意事項」を参照してください。

```
GRANT ASSUMEROLE
       ON { 'iam_role' [, ...] | default | ALL }
       TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
       FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]
```

Redshift Spectrum と Lake Formation の統合に関するアクセス許可の付与

以下に、Redshift Spectrum と Lake Formation の統合構文を示します。

```
GRANT { SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    TO { { IAM_ROLE iam_role } [, ...] | PUBLIC } [ WITH GRANT OPTION ]

GRANT { { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    TO { IAM_ROLE iam_role } [, ...] [ WITH GRANT OPTION ]
```

データ共有アクセス許可の付与

プロデューサー側のデータ共有のアクセス許可

以下は、GRANT を使用してユーザーまたはロールに ALTER または SHARE アクセス許可を付与するための構文です。ユーザーは ALTER アクセス許可でデータ共有を変更したり、SHARE アクセス許可でコンシューマーに使用を許可したりできます。ALTER と SHARE は、データ共有に対してユーザーおよびロールに付与できる唯一のアクセス許可です。

```
GRANT { ALTER | SHARE } ON DATASHARE datashare_name
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

以下は、Amazon Redshift のデータ共有アクセス許可について GRANT を使用するための構文です。USAGE アクセス許可を使用して、データ共有へのアクセスをコンシューマーに付与します。ユーザーまたはユーザーグループに、このアクセス許可を付与することはできません。このアクセス許可は、GRANT ステートメントの WITH GRANT OPTION もサポートしていません。このタイプの GRANT ステートメントを実行できるのは、以前に FOR データ共有に付与された SHARE アクセス許可を持つユーザーまたはユーザーグループのみです。

```
GRANT USAGE
    ON DATASHARE datashare_name
    TO NAMESPACE 'namespaceGUID' | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
```

以下は、Lake Formation アカウントにデータ共有の使用を許可する方法の例です。

```
GRANT USAGE ON DATASHARE salesshare TO ACCOUNT '123456789012' VIA DATA CATALOG;
```

コンシューマー側のデータ共有のアクセス許可

以下は、データ共有から作成された特定のデータベースまたはスキーマに対する GRANT データ共有の使用許可の構文です。

データ共有から作成されたデータベースにコンシューマーがアクセスするために必要なその他のアクセス許可は、データ共有からデータベースを作成するために使用された CREATE DATABASE コマンドが WITH PERMISSIONS 句を使用したかどうかによって異なります。CREATE DATABASE コマンドと WITH PERMISSIONS 句の詳細については、「CREATE DATABASE」を参照してください。

WITH PERMISSIONS 句を使用せずに作成されたデータベース

WITH PERMISSIONS 句を使用せずにデータ共有から作成されたデータベースに USAGE を付与する場合、共有データベースのオブジェクトに対して個別にアクセス許可を付与する必要はありませ

ん。WITH PERMISSIONS 句を使用せずにデータ共有から作成されたデータベースに対して使用を許可されたエンティティは、自動的にデータベース内のすべてのオブジェクトにアクセスできます。

WITH PERMISSIONS 句を使用して作成されたデータベース

WITH PERMISSIONS 句を使用してデータ共有から共有データベースを作成したデータベースに USAGE を付与する場合、コンシューマ側の ID には、共有データベース内のデータベースオブジェクトにアクセスするために、そのデータベースオブジェクトに関連するアクセス許可が付与されている必要があります。ローカルデータベースオブジェクトに対するアクセス許可を付与する場合と同様です。データ共有から作成されたデータベース内のオブジェクトにアクセス許可を付与するには、3 つの部分からなる構文 `database_name.schema_name.object_name` を使用します。共有データベース内の共有スキーマを指す外部スキーマのオブジェクトにアクセス許可を付与するには、2 つの部分からなる構文 `schema_name.object_name` を使用します。

```
GRANT USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
    TO { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

スコープ付きアクセス許可の付与

スコープ設定アクセス許可を使用すると、データベースまたはスキーマ内の特定タイプのすべてのオブジェクトに対するアクセス許可をユーザーまたはロールに付与できます。スコープ設定アクセス許可を持つユーザーやロールは、データベースまたはスキーマ内の現在および将来のすべてのオブジェクトに対して指定されたアクセス許可を持ちます。

データベースレベルのスコープ付きアクセス許可の範囲は、[SVV_DATABASE_PRIVILEGES](#) で確認できます。スキーマレベルのスコープ付きアクセス許可の範囲は、[SVV_SCHEMA_PRIVILEGES](#) で確認できます。

以下は、ユーザーとロールにスコープ付きアクセス許可を付与するための構文です。スコープ設定アクセス許可の詳細については、「[スコープ設定アクセス許可](#)」を参照してください。

```
GRANT { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]

GRANT
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
```

```
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name} [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
TO { username [ WITH GRANT OPTION ] | ROLE role_name | } [, ...]

GRANT USAGE
FOR LANGUAGES IN
{DATABASE db_name}
TO { username [ WITH GRANT OPTION ] | ROLE role_name } [, ...]
```

スコープ設定アクセス許可では、関数とプロシージャのアクセス許可が区別されないことに注意してください。例えば、次のステートメントは、スキーマ Sales_schema 内の関数とプロシージャの両方に対する EXECUTE アクセス許可を bob に付与します。

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

機械学習アクセス許可の付与

以下は、Amazon Redshift での機械学習モデルアクセス許可の構文です。

```
GRANT CREATE MODEL
    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    TO { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

ロールアクセス許可の付与

以下に、Amazon Redshift でロールのアクセス許可を付与する際の構文を示します。

```
GRANT { ROLE role_name } [, ...] TO { { user_name [ WITH ADMIN OPTION ] } |
 ROLE role_name }[, ...]
```

以下に、Amazon Redshift でシステムのアクセス許可を付与する際の構文を示します。ユーザーではなく、ロールにのみアクセス許可を付与できることに注意してください。

```
GRANT
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG | ACCESS SYSTEM TABLE
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE |
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL |
    IGNORE RLS | EXPLAIN RLS |
    EXPLAIN MASKING }[, ...]
  }
  | { ALL [ PRIVILEGES ] }
TO ROLE role_name [, ...]
```

行レベルのセキュリティポリシーフィルターの説明アクセス許可の付与

次の内容は、EXPLAIN プランのクエリにおける行レベルのセキュリティポリシーフィルタを説明する許可を付与する構文です。この権限は、REVOKE ステートメントを使用して取り消すことができます。

```
GRANT EXPLAIN RLS TO ROLE rolename
```

次の内容は、クエリの行レベルのセキュリティポリシーをバイパスする許可を付与する構文です。

```
GRANT IGNORE RLS TO ROLE rolename
```

ポリシーオブジェクトへの RLS ルックアップテーブルのアクセス許可の付与

次の内容は、特定の行レベルのセキュリティポリシーに許可を付与する構文です。

```
GRANT SELECT ON [ TABLE ] table_name [, ...]
TO RLS POLICY policy_name [, ...]
```

パラメータ

SELECT

SELECT ステートメントを使用して、テーブルまたはビューからデータを選択するアクセス許可を付与します。UPDATE 操作または DELETE 操作で既存の列値を参照するには、SELECT アクセス許可も必要です。

INSERT

INSERT ステートメントまたは COPY ステートメントを使用して、データをテーブルにロードするアクセス許可を付与します。

UPDATE

UPDATE ステートメントを使用して、テーブル列を更新するアクセス許可を付与します。UPDATE 操作には SELECT アクセス許可も必要です。これは、更新する行、または列の新しい値を計算する行を決定するには、テーブルの列を参照する必要があるためです。

DELETE

テーブルからデータ行を削除するアクセス許可を付与します。DELETE 操作には、SELECT 権限も必要です。これは、削除する行を決定するには、テーブルの列を参照する必要があるためです。

DROP

テーブルを削除するアクセス許可を付与します。このアクセス許可は、Amazon Redshift、および Lake Formation が有効になっている AWS Glue Data Catalog に対して適用されます。

REFERENCES

外部キー制約を作成するアクセス許可を付与します。参照されるテーブルと参照するテーブルの両方について、このアクセス許可を付与する必要があります。そうしないと、ユーザーは制約を作成できません。

ALTER

データベースオブジェクトに応じて、次のアクセス許可をユーザーまたはユーザーグループに付与します。

- テーブルの場合、ALTER はテーブルまたはビューを変更するアクセス許可を付与します。詳細については、「ALTER TABLE」を参照してください。
- データベースの場合、ALTER はデータベースを変更するアクセス許可を付与します。詳細については、「ALTER DATABASE」を参照してください。
- スキーマの場合、ALTER はスキーマを変更するアクセス許可を付与します。詳細については、「ALTER SCHEMA」を参照してください。
- 外部テーブルの場合、ALTER は Lake Formation で有効になっている AWS Glue Data Catalog 内のテーブルを変更するアクセス許可を付与します。この許可は、Lake Formation を使用する場合にのみ適用されます。

TRUNCATE

テーブルを切り捨てるアクセス許可を付与します。このアクセス許可がない場合、テーブルの所有者またはスーパーユーザーだけがテーブルを切り捨てることができます。TRUNCATE コマンドの詳細については、「the section called “TRUNCATE”」を参照してください。

ALL [PRIVILEGES]

指定したユーザーまたはロールに、使用可能なすべてのアクセス許可を一度に付与します。PRIVILEGES キーワードはオプションです。

GRANT ALL ON SCHEMA では、外部スキーマに対する CREATE アクセス許可は付与されません。

Lake Formation で有効になっている AWS Glue Data Catalog のテーブルに対して、ALL アクセス許可を付与することができます。この場合、個々のアクセス許可 (SELECT、ALTER など) は、データカタログに記録されます。

> **Note**
>
> Amazon Redshift は、ルールおよび TRIGGER アクセス許可をサポートしていません。詳細については、「サポートされていない PostgreSQL 機能」を参照してください。

ASSUMEROLE

指定したロールを持つユーザー、ロール、またはグループに対し、COPY、UNLOAD、EXTERNAL FUNCTION、および CREATE MODEL コマンドを実行するアクセス許可を付与します。指定したコマンドを実行すると、ユーザー、ロール、またはグループがこのロールを引き受けます。ASSUMEROLE アクセス許可の使用を開始する際は、「ASSUMEROLE アクセス許可を付与するための使用上の注意事項」を参照してください。

ON [TABLE] table_name

テーブルまたはビューに対する、指定されたアクセス許可を付与します。TABLE キーワードはオプションです。1 つのステートメントで、複数のテーブルとビューを列挙できます。

ON ALL TABLES IN SCHEMA schema_name

参照されるスキーマ内のすべてのテーブルおよびビューに対する、指定されたアクセス許可を付与します。

(column_name [,...]) ON TABLE table_name

Amazon Redshift テーブルまたはビューの指定された列に対する、指定されたアクセス許可をユーザー、グループ、または PUBLIC に付与します。

(column_list) ON EXTERNAL TABLE schema_name.table_name

参照されるスキーマの Lake Formation テーブルの指定された列について、IAM ロールに、指定されたアクセス許可を付与します。

ON EXTERNAL TABLE schema_name.table_name

参照されるスキーマの指定された Lake Formation テーブルについて、指定されたアクセス許可を IAM ロールに付与します。

ON EXTERNAL SCHEMA schema_name

参照されるスキーマについて、指定されたアクセス許可を IAM ロールに付与します。

ON iam_role

指定されたアクセス許可を IAM ロールに付与します。

TO username

アクセス許可を受け取るユーザーを示します。

TO IAM_ROLE iam_role

アクセス許可を受け取る IAM ロールを示します。

WITH GRANT OPTION

アクセス許可を受け取るユーザーが、他のユーザーにも同じアクセス許可を付与できることを示します。WITH GRANT OPTION をグループや PUBLIC に付与することはできません。

ROLE role_name

ロールにアクセス許可を付与します。

GROUP group_name

ユーザーグループにアクセス許可を付与します。カンマ区切りのリストを使用して、複数のユーザーグループを指定することができます。

PUBLIC

指定されたアクセス許可を、後で作成されるユーザーを含め、すべてのユーザーに付与します。PUBLIC は、常にすべてのユーザーを含むグループを表します。各ユーザーのアクセス許可は、PUBLIC に付与されたアクセス許可、ユーザーが属するグループに付与されたアクセス許可、およびユーザーに個別に付与されたアクセス許可のすべてで構成されます。

PUBLIC を Lake Formation EXTERNAL TABLE に付与すると、Lake Formation の everyone グループにアクセス許可が付与されます。

CREATE

データベースオブジェクトに応じて、次のアクセス許可をユーザーまたはユーザーグループに付与します。

- データベースの場合、CREATE はデータベース内にスキーマを作成することをユーザーに許可します。
- スキーマの場合、CREATE はスキーマ内にオブジェクトを作成することをユーザーに許可します。オブジェクトの名前を変更するには、CREATE アクセス許可を持ち、名前を変更するオブジェクトを所有している必要があります。
- CREATE ON SCHEMA は、Amazon Redshift Spectrum 用の外部スキーマではサポートされていません。外部スキーマの外部テーブルの使用を許可するには、アクセスする必要のあるユーザーに USAGE ON SCHEMA を付与します。外部スキーマの所有者またはスーパーユーザーのみが外部スキーマ内に外部テーブルを作成できます。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用して所有者を変更します。

TEMPORARY | TEMP

指定されたデータベースに一時テーブルを作成するアクセス許可を付与します。Amazon Redshift Spectrum クエリを実行するには、データベースユーザーがデータベースに一時テーブルを作成するアクセス権限を持っている必要があります。

> **Note**
>
> デフォルトでは、PUBLIC グループの自動メンバーシップにより、一時テーブルを作成する権限がユーザーに付与されます。一時テーブルを作成するアクセス許可をユーザーから削除するには、PUBLIC グループから TEMP アクセス許可を取り消します。次に、一時テーブルを作成する権限を、特定のユーザーまたはユーザーグループに明示的に付与します。

ON DATABASE db_name

データベースに対する、指定されたアクセス許可を付与します。

USAGE

特定のスキーマに対する USAGE アクセス許可を付与します。これにより、そのスキーマ内のオブジェクトにユーザーがアクセスできるようになります。ローカルの Amazon Redshift スキーマでは、これらのオブジェクトに対する特定のアクションを個別に許可する必要があります (例: テーブルに対する SELECT または UPDATE アクセス許可)。デフォルトでは、PUBLIC スキーマに対して、すべてのユーザーが CREATE および USAGE アクセス許可を持ちます。

ON SCHEMA 構文を使用して外部スキーマに USAGE を許可する場合は、外部スキーマのオブジェクトに対して個別にアクションを許可する必要はありません。対応するカタログアクセス許可は、外部スキーマオブジェクトに対する詳細なアクセス許可を制御します。

ON SCHEMA schema_name

スキーマに対する、指定されたアクセス許可を付与します。

GRANT CREATE ON SCHEMA および GRANT ALL ON SCHEMA の CREATE アクセス許可は、Amazon Redshift Spectrum 外部スキーマではサポートされていません。外部スキーマの外部テーブルの使用を許可するには、アクセスする必要のあるユーザーに USAGE ON SCHEMA を付与します。外部スキーマの所有者またはスーパーユーザーのみが外部スキーマ内に外部テーブルを作成できます。外部スキーマの所有者を移行するには、「ALTER SCHEMA」を使用して所有者を変更します。

EXECUTE ON ALL FUNCTIONS IN SCHEMA schema_name

参照されるスキーマ内のすべての関数に対する指定されたアクセス許可を付与します。

Amazon Redshift は、pg_catalog 名前空間で定義された pg_proc 組み込みエントリの GRANT または REVOKE ステートメントをサポートしていません。

EXECUTE ON PROCEDURE procedure_name

特定のストアドプロシージャに対する EXECUTE アクセス許可を付与します。ストアドプロシージャ名は重複する場合があるため、プロシージャの引数リストを含める必要があります。詳細については、「ストアドプロシージャの名前付け」を参照してください。

EXECUTE ON ALL PROCEDURES IN SCHEMA schema_name

参照されるスキーマ内のすべてのストアドプロシージャに対する指定されたアクセス許可を付与します。

USAGE ON LANGUAGE language_name

言語に対する USAGE アクセス許可を付与します。

USAGE ON LANGUAGE アクセス許可は、CREATE FUNCTION コマンドを実行してユーザー定義関数 (UDF) を作成するために必要です。詳細については、「UDF のセキュリティとアクセス許可」を参照してください。

USAGE ON LANGUAGE は、CREATE PROCEDURE コマンドを実行してストアドプロシージャを作成するために必要です。詳細については、「ストアドプロシージャのセキュリティおよび権限 」を参照してください。

Python UDF の場合、`plpythonu`を使用します。SQL UDF の場合、`sql`を使用します。ストアドプロシージャの場合、`plpgsql`を使用します。

FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...]

アクセス許可が付与される SQL コマンドを指定します。ALL を指定することで、COPY、UNLOAD、EXTERNAL FUNCTION、および CREATE MODEL ステートメントに対するアクセス許可を付与できます。この句は、ASSUMEROLE アクセス許可の付与にのみ適用されます。

ALTER

データ共有にオブジェクトを追加または削除したり、プロパティ PUBLICACCESSIBLE を設定したりするために、ALTER アクセス許可をユーザーに付与します。詳細については、「ALTER DATASHARE」を参照してください。

SHARE

データコンシューマーをデータ共有に追加するためのアクセス許可をユーザーとユーザーグループに付与します。このアクセス許可は、特定のコンシューマー (アカウントまたは名前空間) がクラスターからデータ共有にアクセスできるようにするために必要です。コンシューマーは、グローバル一意識別子 (GUID) で指定されたものと同じか異なるクラスター名前空間を持つ、同じあるいは異なる AWS アカウントに置くことができます。

ON DATASHARE datashare_name

参照されるデータ共有に対する指定されたアクセス許可を付与します。コンシューマーアクセスコントロールの粒度については、「Amazon Redshift の異なるレベルでのデータ共有」を参照してください。

USAGE

同じアカウント内のコンシューマーアカウントまたは名前空間に USAGE が付与されると、アカウント内の特定のコンシューマアカウントまたは名前空間は、読み込み専用でデータ共有およびデータ共有のオブジェクトにアクセスできます。

TO NAMESPACE 'clusternamespace GUID'

コンシューマーがデータ共有に対する指定されたアクセス許可を受け取ることができる同じアカウント内の名前空間を示します。名前空間には 128 ビットの英数字 GUID を使用します。

TO ACCOUNT 'accountnumber' [VIA DATA CATALOG]

コンシューマーがデータ共有に対する指定されたアクセス許可を受け取ることができる別のアカウントの数を示します。「VIA DATA CATALOG」を指定すると、データ共有の使用を Lake Formation のアカウントに許可することになります。このパラメータを省略すると、クラスターを所有するアカウントに使用を許可することになります。

ON DATABASE shared_database_name> [, ...]

指定されたデータ共有に作成される、指定されたデータベースに対して指定された使用アクセス許可を付与します。

ON SCHEMA shared_schema

指定されたデータ共有に作成される指定されたスキーマに対する指定されたアクセス許可を付与します。

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES } IN

アクセス許可を付与するデータベースオブジェクトを指定します。IN に続くパラメータは、付与されるアクセス許可の範囲を定義します。

CREATE MODEL

特定のユーザーまたはユーザーグループに CREATE MODEL アクセス許可を付与します。

ON MODEL model_name

特定のモデルに対する EXECUTE アクセス許可を付与します。

ACCESS CATALOG

ロールがアクセスできるオブジェクトの関連メタデータを表示するアクセス許可を付与します。

{ role } [, ...]

別のロール、ユーザー、または PUBLIC に付与されるロール。

PUBLIC は、常にすべてのユーザーを含むグループを表します。各ユーザーのアクセス許可は、PUBLIC に付与されたアクセス許可、ユーザーが属するグループに付与されたアクセス許可、およびユーザーに個別に付与されたアクセス許可のすべてで構成されます。

TO { { user_name [WITH ADMIN OPTION] } | role }[, ...]

指定されたロールを、(WITH ADMIN OPTION を使用するか、別のロールを持つ、あるいは PUBLIC である) 指定されたユーザーに付与します。

WITH ADMIN OPTION 句は、ロールが付与されるすべてのユーザーに対し、そのロールに付与されているすべての管理オプションを提供します。

EXPLAIN RLS TO ROLE rolename

EXPLAIN プランにおけるクエリの行レベルのセキュリティポリシーフィルターを説明するアクセス許可をロールに付与します。

IGNORE RLS TO ROLE rolename

クエリの行レベルのセキュリティポリシーをバイパスするアクセス許可をロールに付与します。

使用に関する注意事項

GRANT の使用上の注意事項の詳細については、「the section called “使用に関する注意事項”」を参照してください。

例

GRANT の使用方法の例については、「the section called “例”」を参照してください。

使用に関する注意事項

オブジェクトに対する権限を付与するには、次の条件のうち 1 つを満たす必要があります。

- オブジェクトの所有者であること。
- スーパーユーザーであること。
- そのオブジェクトと権限に関する付与権限があること。

例えば、次のコマンドは、employees テーブルに対する SELECT コマンドの実行と、他のユーザーに対する同じ権限の付与と取り消しの両方をユーザー HR に許可します。

```
grant select on table employees to HR with grant option;
```

HR は、SELECT 以外のオペレーションに関する権限や employees 以外のテーブルに関する権限を付与することはできません。

別の例として、次のコマンドは、employees テーブルに対する ALTER コマンドの実行と、他のユーザーに対する同じ権限の付与と取り消しの両方をユーザー HR に許可します。

```
grant ALTER on table employees to HR with grant option;
```

HR は、ALTER 以外のオペレーションに関する権限や employees 以外のテーブルに関する権限を付与することはできません。

ビューに対する権限が付与されていても、その基礎となるテーブルに対する権限を持っていることにはなりません。同様に、スキーマに対する権限が付与されていても、スキーマ内のテーブルに対する権限を持っていることにはなりません。または、基となるテーブルに対するアクセス権を明示的に付与します。

AWS Lake Formation テーブルに権限を付与するには、テーブルの外部スキーマに関連付けられた IAM ロールに、外部テーブルに権限を付与する権限が必要です。次の例では、IAM ロール `myGrantor` に関連付けられた外部スキーマを作成します。IAM ロール `myGrantor` には、他のユーザーにアクセス許可を付与するアクセス許可があります。GRANT コマンドは、外部スキーマに関連付けられている IAM ロール `myGrantor` のアクセス許可を使用して、IAM ロール `myGrantee` にアクセス許可を付与します。

```
create external schema mySchema
```

```
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
grant select
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

IAM ロールにすべての権限を付与すると、関連する Lake Formation が有効なデータカタログで個々の権限が付与されます。例えば、次の GRANT ALL を実行すると、付与された個々の権限 (SELECT、ALTER、DROP、DELETE、および INSERT) が Lake Formation コンソールに表示されます。

```
grant all
on external table mySchema.mytable
to iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

スーパーユーザーは、オブジェクトの権限を設定する GRANT コマンドと REVOKE コマンドに関係なく、すべてのオブジェクトにアクセスできます。

列レベルのアクセスコントロールの使用上の注意

次の使用上の注意は、Amazon Redshift テーブルとビューに対する列レベルの権限に適用されます。これらの注意事項はテーブルに関して説明しています。例外を明示的に記述しない限り、ビューにも同じ注意事項が適用されます。

- Amazon Redshift テーブルの場合、列レベルで SELECT 権限と UPDATE 権限のみを付与できます。Amazon Redshift ビューの場合、列レベルで SELECT 権限のみを付与できます。
- ALL キーワードは、テーブルの列レベルの GRANT のコンテキストで使用される場合に組み合わされる SELECT 権限および UPDATE 権限のシノニムです。
- テーブル内のすべての列に対する SELECT アクセス許可がない場合、SELECT * 操作を実行すると、アクセス権がある列のみが返されます。ビューを使用する場合、SELECT * オペレーションはビュー内のすべての列にアクセスしようとします。すべての列へのアクセス許可がない場合、これらのクエリはアクセス拒否エラーで失敗します。
- 以下の場合、SELECT * はアクセス可能な列のみに展開されません。
  - SELECT * を使用してアクセス可能な列のみを含む通常のビューを作成することはできません。

  - SELECT * を使用してアクセス可能な列のみを含むマテリアライズドビューを作成することはできません。
- テーブルまたはビューに対する SELECT 権限または UPDATE 権限がある状態で列を追加した場合、テーブルまたはビューに対するすべての列、したがってそのすべての列に対する権限がまだあります。
- テーブルの所有者またはスーパーユーザーのみが、列レベルの権限を付与できます。
- WITH GRANT OPTION 句は、列レベルの権限をサポートしていません。
- テーブルレベルと列レベルの両方で同じ権限を保持することはできません。たとえば、ユーザー `data_scientist` は、テーブル `employee` に対する SELECT 権限と、列 `employee.department` に対する SELECT 権限の両方を保持することはできません。テーブルとテーブル内の列に同じ権限を付与する場合は、次の結果を考慮してください。
  - ユーザーがテーブルでテーブルレベルの権限を持っている場合、列レベルで同じ権限を付与しても効果はありません。
  - ユーザーがテーブルでテーブルレベルの権限を持っている場合に、テーブルの 1 つ以上の列に対して同じ権限を取り消すと、エラーが返されます。代わりに、テーブルレベルで権限を取り消します。
  - ユーザーが列レベルの権限を持っている場合、テーブルレベルで同じ権限を付与するとエラーが返されます。
  - ユーザーが列レベルの権限を持っている場合に、テーブルレベルで同じ権限を取り消すと、テーブルのすべての列に対する列の権限とテーブルの権限の両方が取り消されます。
- 遅延バインディングビューで列レベルの権限を付与することはできません。
- マテリアライズドビューを作成するには、ベーステーブルに対するテーブルレベルの SELECT 権限が必要です。特定の列に対する列レベルの権限がある場合でも、それらの列にのみマテリアライズドビューを作成することはできません。ただし、通常のビューと同様に、マテリアライズドビューの列に SELECT 権限を付与できます。
- 列レベルの権限の付与を検索するには、PG_ATTRIBUTE_INFO ビューを使用します。

ASSUMEROLE アクセス許可を付与するための使用上の注意事項

以下の使用上の注意は、Amazon Redshift での ASSUMEROLE アクセス許可の付与に適用されます。

ASSUMEROLE アクセス許可を使用して、COPY、UNLOAD、EXTERNAL FUNCTION、CREATE MODEL などのコマンドに対する、データベースユーザー、ロール、またはグループの IAM ロー

ルのアクセス許可を制御します。IAM ロールについて、ユーザー、ロール、またはグループに ASSUMEROLE アクセス許可を付与すると、そのユーザー、ロール、またはグループは、そのロールを引き受けてコマンドを実行できます。ASSUMEROLE アクセス許可により、必要に応じて適切なコマンドへのアクセス権を付与することができます。

データベースのスーパーユーザーのみが、ユーザー、ロール、およびグループに対して ASSUMEROLE アクセス許可を付与したり取り消したりできます。スーパーユーザーは、常に ASSUMEROLE アクセス許可を保持します。

ユーザー、ロール、およびグループが ASSUMEROLE アクセス許可を使用できるようにするには、スーパーユーザーは次の 2 つのアクションを実行します。

- クラスターで次のステートメントを 1 回実行します。

```
revoke assumerole on all from public for all;
```

- 適切なコマンドについて、ユーザー、ロール、およびグループに ASSUMEROLE アクセス許可を付与します。

ASSUMEROLE アクセス許可を付与するときに、ON 句でロールチェーンを指定できます。コンマを使用して、ロールチェーン内のロールを区切ります (例: `Role1,Role2,Role3`)。ASSUMEROLE アクセス許可を付与するときにロールチェーンが指定された場合、ASSUMEROLE アクセス許可によって付与された操作を実行するときにロールチェーンを指定する必要があります。ASSUMEROLE アクセス許可によって付与された操作を実行するときに、ロールチェーン内の個々のロールを指定することはできません。例えば、ユーザー、ロール、またはグループにロールチェーン `Role1,Role2,Role3` が付与されている場合、`Role1` のみを指定してオペレーションを実行することはできません。

ユーザーが COPY、UNLOAD、EXTERNAL FUNCTION、または CREATE MODEL 操作の実行を試みた場合、ASSUMEROLE アクセス許可が付与されていないと、次のようなメッセージが表示されます。

```
ERROR:  User awsuser does not have ASSUMEROLE permission on IAM role
 "arn:aws:iam::123456789012:role/RoleA" for COPY
```

ASSUMEROLE アクセス許可を通じて IAM ロールおよびコマンドへのアクセスが付与されたユーザーを一覧表示するには、「HAS_ASSUMEROLE_PRIVILEGE」を参照してください。指定したユーザーに付与された IAM ロールとコマンドのアクセス許可を一覧表示するには、「PG_GET_IAM_ROLE_BY_USER」を参照してください。指定した IAM ロー

ルへのアクセスが許可されたユーザー、ロール、およびグループを一覧表示するには、「PG_GET_GRANTEE_BY_IAM_ROLE」を参照してください。

機械学習アクセス許可の付与に関する使用上の注意事項

ML 関数に関連するアクセス許可を直接付与または取り消すことはできません。ML 関数は ML モデルに属し、アクセス許可はモデルを通じて制御されます。代わりに、ML モデルに関連するアクセス許可を付与することができます。次の例は、モデル customer_churn に関連付けられた ML 関数を実行するアクセス許可をすべてのユーザーに付与する方法を示しています。

```
GRANT EXECUTE ON MODEL customer_churn TO PUBLIC;
```

ML モデル customer_churn に対するすべてのアクセス許可をユーザーに付与することもできます。

```
GRANT ALL on MODEL customer_churn TO ml_user;
```

スキーマに ML 関数がある場合、その ML 関数が既に GRANT EXECUTE ON MODEL を通じて EXECUTE アクセス許可を持っている場合でも、ML 関数に関連する EXECUTE アクセス許可の付与は失敗します。CREATE MODEL コマンドを使用して ML 関数を個別のスキーマに保持するときには、個別のスキーマを使用することをお勧めします。次の例は、その方法を示しています。

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'amzn-s3-demo-bucket'
);
```

例

次の例では、SALES テーブルに対する SELECT 権限をユーザーに付与します。fred

```
grant select on table sales to fred;
```

次の例では、QA_TICKIT スキーマのすべてのテーブルに対する SELECT 権限をユーザー fred に付与します。

```
grant select on all tables in schema qa_tickit to fred;
```

次の例では、スキーマ QA_TICKIT に対するすべてのスキーマ権限をユーザーグループ QA_USERS に付与します。スキーマに対する権限は CREATE と USAGE です。USAGE は、スキーマ内のオブジェクトに対するアクセス権限をユーザーに付与しますが、それらのオブジェクトに対する INSERT や SELECT などの権限は付与しません。各オブジェクトに対する権限を個別に付与します。

```
create group qa_users;
grant all on schema qa_tickit to group qa_users;
```

次の例では、グループ QA_USERS のすべてのユーザーに対して、QA_TICKIT スキーマの SALES テーブルに対するすべての権限を付与します。

```
grant all on table qa_tickit.sales to group qa_users;
```

次の例では、グループ QA_USERS と RO_USERS のすべてのユーザーに対して、QA_TICKIT スキーマ内の SALES テーブルに対するすべての権限を付与します。

```
grant all on table qa_tickit.sales to group qa_users, group ro_users;
```

次の例では、グループ QA_USERS のすべてのユーザーに対して、QA_TICKIT スキーマの SALES テーブルに対する DROP 権限を付与します。

```
grant drop on table qa_tickit.sales to group qa_users;>
```

次の一連のコマンドは、スキーマに対するアクセス権限があっても、スキーマ内のテーブルに対する権限は付与されていないことを示しています。

```
create user schema_user in group qa_users password 'Abcd1234';
create schema qa_tickit;
create table qa_tickit.test (col1 int);
grant all on schema qa_tickit to schema_user;

set session authorization schema_user;
select current_user;


current_user
--------------
```

```
schema_user
(1 row)


select count(*) from qa_tickit.test;


ERROR: permission denied for relation test [SQL State=42501]


set session authorization dw_user;
grant select on table qa_tickit.test to schema_user;
set session authorization schema_user;
select count(*) from qa_tickit.test;


count
-------
0
(1 row)
```

次の一連のコマンドは、ビューに対するアクセス権限があっても、基礎となるテーブルに対するアクセス権限は付与されていないことを示しています。VIEW_USER というユーザーには VIEW_DATE に関するすべての権限が付与されていますが、DATE テーブルから選択することはできません。

```
create user view_user password 'Abcd1234';
create view view_date as select * from date;
grant all on view_date to view_user;
set session authorization view_user;
select current_user;


current_user
--------------
view_user
(1 row)


select count(*) from view_date;


count
-------
```

```
365
(1 row)


select count(*) from date;


ERROR:  permission denied for relation date
```

次の例では、cust_profileテーブルの cust_name 列と cust_phone 列に対する SELECT 権限をユーザー user1 に付与します。

```
grant select(cust_name, cust_phone) on cust_profile to user1;
```

次の例では、cust_name列と cust_phone 列に対する SELECT 権限、そして cust_profile テーブルの cust_contact_preference 列に対する UPDATE 権限を sales_group グループに付与します。

```
grant select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile to
 group sales_group;
```

次の例では、ALL キーワードを使用して、テーブル cust_profile に対する SELECT および UPDATE 権限の両方を sales_admin グループに付与します。

```
grant ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile to group
 sales_admin;
```

次の例では、cust_profile_vwビューの cust_name 列に対する SELECT 権限を user2 ユーザーに付与します。

```
grant select(cust_name) on cust_profile_vw to user2;
```

データ共有へのアクセス許可を付与する例

次の例は、データ共有から作成された特定のデータベースまたはスキーマに対する GRANT データ共有の使用許可を示しています。

次の例では、プロデューサー側の管理者が salesshare データ共有に対する USAGE アクセス許可を指定された名前空間に付与しています。

```
GRANT USAGE ON DATASHARE salesshare TO NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

次の例では、コンシューマー側の管理者がsales_db に対する USAGE アクセス許可を Bob に付与しています。

```
GRANT USAGE ON DATABASE sales_db TO Bob;
```

次の例では、コンシューマー側の管理者が sales_schema スキーマに対する GRANT USAGE アクセス許可を Analyst_role ロールに付与しています。sales_schemaは sales_db を指す外部スキーマです。

```
GRANT USAGE ON SCHEMA sales_schema TO ROLE Analyst_role;
```

この時点で、Bob と Analyst_role は、sales_schema と sales_db のすべてデータベースオブジェクトにアクセスできます。

次の例は、共有データベース内のオブジェクトに追加のオブジェクトレベルのアクセス許可を付与する方法を示しています。これらの追加のアクセス許可は、共有データベースの作成に使用された CREATE DATABASE コマンドで WITH PERMISSIONS 句が使用された場合にのみ必要です。CREATE DATABASE コマンドで WITH PERMISSIONS を使用しなかった場合、共有データベースに USAGE を付与すると、そのデータベース内のすべてのオブジェクトへのフルアクセスが付与されます。

```
GRANT SELECT ON sales_db.sales_schema.tickit_sales_redshift to Bob;
```

スコープ付きアクセス許可を付与する例

次の例では、Sales_db データベース内の現在および将来のすべてのスキーマの使用を Sales ロールに付与します。

```
GRANT USAGE FOR SCHEMAS IN DATABASE Sales_db TO ROLE Sales;
```

次の例では、Sales_db データベース内の現在および将来のすべてのテーブルに対する SELECT アクセス許可をユーザー alice に付与し、また Sales_db のテーブルに対するスコープ付きアクセス許可を他のユーザーに付与する許可を alice に付与します。

```
GRANT SELECT FOR TABLES IN DATABASE Sales_db TO alice WITH GRANT OPTION;
```

次の例では、Sales_schema スキーマ内の関数に対する EXECUTE アクセス許可をユーザー bob に付与します。

```
GRANT EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema TO bob;
```

次の例では、ShareDb データベースの ShareSchema スキーマ内のすべてのテーブルに対するすべてのアクセス許可を Sales ロールに付与します。スキーマを指定するとき、2 つの部分からなる形式 database.schema を使用してスキーマのデータベースを指定できます。

```
GRANT ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema TO ROLE Sales;
```

次の例では、前の例と同じです。2 つの部分からなる形式を使用する代わりに、DATABASE キーワードを使用してデータベースを指定できます。

```
GRANT ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb TO ROLE Sales;
```

ASSUMEROLE 権限を付与する例

次に、ASSUMEROLE 権限を付与する例を示します。

以下に、ユーザーおよびグループに対して ASSUMEROLE 権限の使用を有効にするために、スーパーユーザーがクラスター上で 1 回実行する必要のある、REVOKE ステートメントの例を示します。次に、スーパーユーザーはユーザーおよびグループに対し、適切なコマンドを使用するための ASSUMEROLE 権限を付与します。ユーザーおよびグループに対する ASSUMEROLE 権限の使用を有効にする方法については、「ASSUMEROLE アクセス許可を付与するための使用上の注意事項」を参照してください。

```
revoke assumerole on all from public for all;
```

次の例では、IAM ロール Redshift-S3-Read が COPY オペレーションを実行するための ASSUMEROLE 権限をユーザー reg_user1 に付与します。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-S3-Read'
to reg_user1 for copy;
```

次の例では、IAM ロールチェーン RoleA、RoleB が UNLOAD オペレーションを実行するための ASSUMEROLE 権限をユーザー reg_user1 に付与します。

```
grant assumerole
```

```
on 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB'
to reg_user1
for unload;
```

以下は、IAM ロールチェーン RoleA、RoleBを使用した UNLOAD コマンドの例です。

```
unload ('select * from venue limit 10')
to 's3://companyb/redshift/venue_pipe_'
iam_role 'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:role/RoleB';
```

次の例では、IAM ロール Redshift-Exfunc が外部関数を作成するための ASSUMEROLE 権限を、ユーザー reg_user1 に対し付与しています。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-Exfunc'
to reg_user1 for external function;
```

次の例では、IAM ロール Redshift-model が機械学習モデルを作成するための ASSUMEROLE 権限を、ユーザー reg_user1 に対し付与しています。

```
grant assumerole on 'arn:aws:iam::123456789012:role/Redshift-ML'
to reg_user1 for create model;
```

ROLE 権限を付与する例

次の例では、user1 にロール sample_role1 を付与しています。

```
CREATE ROLE sample_role1;
GRANT ROLE sample_role1 TO user1;
```

次の例では、WITH ADMIN OPTION を使用して sample_role1 を user1 に付与します。さらに user1 の現在のセッションを設定した上で、user1 から user2 に sample_role1 を付与しています。

```
GRANT ROLE sample_role1 TO user1 WITH ADMIN OPTION;
SET SESSION AUTHORIZATION user1;
GRANT ROLE sample_role1 TO user2;
```

次の例では、sample_role1 を sample_role2 に付与します。

```
GRANT ROLE sample_role1 TO ROLE sample_role2;
```

次の例では、sample_role2 を sample_role3 および sample_role4 に付与します。次に、sample_role3 を sample_role1 に付与しようと試みています。

```
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2;
ERROR: cannot grant this role, a circular dependency was detected between these roles
```

次の例では、CREATE USER のシステム権限を sample_role1 に付与しています。

```
GRANT CREATE USER TO ROLE sample_role1;
```

次の例では、システム定義のロール `sys:dba` を user1 に付与します。

```
GRANT ROLE sys:dba TO user1;
```

次の例では、循環依存関係を利用して、sample_role3 を sample_role2 に付与することを試みます。

```
CREATE ROLE sample_role3;
GRANT ROLE sample_role2 TO ROLE sample_role3;
GRANT ROLE sample_role3 TO ROLE sample_role2; -- fail
ERROR:  cannot grant this role, a circular dependency was detected between these roles
```

INSERT

トピック

- 構文
- パラメータ
- 使用に関する注意事項
- INSERT の例

新しい行をテーブルに挿入します。VALUES 構文を使用して 1 行、VALUES 構文を使用して複数行、クエリの結果で定義された 1 つまたは複数の行を挿入できます (INSERT INTO ... SELECT)。

> **Note**
>
> 大量のデータをロードする場合は COPY コマンドを使用することを強くお勧めします。個々に INSERT ステートメントを使ってテーブルにデータを入力すると著しく時間がかかる場合

があります。または、他の Amazon Redshift データベーステーブルにデータが既に存在する場合、パフォーマンスを向上させるには、INSERT INTO SELECT または CREATE TABLE AS を使用します。COPY コマンドを使用してテーブルをロードする方法の詳細については、「Amazon Redshift でのデータのロード」を参照してください。

> **Note**
>
> 単一 SQL ステートメントの最大サイズは 16 MB です。

構文

```
INSERT INTO table_name [ ( column [, ...] ) ]
{DEFAULT VALUES |
VALUES ( { expression | DEFAULT } [, ...] )
[, ( { expression | DEFAULT } [, ...] )
[, ...] ] |
query }
```

パラメータ

table_name

一時テーブルまたは永続的テーブル テーブルの所有者またはテーブルに対して INSERT 権限を持つユーザーのみが行を挿入できます。query 句を使用して行を挿入する場合、クエリで指定したテーブルに対して SELECT 権限を持っている必要があります。

> **Note**
>
> INSERT (外部テーブル) を使用して、SELECT クエリの結果を外部カタログの既存のテーブルに挿入します。詳細については、「INSERT (外部テーブル)」を参照してください。

column

テーブルの 1 つまたは複数の列に値を挿入できます。ターゲットの列名は、任意の順序で列挙できます。列リストを指定しない場合、挿入する値は、CREATE TABLE ステートメントで宣言し

た順序で、テーブルの列に対応している必要があります。挿入する値の数が、テーブルの列数よりも少ない場合、最初の n 列がロードされます。

宣言されたデフォルト値または NULL 値が、INSERT ステートメントに含まれない列に (暗黙的または明示的に) ロードされます。

DEFAULT VALUES

テーブルの作成時に、テーブルの列にデフォルト値が割り当てられた場合、これらのキーワードを使用して、デフォルト値の全体を構成する行を挿入します。いずれかの列がデフォルト値ではない場合、それらの列には NULL が挿入されます。いずれかの列が NOT NULL と宣言されている場合、INSERT ステートメントはエラーを返します。

VALUES

このキーワードを使用して、1 つまたは複数の行を挿入します。各行は 1 つまたは複数の値で構成されます。各行の VALUES リストは、列リストに対応する必要があります。複数の行を挿入する場合、式リストの間はカンマで区切ります。VALUES キーワードは繰り返さないでください。複数行の INSERT ステートメントのすべての VALUES には、同じ数の値が含まれている必要があります。

expression

1 つの値、または 1 つの値に評価される式。各値は、挿入される列のデータ型と互換性がある必要があります。可能な場合、データ型が列に宣言されているデータ型と一致しない値は、互換性のあるデータ型に自動的に変換されます。次に例を示します。

- 10 進値の `1.1` は `1` として INT に挿入されます。
- 10 進値の `100.8976` は `100.90` として DEC (5,2) 列に挿入されます。

値を互換性のあるデータ型に明示的に変換するには、式に型変換構文を含めます。例えば、テーブル T1 の列 COL1 が CHAR(3) 列の場合。

```
insert into t1(col1) values('Incomplete'::char(3));
```

このステートメントは、列に値 `Inc` を挿入します。

1 行の INSERT VALUES ステートメントの場合、式としてスカラーサブクエリを使用できます。サブクエリの結果は適切な列に挿入されます。

> **Note**
>
> 複数行の INSERT VALUES ステートメントでは、式にサブクエリを使用できません。

DEFAULT

このキーワードを使用して、テーブル作成時の定義に従って列のデフォルト値を挿入します。列のデフォルト値が存在しない場合、NULL が挿入されます。NOT NULL の制約がある列に CREATE TABLE ステートメントで割り当てられた明示的なデフォルト値がない場合、それらの列にデフォルト値を挿入することはできません。

query

クエリを定義して、1 つまたは複数の行をテーブルに挿入します。クエリで生成されたすべての行がテーブルに挿入されます。クエリは、テーブルの列と互換性がある列リストを返す必要がありますが、列名が一致する必要はありません。

使用に関する注意事項

> **Note**
>
> 大量のデータをロードする場合は COPY コマンドを使用することを強くお勧めします。個々に INSERT ステートメントを使ってテーブルにデータを入力すると著しく時間がかかる場合があります。または、他の Amazon Redshift データベーステーブルにデータが既に存在する場合、パフォーマンスを向上させるには、INSERT INTO SELECT または CREATE TABLE AS を使用します。COPY コマンドを使用してテーブルをロードする方法の詳細については、「Amazon Redshift でのデータのロード」を参照してください。

挿入される値のデータ形式は、CREATE TABLE 定義で指定されたデータ形式と一致する必要があります。

テーブルに新しく多数の行を挿入した後:

- ストレージ容量を再利用し、行を再ソートするため、テーブルにバキューム処理を実行します。
- テーブルを分析して、クエリプランナーの統計情報を更新します。

値が DECIMAL 列に挿入され、指定されたスケールを超えると、必要に応じて、ロードされる値は切り上げられます。例えば、`20.259`という値を DECIMAL(8,2) に挿入すると、ソートされた値は `20.26` になります。

GENERATED BY DEFAULT AS IDENTITY 列を追加できます。GENERATED BY DEFAULT AS IDENTITY として定義された列を、指定した値で更新できます。詳細については、「GENERATED BY DEFAULT AS IDENTITY」を参照してください。

INSERT の例

TICKIT データベースの CATEGORY テーブルには、次の行が含まれています。

```
 catid | catgroup |  catname  |                  catdesc
-------+----------+-----------+--------------------------------------------
     1 | Sports   | MLB       | Major League Baseball
     2 | Sports   | NHL       | National Hockey League
     3 | Sports   | NFL       | National Football League
     4 | Sports   | NBA       | National Basketball Association
     5 | Sports   | MLS       | Major League Soccer
     6 | Shows    | Musicals  | Musical theatre
     7 | Shows    | Plays     | All non-musical theatre
     8 | Shows    | Opera     | All opera and light opera
     9 | Concerts | Pop       | All rock and pop music concerts
    10 | Concerts | Jazz      | All jazz singers and bands
    11 | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

CATEGORY テーブルと同様のスキーマを持つ CATEGORY_STAGE テーブルを作成します。ただし、列のデフォルト値を定義します。

```
create table category_stage
(catid smallint default 0,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');
```

次の INSERT ステートメントでは、CATEGORY テーブルのすべての行を選択し、CATEGORY_STAGE テーブルに挿入します。

```
insert into category_stage
```

```
(select * from category);
```

クエリを囲むかっこはオプションです。

このコマンドは、CATEGORY_STAGE テーブルに新しい行を挿入します。各列には順番に値が指定されます。

```
insert into category_stage values
(12, 'Concerts', 'Comedy', 'All stand-up comedy performances');
```

特定の値とデフォルトの値を組み合わせた新しい行を挿入できます。

```
insert into category_stage values
(13, 'Concerts', 'Other', default);
```

次のクエリを実行して、挿入される行を返します。

```
select * from category_stage
where catid in(12,13) order by 1;

 catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
    12 | Concerts | Comedy  | All stand-up comedy performances
    13 | Concerts | Other   | General
(2 rows)
```

次の例は、複数行の INSERT VALUES ステートメントを示しています。最初の例では、2 行について特定の CATID 値を挿入し、両方の行の他の列にはデフォルト値を挿入します。

```
insert into category_stage values
(14, default, default, default),
(15, default, default, default);

select * from category_stage where catid in(14,15) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
    14 | General  | General | General
    15 | General  | General | General
(2 rows)
```

次の例では、特定の値とデフォルトの値の多様な組み合わせを使用して 3 つの行を挿入します。

```
insert into category_stage values
(default, default, default, default),
(20, default, 'Country', default),
(21, 'Concerts', 'Rock', default);

select * from category_stage where catid in(0,20,21) order by 1;
 catid | catgroup | catname | catdesc
-------+----------+---------+---------
     0 | General  | General | General
    20 | General  | Country | General
    21 | Concerts | Rock    | General
(3 rows)
```

この例の最初にある一連の VALUES は、単一列の INSERT ステートメントに DEFAULT VALUES を指定した場合と同じ結果を提供します。

次の例では、テーブルに IDENTITY 列がある場合の INSERT の動作を示します。まず、CATEGORY テーブルの新しいバージョンを作成してから、CATEGORY よりこのテーブルに行を挿入します。

```
create table category_ident
(catid int identity not null,
catgroup varchar(10) default 'General',
catname varchar(10) default 'General',
catdesc varchar(50) default 'General');



insert into category_ident(catgroup,catname,catdesc)
select catgroup,catname,catdesc from category;
```

CATID IDENTITY 列には、整数値を指定して挿入できないことに注意してください。IDENTITY 列の値は、自動的に生成されます。

次の例は、複数行の INSERT VALUES ステートメントでは、式としてサブクエリを使用できないことを示しています。

```
insert into category(catid) values
((select max(catid)+1 from category)),
((select max(catid)+2 from category));

ERROR: can't use subqueries in multi-row VALUES
```

次の例は、WITH SELECT 句を使用して venue テーブルからのデータが入力されたテンポラリテーブルへの挿入を示しています。venue テーブルの詳細については、「サンプルデータベース」を参照してください。

まず、テンポラリテーブル #venuetemp を作成します。

```
CREATE TABLE #venuetemp AS SELECT * FROM venue;
```

#venuetemp テーブル内の行を一覧表示します。

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity   | venuestate| venueseats
--------+--------------------------+------------+-----------+------------
1        Toyota Park                Bridgeview   IL          0
2        Columbus Crew Stadium      Columbus     OH          0
3        RFK Stadium                Washington   DC          0
4        CommunityAmerica Ballpark  Kansas City  KS          0
5        Gillette Stadium           Foxborough   MA          68756
...
```

WITH SELECT 句を使用して #venuetemp テーブルに重複する行を 10 行挿入します。

```
INSERT INTO #venuetemp (WITH venuecopy AS (SELECT * FROM venue) SELECT * FROM venuecopy
 ORDER BY 1 LIMIT 10);
```

#venuetemp テーブル内の行を一覧表示します。

```
SELECT * FROM #venuetemp ORDER BY venueid;

venueid | venuename                | venuecity   | venuestate| venueseats
--------+--------------------------+------------+-----------+------------
1        Toyota Park                Bridgeview   IL          0
1        Toyota Park                Bridgeview   IL          0
2        Columbus Crew Stadium      Columbus     OH          0
2        Columbus Crew Stadium      Columbus     OH          0
3        RFK Stadium                Washington   DC          0
3        RFK Stadium                Washington   DC          0
4        CommunityAmerica Ballpark  Kansas City  KS          0
4        CommunityAmerica Ballpark  Kansas City  KS          0
5        Gillette Stadium           Foxborough   MA          68756
```

```
5        Gillette Stadium           Foxborough   MA          68756
...
```

INSERT (外部テーブル)

SELECT クエリの結果を、AWS Glue、AWS Lake Formation、Apache Hive メタストアなどの外部カタログの既存の外部テーブルに挿入します。CREATE EXTERNAL SCHEMA コマンドに使用されるのと同じ AWS Identity and Access Management (IAM) ロールを使用して、外部カタログおよび Amazon S3 とのやり取りを行います。

パーティション分割されていないテーブルの場合、INSERT (外部テーブル) コマンドは、指定されたテーブルプロパティとファイル形式に基づいて、テーブルで定義された Amazon S3 の場所にデータを書き込みます。

パーティション分割されたテーブルの場合、INSERT (外部テーブル) は、テーブルで指定されたパーティションキーに従って、Amazon S3 の場所にデータを書き込みます。また、INSERT オペレーションの完了後、新しいパーティションを外部カタログに自動的に登録します。

トランザクションブロック (BEGIN ... END) 内で INSERT を (外部テーブル) 実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。

構文

```
INSERT INTO external_schema.table_name
{ select_statement }
```

パラメータ

external_schema.table_name

既存の外部スキーマと挿入するターゲット外部テーブルの名前。

select_statement

クエリを定義して 1 つ以上の行を外部テーブルに挿入するステートメント。クエリによって生成されるすべての行は、テーブル定義に基づいて、テキストまたは Parquet 形式で Amazon S3 に書き込まれます。クエリは、外部テーブルの列のデータ型と互換性のある列リストを返す必要があります。ただし、列名は一致する必要はありません。

使用に関する注意事項

SELECT クエリの列数は、データ列とパーティション列の合計と同じであることが必要です。各データ列の場所とデータ型は、外部テーブルのものと一致する必要があります。パーティション列の場所は、CREATE EXTERNAL TABLE コマンドで定義されたのと同じ順序で、SELECT クエリの最後にあることが必要です。列名は一致する必要はありません。

場合によっては、AWS Glueデータカタログまたは Hive メタストアに対して INSERT (外部テーブル) コマンドを実行する必要があります。AWS Glue の場合、外部スキーマの作成に使用される IAM ロールには、Amazon S3 と AWS Glue に対する読み取りと書き込みの両方のアクセス許可が必要です。AWS Lake Formation カタログを使用する場合、この IAM ロールが新しい Lake Formation テーブルの所有者になります。この IAM ロールには、少なくとも以下のアクセス許可が必要です。

- 外部テーブルに対する SELECT、INSERT、UPDATE アクセス許可
- 外部テーブルの Amazon S3 パスに対するデータの場所のアクセス許可

ファイル名が必ず一意になるように、Amazon Redshift ではデフォルトで、Amazon S3 にアップロードされる各ファイルの名前に以下の形式が使用されます。

`<date>_<time>_<microseconds>_<query_id>_<slice-number>_part_<part-number>.<format>`.

例は `20200303_004509_810669_1007_0001_part_00.parquet` です。

INSERT (外部テーブル) コマンドを実行するときは、以下の点を考慮してください。

- PARQUET または TEXTFILE 以外の形式の外部テーブルはサポートされていません。
- このコマンドは、'write.parallel'、'write.maxfilesize.mb'、'compression_type'、'serialization.null.format' などの既存のテーブルプロパティをサポートしています。これらの値を更新するには、ALTER TABLE SET TABLE PROPERTIES コマンドを実行します。
- 'numRows' テーブルプロパティは、INSERT オペレーションの最後に向かって自動的に更新されます。テーブルプロパティは、CREATE EXTERNAL TABLE AS オペレーションによって作成されていない場合、テーブルに定義または追加する必要があります。
- LIMIT 句は、外側の SELECT クエリではサポートされていません。代わりに、ネストされた LIMIT 句を使用してください。
- STL_UNLOAD_LOG テーブルを使用して、各 INSERT (外部テーブル) オペレーションによって Amazon S3 に書き込まれたファイルを追跡できます。

- Amazon Redshift は、INSERT (外部テーブル) に対する Amazon S3 の標準暗号化のみをサポートします。

INSERT (外部テーブル) の例

以下の例では、SELECT ステートメントの結果を外部テーブルに挿入します。

```
INSERT INTO spectrum.lineitem
SELECT * FROM local_lineitem;
```

以下の例では、静的パーティション分割を使用して、SELECT ステートメントの結果をパーティション分割された外部テーブルに挿入します。パーティション列は SELECT ステートメントでハードコーディングされています。パーティション列はクエリの最後にあることが必要です。

```
INSERT INTO spectrum.customer
SELECT name, age, gender, 'May', 28 FROM local_customer;
```

以下の例では、動的パーティション分割を使用して、SELECT ステートメントの結果をパーティション分割された外部テーブルに挿入します。パーティション列はハードコーディングされていません。データは自動的に、既存のパーティションフォルダに追加されるか、または新しいパーティションが追加された場合は新しいフォルダに追加されます。

```
INSERT INTO spectrum.customer
SELECT name, age, gender, month, day FROM local_customer;
```

LOCK

データベーステーブルへのアクセスを制限します。このコマンドは、トランザクションブロック内で実行されている場合のみ意味を持ちます。

LOCK コマンドは、「ACCESS EXCLUSIVE」モードでテーブルレベルのロックを取得し、必要に応じて、競合するロックが解放されるまで待機します。このように明示的にテーブルをロックすると、他のトランザクションやセッションからテーブルへの読み書きを実行しようとした場合、その操作で待機状態が発生します。あるユーザーが作成した明示的テーブルロックは、別のユーザーがそのテーブルからデータを選択したり、そのテーブルにデータをロードすることを一時的に禁止します。LOCK コマンドを含んでいるトランザクションが終了すると、ロックは解放されます。

制限が緩いテーブルロックは、書き込み操作など、テーブルを参照するコマンドによって黙示的に取得されます。例えば、テーブルからデータを読み取ろうとした際に、別のユーザーがテーブルを更新

していた場合、読み込もうとしたデータは、既に確定しているデータのスナップショットになります。(クエリがシリアライズ可能な分離ルールに違反しているときは、クエリが停止する場合もあります。)「同時書き込み操作を管理する」を参照してください。

DROP TABLE や TRUNCATE など、一部の DDL 操作は排他的ロックを生成します。このような操作により、データの読み取りが禁止されます。

ロックの競合が発生すると、Amazon Redshift はエラーメッセージを表示して、競合するトランザクションを開始したユーザーに警告します。ロックの競合が発生したトランザクションは停止されます。ロックの競合が発生すると、Amazon Redshift は必ず STL_TR_CONFLICT テーブルにエントリを書き込みます。

構文

```
LOCK [ TABLE ] table_name [, ...]
```

パラメータ

TABLE

オプションキーワード

table_name

ロックするテーブルの名前。テーブル名のカンマ区切りリストを使って、複数のテーブルをロックできます。ビューをロックすることはできません。

例

```
begin;

lock event, sales;

...
```

MERGE

ソーステーブルの行を条件付きでターゲットテーブルにマージします。従来、これは複数の insert、update、delete ステートメントを別々に使用することによってのみ実現していまし

た。MERGE で組み合わせることができる操作の詳細については、「UPDATE」、「DELETE」、「INSERT」を参照してください。

構文

```
MERGE INTO target_table
USING source_table [ [ AS ] alias ]
ON match_condition
[ WHEN MATCHED THEN { UPDATE SET col_name = { expr } [,...] | DELETE }
WHEN NOT MATCHED THEN INSERT [ ( col_name [,...] ) ] VALUES ( { expr } [, ...] ) |
REMOVE DUPLICATES ]
```

パラメータ

target_table

MERGE ステートメントがマージされる一時または永続テーブル。

source_table

target_table にマージする行を提供する一時または永続テーブル。source_table は、Spectrum テーブルにすることもできます。

alias

source_table の一時的な代替名。

このパラメータはオプションです。alias の先頭に AS を付けることもできます。

match_condition

ソーステーブルの列とターゲットテーブルの列の間に等しい述語を指定します。これを使用して、source_table の行が target_table の行と一致するかどうかを判断します。条件が満たされると、MERGE はその行に対して matched_clause を実行します。それ以外の場合は、MERGE はその行に対して not_matched_clause を実行します。

WHEN MATCHED

ソース行とターゲット行の一致条件が True と評価された場合に実行するアクションを指定します。UPDATE アクションまたは DELETE アクションのいずれかを指定できます。

UPDATE

target_table 内の一致した行を更新します。指定した col_name の値のみが更新されます。

DELETE

target_table 内の一致した行を削除します。

WHEN NOT MATCHED

一致条件が False または Unknown と評価された場合に実行するアクションを指定します。この句には INSERT 挿入アクションのみを指定できます。

INSERT

match_condition に従って、target_table のどの行とも一致しない source_table の target_table 行に挿入します。ターゲットの col_name は、任意の順序でリストできます。col_name の値を指定しない場合、デフォルトの順序は、テーブルのすべての列が宣言した順序になります。

col_name

修正する 1 つまたは複数の列名。ターゲット列を指定する際にテーブル名を含めないでください。

expr

col_name の新しい値を定義する式。

REMOVE DUPLICATES

MERGE コマンドを簡易モードで実行することを指定します。簡易モードには次の要件があります。

- target_table と source_table は、列数が同じで、列タイプも互換性がある必要があります。
- MERGE コマンドから WHEN 句、UPDATE 句、INSERT 句を省略します。
- MERGE コマンドで REMOVE DUPLICATES 句を使用します。

簡易モードの場合、MERGE は次の操作を行います。

- source_table と一致する target_table 内の行は、source_table の値と一致するように更新されます。
- target_table と一致しない source_table 内の行は、target_table に挿入されます。
- target_table 内の複数の行が source_table 内の同じ行と一致する場合、重複する行は削除されます。Amazon Redshift は 1 つの行を保持し、それを更新します。source_table 内の行と一致しない重複行は変更されません。

REMOVE DUPLICATES を使用すると、WHEN MATCHED や WHEN NOT MATCHED を使用するよりもパフォーマンスが向上します。target_table と source_table に互換性があ

り、target_table に重複業を保持する必要がない場合は、REMOVE DUPLICATES を使用することをお勧めします。

使用に関する注意事項

- MERGE ステートメントを実行するには、source_table と target_table の両方の所有者であるか、それらのテーブルの SELECT 権限を持っている必要があります。さらに、MERGE ステートメントに含まれるオペレーションに応じて、target_table の UPDATE、DELETE、および INSERT 権限が必要です。
- target_table をシステムテーブル、カタログテーブル、または外部テーブルにすることはできません。
- source_table と target_table を同じテーブルにすることはできません。
- MERGE ステートメントで WITH 句を使用することはできません。
- target_table 内の行を source_table 内の複数の行と一致させることはできません。

  次の例を考えます。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (1, 'Bob'), (2, 'John');
INSERT INTO source VALUES (1, 'Tony'), (1, 'Alice'), (3, 'Bill');

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.

MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);
ERROR: Found multiple matches to update the same tuple.
```

  ID 値が `1` の `source` テーブルに複数の行があるため、どちらの MERGE ステートメントでも操作は失敗します。

- match_condition と expr はSUPER 型の列を部分的に参照することはできません。例えば、SUPER 型のオブジェクトが配列または構造体である場合、その列の個々の要素を match_condition や expr に使用することはできませんが、列全体を使用することはできます。

次の例を考えます。

```
CREATE TABLE IF NOT EXISTS target (key INT, value SUPER);
CREATE TABLE IF NOT EXISTS source (key INT, value SUPER);

INSERT INTO target VALUES (1, JSON_PARSE('{"key": 88}'));
INSERT INTO source VALUES (1, ARRAY(1, 'John')), (2, ARRAY(2, 'Bill'));

MERGE INTO target USING source ON target.key = source.key
WHEN matched THEN UPDATE SET value = source.value[0]
WHEN NOT matched THEN INSERT VALUES (source.key, source.value[0]);
ERROR: Partial reference of SUPER column is not supported in MERGE statement.
```

SUPER 型の詳細については、「SUPER 型」を参照してください。

- source_table が大きい場合は、target_table と source_table の両方の結合列を分散キーとして定義するとパフォーマンスが向上する可能性があります。
- REMOVE DUPLICATES 句を使用するには、target_table に対する SELECT、INSERT、および DELETE アクセス許可が必要です。
- source_table はビューまたはサブクエリです。次に、source_table が重複する行を削除するサブクエリである MERGE ステートメントの例を示します。

```
MERGE INTO target
USING (SELECT id, name FROM source GROUP BY 1, 2) as my_source
ON target.id = my_source.id
WHEN MATCHED THEN UPDATE SET id = my_source.id, name = my_source.name
WHEN NOT MATCHED THEN INSERT VALUES (my_source.id, my_source.name);
```

例

次の例では、2 つのテーブルを作成し、それらに対して MERGE オペレーションを実行して、ターゲットテーブルの一致する行を更新し、一致しない行を挿入します。次に、ソーステーブルに別の値を挿入し、別の MERGE オペレーションを実行します。今度は、一致する行を削除して、ソーステーブルから新しい行を挿入します。

まず、ソーステーブルとターゲットテーブルを作成してデータを入力します。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));
```

```
INSERT INTO target VALUES (101, 'Bob'), (102, 'John'), (103, 'Susan');
INSERT INTO source VALUES (102, 'Tony'), (103, 'Alice'), (104, 'Bill');

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | John
 103 | Susan
(3 rows)

SELECT * FROM source;
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
(3 rows)
```

次に、ソーステーブルをターゲットテーブルにマージし、ターゲットテーブルを一致する行で更新し、一致しない行をソーステーブルから挿入します。

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN UPDATE SET id = source.id, name = source.name
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 102 | Tony
 103 | Alice
 104 | Bill
(4 rows)
```

なお、ID 値が 102 と 103 の行は、ターゲットテーブルの名前値と一致するように更新されます。また、ID 値が 104 で名前値が Bill の新しい行がターゲットテーブルに挿入されます。

次に、ソーステーブルに新しい行を挿入します。

```
INSERT INTO source VALUES (105, 'David');
```

```
SELECT * FROM source;
 id  |    name
-----+------------
 102 | Tony
 103 | Alice
 104 | Bill
 105 | David
(4 rows)
```

最後に、MERGE オペレーションを実行して、ターゲットテーブルから一致する行を削除し、一致しない行を挿入します。

```
MERGE INTO target USING source ON target.id = source.id
WHEN MATCHED THEN DELETE
WHEN NOT MATCHED THEN INSERT VALUES (source.id, source.name);

SELECT * FROM target;
 id  |    name
-----+------------
 101 | Bob
 105 | David
(2 rows)
```

ID 値が 102、103、104 の行がターゲットテーブルから削除され、ID 値が 105 で名前値が David の新しい行がターゲットテーブルに挿入されます。

次の例は、REMOVE DUPLICATES 句を使用した MERGE コマンドの簡略化された構文を示しています。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (11, 'Alice'), (23, 'Bill');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
11 | Alice
```

```
23 | David
22 | Clarence
(4 rows)
```

次の例は、source_table に一致する行がある場合に、target_table から重複する行を削除する、REMOVE DUPLICATES 句を使用した MERGE コマンドの簡素化された構文を示しています。

```
CREATE TABLE target (id INT, name CHAR(10));
CREATE TABLE source (id INT, name CHAR(10));

INSERT INTO target VALUES (30, 'Tony'), (30, 'Daisy'), (11, 'Alice'), (23, 'Bill'),
 (23, 'Nikki');
INSERT INTO source VALUES (23, 'David'), (22, 'Clarence');

MERGE INTO target USING source ON target.id = source.id REMOVE DUPLICATES;

SELECT * FROM target;
id | name
---+------------
30 | Tony
30 | Daisy
11 | Alice
23 | David
22 | Clarence
(5 rows)
```

MERGE の実行後、target_table には ID 値 23 の行が 1 つのみになります。source_table には ID 値 30 の行がなかったため、target_table には ID 値 30 の 2 つの重複行が残ります。

以下も参照してください。

INSERT, UPDATE, DELETE

PREPARE

実行用のステートメントを準備します。

PREPARE によって、準備済みステートメントが作成されます。PREPARE ステートメントを実行すると、指定されたステートメント (SELECT、INSERT、UPDATE、または DELETE) が解析、再出力、および計画されます。準備済みステートメントに対して EXECUTE コマンドを発行すると、Amazon Redshift は (指定したパラメータ値に基づいてパフォーマンスを改善するように) 必要に応じてクエリ実行計画を修正してから、その準備済みステートメントを実行することがあります。

構文

```
PREPARE plan_name [ (datatype [, ...] ) ] AS statement
```

パラメータ

plan_name

この特定の準備済みステートメントに付けられた任意の名前。この名前は単一セッション内で一意でなければならず、以降は事前に準備済みのステートメントの実行、またはその割当解除に使われます。

datatype

準備済みステートメントに対するパラメータのデータ型。準備済みステートメント自体のパラメータを参照するには、S1、S2 などと使います。

statement

SELECT、INSERT、UPDATE または DELETE の任意のステートメント。

使用に関する注意事項

準備済みステートメントでは、パラメータ (ステートメントの実行時にステートメントに代入される値) を使用できます。準備済みステートメントにパラメータを含めるには、PREPARE ステートメントにデータ型のリストを提供し、準備するステートメント自体で、$1、$2 などの表記を使って位置によりパラメータを参照します。このステートメントを実行する際に、EXECUTE ステートメントでこれらのパラメータの実際の値を指定します。詳細については、「EXECUTE」を参照してください。

準備済みステートメントは、現在のセッションの有効期間中のみ有効です。セッションが終了すると、準備済みステートメントは破棄されるため、再使用する場合は、作り直す必要があります。これは、単一の準備済みステートメントを複数の同時データベースクライアントで使用することはできないことも意味します。ただし、各クライアントは独自の準備済みステートメントを作成して使用することができます。準備済みステートメントは、DEALLOCATE コマンドを使って、手動で削除できます。

単一のセッションを使用して多数の類似したステートメントを実行する場合に、準備済みステートメントは、パフォーマンス面での最大のメリットをもたらします。前述したように、準備済みステー

トメントを新たに実行するたびに、Amazon Redshift は指定されたパラメータ値に基づいてパフォーマンスを向上させるため、クエリの実行計画を修正することがあります。Amazon Redshift が特定の EXECUTE ステートメントに選択したクエリ実行計画を確認するには、EXPLAIN コマンドを使用します。

クエリの計画と、クエリの最適化のために Amazon Redshift が収集した統計情報の詳細については、「ANALYZE コマンド」を参照してください。

例

一時テーブルを作成し、INSERT ステートメントを準備した上でそれを実行します。

```
DROP TABLE IF EXISTS prep1;
CREATE TABLE prep1 (c1 int, c2 char(20));
PREPARE prep_insert_plan (int, char)
AS insert into prep1 values ($1, $2);
EXECUTE prep_insert_plan (1, 'one');
EXECUTE prep_insert_plan (2, 'two');
EXECUTE prep_insert_plan (3, 'three');
DEALLOCATE prep_insert_plan;
```

SELECT ステートメントを準備して実行します。

```
PREPARE prep_select_plan (int)
AS select * from prep1 where c1 = $1;
EXECUTE prep_select_plan (2);
EXECUTE prep_select_plan (3);
DEALLOCATE prep_select_plan;
```

以下の資料も参照してください。

DEALLOCATE, EXECUTE

REFRESH MATERIALIZED VIEW

マテリアライズドビューの更新

マテリアライズドビューを作成する際、そのコンテンツには、その時点での基となるデータベーステーブルまたはテーブルの状態が反映されます。アプリケーションが基となるテーブルにあるデータを変更しても、マテリアライズドビューのデータは変更されません。マテリアライズドビューのデータは、`REFRESH MATERIALIZED VIEW` ステートメントを使用して随時更新できます。このステー

トメントを使用する際は、Amazon Redshift は、ベーステーブルまたはテーブルで行われた変更を特定し、特定した変更をマテリアライズドビューに適用します。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

構文

```
REFRESH MATERIALIZED VIEW mv_name
```

パラメータ

mv_name

更新するマテリアライズドビューの名前。

使用に関する注意事項

マテリアライズドビューの所有者のみが、そのマテリアライズドビューに対して REFRESH MATERIALIZED VIEW のオペレーションを実行できます。さらに、REFRESH MATERIALIZED VIEWを正常に実行するには、所有者が基となるベーステーブルに対する SELECT 権限を持っている必要があります。

REFRESH MATERIALIZED VIEW コマンドは、独自のトランザクションとして実行されます。Amazon Redshift トランザクションのセマンティクスに従って、ベーステーブルの、どのデータが REFRESH コマンドに表示されるか、または REFRESH コマンドによって加えられた変更が Amazon Redshift で実行中の他のトランザクションにいつ表示されるかを判断します。

- 増分マテリアライズドビューの場合、REFRESH MATERIALIZED VIEWはコミット済みのベーステーブル行のみを使用します。したがって、同じトランザクション内のデータ操作言語 (DML) ステートメントの後に更新操作が実行された場合、その DML ステートメントの変更は更新のために表示されません。
- マテリアライズドビューの完全更新の場合、REFRESH MATERIALIZED VIEWは、通常の Amazon Redshift トランザクションセマンティクスに従って、更新トランザクションに表示されるすべてのベーステーブル行を表示します。
- 入力引数型よっては、マテリアライズドビューの増分リフレッシュが Amazon Redshift でサポートされています。特定の入力引数型を持つ次の関数です。DATE (タイムスタンプ)、DATE_PART (日付、時刻、間隔、time-tz)、DATE_TRUNC (タイムスタンプ、間隔)。

- 増分更新は、ベーステーブルがデータ共有内にあるマテリアライズドビューでサポートされます。

Amazon Redshift の一部のオペレーションは、マテリアライズドビューに影響します。こうしたオペレーションの中には、マテリアライズドビューを定義するクエリで増分更新のための SQL 機能のみが使用されている場合でも、`REFRESH MATERIALIZED VIEW`オペレーションによってマテリアライズドビューが完全に再計算されるものがあります。次に例を示します。

- マテリアライズドビューが更新されない場合、バックグラウンドでのバキューム操作がブロックされることがあります。内部でしきい値の期間を定義したら、バキュームの実行が許可されます。バキュームを操作すると、依存するマテリアライズドビューは、次の更新時に (増分更新であっても) 再計算する対象としてマークされます。VACUUM の詳細については、「VACUUM」を参照してください。イベントとステータス変更の詳細については、「STL_MV_STATE」を参照してください。
- ベーステーブルで実行されるユーザー操作には、次に REFRESH が実行された際に、マテリアライズドビューが完全に再計算されるように強制するものがあります。このような操作の例としては、手動で呼び出された VACUUM、従来のサイズ変更、ALTER DISTKEY 操作、ALTER SORTKEY 操作、および切り捨て操作があります。自動オペレーションでは、次に REFRESH オペレーションが実行されたときにマテリアライズドビューが完全に再計算されることもあります。例えば、auto-vacuum delete オペレーションでは、完全な再計算が発生する可能性があります。イベントとステータス変更の詳細については、「STL_MV_STATE」を参照してください。

データ共有内のマテリアライズドビューの増分更新

Amazon Redshift は、ベーステーブルを共有している場合、コンシューマーデータ共有でのマテリアライズドビューの自動更新と増分更新をサポートしています。増分更新は、Amazon Redshift が前回の更新後に発生したベーステーブルの変更を特定し、マテリアライズドビューの対応するレコードのみを更新する操作です。この動作の詳細については、「CREATE MATERIALIZED VIEW」を参照してください。

増分更新の制約事項

Amazon Redshift では、現在のところ、次の SQL 要素のいずれかを使用してクエリで定義されたマテリアライズドビューの増分更新はサポートされていません。

- OUTER JOIN (右、左、またはフル)。
- 集合演算: UNION、INTERSECT、EXCEPT、MINUS。

- UNION ALL はサブクエリで発生し、集計関数または GROUP BY 句がクエリに存在する場合に使用します。
- 集計関数は、MEDIAN、PERCENTILE_CONT、LISTAGG、STDDEV_SAMP、STDDEV_POP、APPROXIMATE COUNT、APPROXIMATE PERCENTILE、およびビット単位の集計関数です。

> **Note**
>
> COUNT、SUM、MIN、MAX、および AVG 集計関数がサポートされています。

- DISTINCT COUNT、DISTINCT SUM などの DISTINCT 集計関数。
- ウィンドウ関数。
- 共通部分式の最適化など、クエリの最適化に一時テーブルを使用するクエリ。
- サブクエリ
- マテリアライズドビューを定義するクエリで以下の形式を参照する外部テーブル。
  - Delta Lake
  - Hudi

  増分更新は、プレビュートラック上の他の形式を参照する外部テーブルを使用して定義されたマテリアライズドビューでサポートされています。プレビュークラスターのセットアップの詳細については、「Amazon Redshift 管理ガイド」の「プレビュークラスターの作成」を参照してください。プレビューワークグループの設定の詳細については、「Amazon Redshift 管理ガイド」の「プレビューワークグループの作成」を参照してください。
- 日時関数、RANDOM、STABLE 以外のユーザー定義関数など、可変関数。
- ゼロ ETL 統合の増分更新に関する制限については、「Amazon Redshift とのゼロ ETL 統合を使用する場合の考慮事項」を参照してください。

VACUUM などのバックグラウンド操作がマテリアライズドビューの更新オペレーションに与える影響など、マテリアライズドビューの制限の詳細については、使用に関する注意事項 を参照してください。

例

次の例では、マテリアライズドビュー `tickets_mv` を更新します。

```
REFRESH MATERIALIZED VIEW tickets_mv;
```

RESET

設定パラメータの値をデフォルト値に戻します。

指定した単一のパラメータ、またはすべてのパラメータを同時にリセットできます。単一のパラメータを指定値に設定するには、SETコマンドを使用します。パラメータの現在の値を表示するには、SHOWコマンドを使用します。

構文

```
RESET { parameter_name | ALL }
```

次のステートメントは、セッションコンテキスト変数の値を NULL に設定します。

```
RESET { variable_name | ALL }
```

パラメータ

parameter_name

リセットするパラメータの名前。パラメータの詳細については、「サーバー設定の変更」を参照してください。

すべて

すべてのセッションコンテキスト変数を含む、すべてのランタイムパラメータをリセットします。

変数

リセットする変数の名前。RESET の値がセッションコンテキスト変数の場合、Amazon Redshift はその値を NULL に設定します。

例

次の例では、query_groupパラメータをデフォルト値にリセットします。

```
reset query_group;
```

次の例では、すべてのランタイムパラメータをデフォルト値にリセットします。

```
reset all;
```

次の例では、コンテキスト変数をリセットします。

```
RESET app_context.user_id;
```

REVOKE

テーブルの作成、削除、または更新を行うためのアクセス許可をユーザーまたはロールから削除します。

ON SCHEMA 構文を使用するデータベースユーザーおよびロールには、外部スキーマに対する USAGE アクセス許可の GRANT (付与) または REVOKE (取り消し) のみを行うことができます。AWS Lake Formation で ON EXTERNAL SCHEMA を使用する場合は、AWS Identity and Access Management(IAM) ロールに対して、アクセス許可の GRANT (付与) および REVOKE (取り消し) のみを行うことができます。アクセス許可のリストについては、構文を参照してください。

ストアドプロシージャの場合、USAGE ON LANGUAGE `plpgsql` アクセス許可はデフォルトで PUBLIC に付与されます。EXECUTE ON PROCEDURE アクセス許可は、デフォルトで所有者とスーパーユーザーにのみ付与されます。

REVOKE コマンドで、削除するアクセス許可を指定します。アクセス許可を付与するには、GRANT コマンドを使用します。

構文

```
REVOKE [ GRANT OPTION FOR ]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | REFERENCES | ALTER | TRUNCATE } [,...] |
 ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...] | ALL TABLES IN SCHEMA schema_name [, ...] }
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
{ { CREATE | TEMPORARY | TEMP | ALTER } [,...] | ALL [ PRIVILEGES ] }
ON DATABASE db_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
{ { CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
ON SCHEMA schema_name [, ...]
FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
EXECUTE
    ON FUNCTION function_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
{ { EXECUTE } [,...] | ALL [ PRIVILEGES ] }
    ON PROCEDURE procedure_name ( [ [ argname ] argtype [, ...] ] ) [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]


REVOKE [ GRANT OPTION FOR ]
USAGE
    ON LANGUAGE language_name [, ...]
    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
[ RESTRICT ]
```

テーブルの列レベルのアクセス許可の取り消し

Amazon Redshift テーブルとビューに対する列レベルのアクセス許可の構文を次に示します。

```
REVOKE { { SELECT | UPDATE } ( column_name [, ...] ) [, ...] | ALL [ PRIVILEGES ]
 ( column_name [,...] ) }
     ON { [ TABLE ] table_name [, ...] }
     FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
     [ RESTRICT ]
```

ASSUMEROLE アクセス許可の取り消し

以下に、指定されたロールを持つユーザーおよびグループから ASSUMEROLE アクセス許可を取り消すための構文を示します。

```
REVOKE ASSUMEROLE
```

```
    ON { 'iam_role' [, ...]  | default | ALL }
    FROM { user_name | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    FOR { ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL }
```

Lake Formation の Redshift Spectrum のアクセス許可の取り消し

以下に、Redshift Spectrum と Lake Formation の統合構文を示します。

```
REVOKE [ GRANT OPTION FOR ]
{ SELECT | ALL [ PRIVILEGES ] } ( column_list )
    ON EXTERNAL TABLE schema_name.table_name
    FROM { IAM_ROLE iam_role } [, ...]

REVOKE [ GRANT OPTION FOR ]
{ { SELECT | ALTER | DROP | DELETE | INSERT }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL TABLE schema_name.table_name [, ...]
    FROM { { IAM_ROLE iam_role } [, ...] | PUBLIC }

REVOKE [ GRANT OPTION FOR ]
{ { CREATE | ALTER | DROP }  [, ...] | ALL [ PRIVILEGES ] }
    ON EXTERNAL SCHEMA schema_name [, ...]
    FROM { IAM_ROLE iam_role } [, ...]
```

データ共有アクセス許可の取り消し

プロデューサー側のデータ共有のアクセス許可

以下は、REVOKE を使用してユーザーまたはロールから ALTER または SHARE アクセス許可を削除するための構文です。アクセス許可が取り消されたユーザーは、データ共有を変更したり、コンシューマーに使用を許可したりできなくなります。

```
REVOKE { ALTER | SHARE } ON DATASHARE datashare_name
 FROM { username [ WITH GRANT OPTION ] | ROLE role_name | GROUP group_name | PUBLIC }
 [, ...]
```

以下は、REVOKE を使用してデータ共有へのコンシューマのアクセスを削除するための構文です。

```
REVOKE USAGE
 ON DATASHARE datashare_name
 FROM NAMESPACE 'namespaceGUID' [, ...] | ACCOUNT 'accountnumber' [ VIA DATA CATALOG ]
 [, ...]
```

以下は、Lake Formation アカウントからデータ共有の使用を取り消す例です。

```
REVOKE USAGE ON DATASHARE salesshare FROM ACCOUNT '123456789012' VIA DATA CATALOG;
```

コンシューマー側のデータ共有のアクセス許可

以下は、データ共有から作成された特定のデータベースまたはスキーマに対するデータ共有使用許可の REVOKE 構文です。WITH PERMISSIONS 句で作成されたデータベースから使用許可を取り消しても、基礎となるオブジェクトに付与されたオブジェクトレベルの許可を含め、ユーザーまたはロールに付与した追加の許可は取り消されません。そのユーザーまたはロールに使用許可を再付与しても、使用を取り消す前に持っていた追加の権限はすべて保持されます。

```
REVOKE USAGE ON { DATABASE shared_database_name [, ...] | SCHEMA shared_schema}
 FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
```

スコープ付きアクセス許可の取り消し

スコープ設定アクセス許可を使用すると、データベースまたはスキーマ内の特定タイプのすべてのオブジェクトに対するアクセス許可をユーザーまたはロールに付与できます。スコープ設定アクセス許可を持つユーザーやロールは、データベースまたはスキーマ内の現在および将来のすべてのオブジェクトに対して指定されたアクセス許可を持ちます。

データベースレベルのスコープ付きアクセス許可の範囲は、SVV_DATABASE_PRIVILEGES で確認できます。スキーマレベルのスコープ付きアクセス許可の範囲は、SVV_SCHEMA_PRIVILEGES で確認できます。

以下は、ユーザーとロールからスコープ付きアクセス許可を取り消すための構文です。スコープ設定アクセス許可の詳細については、「スコープ設定アクセス許可」を参照してください。

```
REVOKE [ GRANT OPTION ]
{ CREATE | USAGE | ALTER } [,...] | ALL [ PRIVILEGES ] }
FOR SCHEMAS IN
DATABASE db_name
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ]
{ { SELECT | INSERT | UPDATE | DELETE | DROP | ALTER | TRUNCATE | REFERENCES }
 [, ...] } | ALL [PRIVILEGES] } }
FOR TABLES IN
{ SCHEMA schema_name [ DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]
```

```
REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR FUNCTIONS IN
{ SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ] { EXECUTE | ALL [ PRIVILEGES ] }
FOR PROCEDURES IN
{ SCHEMA schema_name [DATABASE db_name ] | DATABASE db_name }
FROM { username | ROLE role_name } [, ...]

REVOKE [ GRANT OPTION ] USAGE
FOR LANGUAGES IN
DATABASE db_name
FROM { username | ROLE role_name } [, ...]
```

スコープ設定アクセス許可では、関数とプロシージャのアクセス許可が区別されないことに注意してください。例えば、次のステートメントは、スキーマ Sales_schema で bob の関数とプロシージャの両方に対する EXECUTE アクセス許可を取り消します。

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

機械学習アクセス許可の取り消し

以下は、Amazon Redshift での機械学習モデルアクセス許可の構文です。

```
REVOKE [ GRANT OPTION FOR ]
    CREATE MODEL FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON MODEL model_name [, ...]

    FROM { username | ROLE role_name | GROUP group_name | PUBLIC } [, ...]
    [ RESTRICT ]
```

ロールアクセス許可の取り消し

以下に、Amazon Redshift でロールのアクセス許可を取り消すための構文を示します。

```
REVOKE [ ADMIN OPTION FOR ] { ROLE role_name } [, ...] FROM { user_name } [, ...]
```

```
REVOKE { ROLE role_name } [, ...] FROM { ROLE role_name } [, ...]
```

以下に、Amazon Redshift でシステムのアクセス許可を取り消すための構文を示します。

```
REVOKE
  {
    { CREATE USER | DROP USER | ALTER USER |
    CREATE SCHEMA | DROP SCHEMA |
    ALTER DEFAULT PRIVILEGES |
    ACCESS CATALOG |
    CREATE TABLE | DROP TABLE | ALTER TABLE |
    CREATE OR REPLACE FUNCTION | CREATE OR REPLACE EXTERNAL FUNCTION |
    DROP FUNCTION |
    CREATE OR REPLACE PROCEDURE | DROP PROCEDURE |
    CREATE OR REPLACE VIEW | DROP VIEW |
    CREATE MODEL | DROP MODEL |
    CREATE DATASHARE | ALTER DATASHARE | DROP DATASHARE |
    CREATE LIBRARY | DROP LIBRARY |
    CREATE ROLE | DROP ROLE
    TRUNCATE TABLE
    VACUUM | ANALYZE | CANCEL }[, ...]
  }
  | { ALL [ PRIVILEGES ] }
FROM { ROLE role_name } [, ...]
```

行レベルのセキュリティポリシーフィルターの説明アクセス許可の取り消し

次の内容は、EXPLAIN プランのクエリにおける行レベルのセキュリティポリシーフィルタを説明する許可を取り消す構文です。この権限は、REVOKE ステートメントを使用して取り消すことができます。

```
REVOKE EXPLAIN RLS FROM ROLE rolename
```

次の内容は、クエリの行レベルのセキュリティポリシーをバイパスする許可を付与する構文です。

```
REVOKE IGNORE RLS FROM ROLE rolename
```

次の内容は、指定された行レベルのセキュリティポリシーから許可を取り消す構文です。

```
REVOKE SELECT ON [ TABLE ] table_name [, ...]
            FROM RLS POLICY policy_name [, ...]
```

パラメータ

GRANT OPTION FOR

他のユーザーに特定の許可を付与するオプションのみを取り消し、許可自体は取り消しません。グループや PUBLIC の GRANT OPTION を取り消すことはできません。

SELECT

SELECT ステートメントを使用して、テーブルやビューからデータを選択する許可を取り消します。

INSERT

INSERT ステートメントまたは COPY ステートメントを使用して、データをテーブルにロードする許可を取り消します。

UPDATE

UPDATE ステートメントを使用して、テーブル列を更新する許可を取り消します。

DELETE

テーブルからデータ行を削除する許可を取り消します。

REFERENCES

外部キー制限を作成する許可を取り消します。参照先テーブルと参照元テーブルの両方で、この許可を取り消してください。

TRUNCATE

テーブルを切り捨てるアクセス許可を取り消します。このアクセス許可がない場合、テーブルの所有者またはスーパーユーザーだけがテーブルを切り捨てることができます。TRUNCATE コマンドの詳細については、「the section called “TRUNCATE”」を参照してください。

ALL [PRIVILEGES]

指定されたユーザーまたはグループから、使用可能なすべての許可を一括で取り消します。PRIVILEGES キーワードはオプションです。

> **Note**
>
> Amazon Redshift は、ルールおよび TRIGGER アクセス許可をサポートしていません。詳細については、「サポートされていない PostgreSQL 機能」を参照してください。

ALTER

データベースオブジェクトに応じて、ユーザーまたはユーザーグループから以下のアクセス許可を取り消します。

- テーブルの場合、ALTER はテーブルまたはビューを変更するアクセス許可を取り消します。詳細については、「ALTER TABLE」を参照してください。
- データベースの場合、ALTER はデータベースを変更するアクセス許可を取り消します。詳細については、「ALTER DATABASE」を参照してください。
- スキーマの場合、ALTER はスキーマの変更を取り消すことを許可します。詳細については、「ALTER SCHEMA」を参照してください。
- 外部テーブルの場合、ALTER は、Lake Formation で有効になっている AWS Glue Data Catalog 内のテーブルを変更するアクセス許可を取り消します。この許可は、Lake Formation を使用する場合にのみ適用されます。

DROP

テーブルを削除する許可を取り消します。この許可は、Amazon Redshift および Lake Formation が有効になっている AWS Glue Data Catalog に対して適用されます。

ASSUMEROLE

指定したロールを持つユーザー、ロール、またはグループから COPY、UNLOAD、EXTERNAL FUNCTION、または CREATE MODEL コマンドを実行するアクセス許可を取り消します。

ON [TABLE] table_name

テーブルまたはビューに関して、指定された許可を取り消します。TABLE キーワードはオプションです。

ON ALL TABLES IN SCHEMA schema_name

参照されたスキーマ内のすべてのテーブルに指定された許可を取り消します。

(column_name [,...]) ON TABLE table_name

Amazon Redshift テーブルまたはビューの指定した列のユーザー、グループ、または PUBLIC から指定された許可を取り消します。

(column_list) ON EXTERNAL TABLE schema_name.table_name

参照されるスキーマで、Lake Formation テーブルの指定された列の IAM ロールから、指定された許可を取り消します。

ON EXTERNAL TABLE schema_name.table_name

参照されるスキーマの指定された Lake Formation テーブルの IAM ロールから、指定された許可を取り消します。

ON EXTERNAL SCHEMA schema_name

参照されるスキーマの IAM ロールから、指定された許可を取り消します。

FROM IAM_ROLE iam_role

許可が取り消される IAM ロールを示します。

ROLE role_name

指定されたロールからアクセス許可を取り消します。

GROUP group_name

指定されたユーザーグループから許可を取り消します。

PUBLIC

すべてのユーザーから、指定された許可を取り消します。PUBLIC は、常にすべてのユーザーを含むグループを表します。各ユーザーのアクセス許可は、PUBLIC に付与されたアクセス許可、ユーザーが属するグループに付与されたアクセス許可、およびユーザーに個別に付与されたアクセス許可のすべてで構成されます。

Lake Formation 外部テーブルから PUBLIC を取り消すと、Lake Formation の everyone グループからアクセス許可が取り消されます。

CREATE

データベースオブジェクトに応じて、ユーザーまたはグループから以下のアクセス許可を取り消します。

- データベースでは、REVOKE の CREATE 句を使用してユーザーがデータベース内でスキーマを作成することを阻止します。
- スキーマでは、REVOKE の CREATE 句を使用してユーザーがスキーマ内でオブジェクトを作成することを阻止します。オブジェクトの名前を変更するには、CREATE アクセス許可を持ち、名前を変更するオブジェクトを所有している必要があります。

> Note
>
> デフォルトでは、PUBLIC スキーマに対して、すべてのユーザーが CREATE と USAGE アクセス許可を持ちます。

TEMPORARY | TEMP

指定されたデータベースに一時テーブルを作成するアクセス許可を取り消します。

> Note
>
> デフォルトでは、PUBLIC グループの自動メンバーシップにより、一時テーブルを作成するアクセス許可がユーザーに付与されます。ユーザーが一時テーブルを作成するためのアクセス許可を削除するには、PUBLIC グループから TEMP アクセス許可を取り消してから、特定のユーザーまたはユーザーのグループに対して、一時テーブルを作成するアクセス許可を明示的に付与します。

ON DATABASE db_name

指定されたデータベースに関するアクセス許可を取り消します。

USAGE

特定のスキーマ内のオブジェクトに対する USAGE アクセス許可を取り消します。ユーザーはこれらのオブジェクトにアクセスできなくなります。これらのオブジェクトに関する特定のアクションは、個別に取り消す必要があります (関数に対する EXECUTE アクセス許可など)。

> Note
>
> デフォルトでは、PUBLIC スキーマに対して、すべてのユーザーが CREATE と USAGE アクセス許可を持ちます。

ON SCHEMA schema_name

指定されたスキーマに関するアクセス許可を取り消します。スキーマアクセス許可を使用して、テーブルの作成を制御できます。データベースの CREATE アクセス許可は、スキーマの作成だけを制御します。

RESTRICT

ユーザーが直接付与したアクセス許可だけを取り消します。この動作がデフォルトです。

EXECUTE ON PROCEDURE procedure_name

特定のストアドプロシージャに対する EXECUTE アクセス許可を取り消します。ストアドプロシージャ名は重複する場合があるため、プロシージャの引数リストを含める必要があります。詳細については、「ストアドプロシージャの名前付け」を参照してください。

EXECUTE ON ALL PROCEDURES IN SCHEMA procedure_name

参照されるスキーマ内のすべてのプロシージャに対する指定されたアクセス許可を取り消します。

USAGE ON LANGUAGE language_name

言語に対する USAGE アクセス許可を取り消します。Python ユーザー定義関数 (UDF) の場合、`plpythonu`を使用します。SQL UDF の場合、`sql`を使用します。ストアドプロシージャの場合、`plpgsql`を使用します。

UDF を作成するには、SQL または `plpythonu` (Python) 用の言語に対する使用のアクセス権限が必要です。デフォルトでは、USAGE ON LANGUAGE SQL は PUBLIC に付与されます。ただし、特定のユーザーやグループに対しては USAGE ON LANGUAGE PLPYTHONU を明示的に付与する必要があります。

SQL の使用を取り消すには、最初に PUBLIC に対して使用を取り消します。次に、SQL UDF の作成を許可された特定のユーザーやグループにのみ、SQL の使用を許可します。次の例では、最初に PUBLIC に対して SQL の使用を取り消し、次にユーザーグループ `udf_devs` に使用を許可します。

```
revoke usage on language sql from PUBLIC;
grant usage on language sql to group udf_devs;
```

詳細については、「UDF のセキュリティとアクセス許可」を参照してください。

ストアドプロシージャの使用を取り消すには、最初に PUBLIC に対して使用を取り消します。次に、SQL UDF の作成を許可された特定のユーザーやグループにのみ、`plpgsql`の使用を許可します。詳細については、「ストアドプロシージャのセキュリティおよび権限 」を参照してください。

{ ALL | COPY | UNLOAD | EXTERNAL FUNCTION | CREATE MODEL } [, ...] の場合、

アクセス許可を取り消す SQL コマンドを指定します。ALL を指定することで、COPY、UNLOAD、EXTERNAL FUNCTION、および CREATE MODEL ステートメントに対するアクセス許可を取り消すことができます。この句は、ASSUMEROLE アクセス許可の取り消しにのみ適用されます。

ALTER

データ共有を所有していないユーザーまたはユーザーグループがデータ共有を変更できるようにする、ユーザーまたはユーザーグループの ALTER アクセス許可を取り消します。このアクセス許可は、データ共有にオブジェクトを追加または削除したり、プロパティ PUBLICACCESSIBLE を設定したりするために必要です。詳細については、「ALTER DATASHARE」を参照してください。

SHARE

データ共有にコンシューマーをするための、ユーザーおよびユーザーグループのアクセス許可を取り消します。特定のコンシューマークラスターからデータ共有にアクセスするのを停止するには、このアクセス許可を取り消す必要があります。

ON DATASHARE datashare_name

参照されるデータ共有に対する指定されたアクセス許可を付与します。

FROM ユーザーネーム

アクセス許可を失うユーザーを示します。

FROM GROUP group_name

アクセス許可を失うユーザーグループを示します。

WITH GRANT OPTION

アクセス許可を失うユーザーが、他のユーザーの同じアクセス許可を取り消すことができることを示します。グループや PUBLIC の WITH GRANT OPTION を取り消すことはできません。

USAGE

同じアカウント内のコンシューマーアカウントまたは名前空間に対して USAGE が取り消された場合、アカウント内の指定されたコンシューマーアカウントまたは名前空間は、読み込み専用でデータ共有およびデータ共有のオブジェクトにアクセスできません。

USAGE アクセス許可を取り消すと、コンシューマーからのデータ共有へのアクセス権が取り消されます。

FROM NAMESPACE 'clusternamespace GUID'

データ共有へのアクセス許可を失うコンシューマーと同じアカウントの名前空間を示します。名前空間は、128 ビットの英数字のグローバル一意識別子 (GUID) を使用します。

FROM ACCOUNT 'accountnumber' [VIA DATA CATALOG]

データ共有へのアクセス許可を失うコンシューマーと別のアカウントのアカウント番号を示します。「VIA DATA CATALOG」を指定すると、データ共有の使用を Lake Formation のアカウントから取り消すことになります。アカウント番号を省略すると、クラスターを所有するアカウントから取り消すことになります。

ON DATABASE shared_database_name> [, ...]

指定されたデータ共有に作成された指定されたデータベースに対する指定された使用アクセス許可を取り消します。

ON SCHEMA shared_schema

指定されたデータ共有に作成された指定されたスキーマに対する指定されたアクセス許可を取り消します。

FOR { SCHEMAS | TABLES | FUNCTIONS | PROCEDURES | LANGUAGES } IN

アクセス許可を取り消すデータベースオブジェクトを指定します。IN に続くパラメータは、取り消されたアクセス許可のスコープを定義します。

CREATE MODEL

指定されたデータベースに機械学習モデルを作成する CREATE MODEL アクセス許可を取り消します。

ON MODEL model_name

特定のモデルについての EXECUTE アクセス許可を取り消します。

ACCESS CATALOG

ロールがアクセスできるオブジェクトの関連メタデータを表示するアクセス許可を取り消します。

[ADMIN OPTION FOR] { role } [, ...]

指定された (WITH ADMIN OPTION を使用している) ユーザーから取り消すロール。

FROM { role } [, ...]

取り消すために指定したロールが付与されているロール。

使用に関する注意事項

REVOKE の使用上の注意事項の詳細については、「the section called “使用に関する注意事項”」を参照してください。

例

REVOKE の使用方法の例については、「the section called “例”」を参照してください。

使用に関する注意事項

オブジェクトから権限を取り消すには、次の条件のうち 1 つを満たす必要があります。

- オブジェクトの所有者であること。
- スーパーユーザーであること。
- そのオブジェクトと権限に関する付与権限があること。

  例えば、次のコマンドは、employees テーブルに対する SELECT コマンドの実行と、他のユーザーに対する同じ権限の付与と取り消しの両方をユーザー HR に許可します。

  ```
  grant select on table employees to HR with grant option;
  ```

  HR は、SELECT 以外のオペレーションに関する権限や employees 以外のテーブルに関する権限を取り消すことはできません。

スーパーユーザーは、オブジェクトの権限を設定する GRANT コマンドと REVOKE コマンドに関係なく、すべてのオブジェクトにアクセスできます。

PUBLIC は、常にすべてのユーザーを含むグループを表します。デフォルトでは、PUBLIC スキーマに対して、PUBLIC のすべてのメンバーが CREATE 特権および USAGE 特権を持ちます。PUBLIC スキーマのすべてのユーザーのアクセス許可を制限するには、まず PUBLIC スキーマの PUBLIC からすべてのアクセス許可を取り消し、次に特定のユーザーまたはグループに権限を付与します。次の例では、PUBLIC スキーマのテーブル作成権限を管理します。

```
revoke create on schema public from public;
```

Lake Formation テーブルから権限を取り消すには、テーブルの外部スキーマに関連付けられた IAM ロールに、外部テーブルから許可を取り消す権限が必要です。次の例では、IAM ロール `myGrantor` に関連付けられた外部スキーマを作成します。IAM ロール `myGrantor` には、他のユーザーからア

クセス許可を取り消すアクセス許可があります。REVOKE コマンドは、外部スキーマに関連付けられている IAM ロール `myGrantor` のアクセス許可を使用して、IAM ロール `myGrantee` のアクセス許可を取り消します。

```
create external schema mySchema
from data catalog
database 'spectrum_db'
iam_role 'arn:aws:iam::123456789012:role/myGrantor'
create external database if not exists;
```

```
revoke select
on external table mySchema.mytable
from iam_role 'arn:aws:iam::123456789012:role/myGrantee';
```

> **Note**
>
> Lake Formation で有効になっている AWS Glue Data Catalog の `ALL` アクセス許可が、IAM ロールにもある場合は、この `ALL` アクセス許可は取り消されません。`SELECT` アクセス許可のみ取り消されます。Lake Formation アクセス許可は、Lake Formation コンソールで表示することができます。

ASSUMEROLE アクセス許可を取り消すための使用上の注意事項

以下の使用上の注意は、Amazon Redshift で ASSUMEROLE 権限を取り消す場合に適用されます。

データベースのスーパーユーザーのみが、ユーザーおよびグループの ASSUMEROLE 権限を取り消すことができます。スーパーユーザーは、常に ASSUMEROLE 権限を保持します。

ユーザーおよびグループに対して ASSUMEROLE 権限の使用を有効にするために、スーパーユーザーはクラスター上で次のステートメントを 1 回実行します。ユーザーおよびグループに ASSUMEROLE 権限を付与する前に、スーパーユーザーはクラスター上で次のステートメントを 1 回実行する必要があります。

```
revoke assumerole on all from public for all;
```

機械学習アクセス許可の取り消しに関する使用上の注意事項

ML 関数に関連するアクセス許可を直接付与または取り消すことはできません。ML 関数は ML モデルに属し、アクセス許可はモデルを通じて制御されます。代わりに、ML モデルに関連するアクセス

許可を取り消すことができます。次の例は、モデル customer_churn に関連付けられたすべてのユーザーから実行アクセス許可を取り消す方法を示しています。

```
REVOKE EXECUTE ON MODEL customer_churn FROM PUBLIC;
```

ML モデル customer_churn に対するすべてのアクセス許可をユーザーから取り消すこともできます。

```
REVOKE ALL on MODEL customer_churn FROM ml_user;
```

スキーマに ML 関数がある場合、その ML 関数が既に GRANT EXECUTE ON MODEL を通じて EXECUTE アクセス許可を持っている場合でも、ML 関数に関連する EXECUTE アクセス許可の付与または取り消しは失敗します。CREATE MODEL コマンドを使用して ML 関数を個別のスキーマに保持するときには、個別のスキーマを使用することをお勧めします。次の例は、その方法を示しています。

```
CREATE MODEL ml_schema.customer_churn
FROM customer_data
TARGET churn
FUNCTION ml_schema.customer_churn_prediction
IAM_ROLE default
SETTINGS (
 S3_BUCKET 'amzn-s3-demo-bucket'
);
```

例

次の例では、SALES テーブルに関する INSERT 権限を GUESTS ユーザーグループから取り消します。このコマンドを実行すると、GUESTS のメンバーは、INSERT コマンドを使って SALES テーブルにデータをロードすることができなくなります。

```
revoke insert on table sales from group guests;
```

次の例では、QA_TICKIT スキーマのすべてのテーブルに対する SELECT 権限をユーザー fred から取り消します。

```
revoke select on all tables in schema qa_tickit from fred;
```

次の例では、SELECT 権限をユーザー bobr のビューから取り消します。

```
revoke select on table eventview from bobr;
```

次の例では、TICKIT データベースで一時テーブルを作成する権限をすべてのユーザーから取り消します。

```
revoke temporary on database tickit from public;
```

次の例では、cust_profileテーブルの cust_name 列 と cust_phone 列に対する SELECT 権限をユーザー user1 から取り消します。

```
revoke select(cust_name, cust_phone) on cust_profile from user1;
```

次の例では、sales_groupグループから、cust_profileテーブルの cust_name 列と cust_phone 列に対する SELECT 権限と cust_contact_preference 列に対する UPDATE 権限を取り消します。

```
revoke select(cust_name, cust_phone), update(cust_contact_preference) on cust_profile
 from group sales_group;
```

次の例では、ALL キーワードを使用して、sales_adminグループ cust_profile テーブルの 3 つの列に対する SELECT 権限と UPDATE 権限の両方を取り消す方法を説明します。

```
revoke ALL(cust_name, cust_phone,cust_contact_preference) on cust_profile from group
 sales_admin;
```

次の例では、cust_profile_vwビューの cust_name 列に対する SELECT 権限を user2 ユーザーから取り消す方法を説明します。

```
revoke select(cust_name) on cust_profile_vw from user2;
```

データ共有経由で作成されたデータベースから USAGE アクセス許可を取り消す例

次の例では、13b8833d-17c6-4f16-8fe4-1a018f5ed00d 名前空間から salesshare データ共有へのアクセスを取り消します。

```
REVOKE USAGE ON DATASHARE salesshare FROM NAMESPACE
 '13b8833d-17c6-4f16-8fe4-1a018f5ed00d';
```

次の例では、sales_db に対する USAGE アクセス許可を Bob から取り消します。

```
REVOKE USAGE ON DATABASE sales_db FROM Bob;
```

次の例では、sales_schema に対する USAGE アクセス許可を Analyst_role から REVOKE します。

```
REVOKE USAGE ON SCHEMA sales_schema FROM ROLE Analyst_role;
```

スコープ付きアクセス許可を取り消す例

次の例では、Sales_db データベース内の現在および将来のすべてのスキーマの使用を Sales ロールから取り消します。

```
REVOKE USAGE FOR SCHEMAS IN DATABASE Sales_db FROM ROLE Sales;
```

次の例では、Sales_db データベース内の現在および将来のすべてのテーブルに対する SELECT アクセス許可をユーザー alice に付与する権限を取り消します。alice は Sales_db 内のすべてのテーブルへのアクセスを保持します。

```
REVOKE GRANT OPTION SELECT FOR TABLES IN DATABASE Sales_db FROM alice;
```

次の例では、Sales_schema スキーマ内の関数の EXECUTE アクセス許可をユーザー bob から削除します。

```
REVOKE EXECUTE FOR FUNCTIONS IN SCHEMA Sales_schema FROM bob;
```

次の例では、ShareDb データベースの ShareSchema スキーマ内のすべてのテーブルに対するすべてのアクセス許可を Sales ロールから削除します。スキーマを指定するとき、2 つの部分からなる形式 database.schema を使用してスキーマのデータベースを指定することもできます。

```
REVOKE ALL FOR TABLES IN SCHEMA ShareDb.ShareSchema FROM ROLE Sales;
```

次の例では、前の例と同じです。2 つの部分からなる形式を使用する代わりに、DATABASE キーワードを使用してスキーマのデータベースを指定できます。

```
REVOKE ALL FOR TABLES IN SCHEMA ShareSchema DATABASE ShareDb FROM ROLE Sales;
```

AssumeROLE 権限を取り消す例

次に、ASSUMEROLE 権限を取り消す例を示します。

スーパーユーザーは、クラスターで次のステートメントを 1 回実行することによって、ユーザーおよびグループに対して ASSUMEROLE 権限の使用を有効にする必要があります。

```
revoke assumerole on all from public for all;
```

次のステートメントは、すべてのオペレーションのすべてのロールについて、ユーザー reg_user1 からの ASSUMEROLE 権限を取り消します。

```
revoke assumerole on all from reg_user1 for all;
```

ROLE 権限を取り消す例

次の例では、sample_role1 を sample_role2 から取り消します。

```
CREATE ROLE sample_role2;
GRANT ROLE sample_role1 TO ROLE sample_role2;
REVOKE ROLE sample_role1 FROM ROLE sample_role2;
```

次の例では、user1 からシステム権限を取り消します。

```
GRANT ROLE sys:DBA TO user1;
REVOKE ROLE sys:DBA FROM user1;
```

次の例では、user1 から sample_role1 と sample_role2 を取り消します。

```
CREATE ROLE sample_role1;
CREATE ROLE sample_role2;
GRANT ROLE sample_role1, ROLE sample_role2 TO user1;
REVOKE ROLE sample_role1, ROLE sample_role2 FROM user1;
```

次の例では、ADMIN OPTION を指定して user1 から sample_role2 を取り消します。

```
GRANT ROLE sample_role2 TO user1 WITH ADMIN OPTION;
REVOKE ADMIN OPTION FOR ROLE sample_role2 FROM user1;
REVOKE ROLE sample_role2 FROM user1;
```

次の例では、sample_role5 から sample_role1 と sample_role2 を取り消します。

```
CREATE ROLE sample_role5;
GRANT ROLE sample_role1, ROLE sample_role2 TO ROLE sample_role5;
```

```
REVOKE ROLE sample_role1, ROLE sample_role2 FROM ROLE sample_role5;
```

次の例では、sample_role1 のシステム権限、CREATE SCHEMA および DROP SCHEMA を取り消します。

```
GRANT CREATE SCHEMA, DROP SCHEMA TO ROLE sample_role1;
REVOKE CREATE SCHEMA, DROP SCHEMA FROM ROLE sample_role1;
```

ROLLBACK

現在のトランザクションを停止し、そのトランザクションで実行されたすべての更新を破棄します。

このコマンドは、ABORTコマンドと同じ機能を実行します。

構文

```
ROLLBACK [ WORK | TRANSACTION ]
```

パラメータ

WORK

オプションキーワード このキーワードは、ストアドプロシージャ内ではサポートされていません。

TRANSACTION

オプションキーワード WORK と TRANSACTION は同義語です。ストアドプロシージャ内ではいずれもサポートされていません。

ストアドプロシージャ内での ROLLBACK の使用方法については、トランザクションの管理を参照してください。

例

次の例では、テーブルを作成し、データがそのテーブルに挿入されるトランザクションを開始します。次に ROLLBACK コマンドを実行すると、データ挿入がロールバックされ、テーブルは空の状態になります。

次のコマンドを実行すると、MOVIE_GROSS という名前のテーブルが作成されます。

```
create table movie_gross( name varchar(30), gross bigint );
```

次のコマンドセットを実行すると、2 つのデータ行をテーブルに挿入するトランザクションが開始されます。

```
begin;

insert into movie_gross values ( 'Raiders of the Lost Ark', 23400000);

insert into movie_gross values ( 'Star Wars', 10000000 );
```

その後、次のコマンドを実行すると、テーブルからデータが選択され、挿入が正常に実行されたことが示されます。

```
select * from movie_gross;
```

コマンド出力に、両方の行が正常に挿入されたことが示されます。

```
name           |  gross
-------------------------+----------
Raiders of the Lost Ark | 23400000
Star Wars               | 10000000
(2 rows)
```

このコマンドはデータ変更を、トランザクションの開始時点までロールバックします。

```
rollback;
```

テーブルからデータを選択すると、空のテーブルが表示されます。

```
select * from movie_gross;

name | gross
------+-------
(0 rows)
```

SELECT

テーブル、ビュー、およびユーザー定義関数から行を返します。

> **Note**
>
> 単一 SQL ステートメントの最大サイズは 16 MB です。

構文

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number | [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...] ]
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ [ START WITH expression ] CONNECT BY expression ]
[ GROUP BY expression [, ...] ]
[ HAVING condition ]
[ QUALIFY condition ]
[ { UNION | ALL | INTERSECT | EXCEPT | MINUS } query ]
[ ORDER BY expression [ ASC | DESC ] ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

トピック

WITH 句

WITH 句は、クエリ内の SELECT リストに先行するオプション句です。WITH 句は、1 つまたは複数の common_table_expressions を定義します。各共通テーブル式 (CTE) は、ビュー定義に似ている一時テーブルを定義します。これらの一時テーブルは、FROM 句で参照できます。それらは、所属するクエリが実行されている間にのみ使用されます。WITH 句内の各 CTE は、テーブル名、列名のオプションリスト、およびテーブルに対して評価を実行するクエリ表現 (SELECT ステートメント) を指定します。一時テーブル名を定義しているのと同じクエリ式の FROM 句で一時テーブル名を参照すると、CTE は再帰的になります。

WITH 句のサブクエリは、単一のクエリ実行中に、使用可能なテーブルを効率的に定義します。SELECT ステートメントの本文内でサブクエリを使用することで、すべてのケースで同じ結果を実現できますが、WITH 句のサブクエリの方が、読み書きが簡単になることがあります。可能な場合は、複数回参照される、WITH 句のサブクエリは、一般的な副次式として最適化されます。つまり、一度 WITH サブクエリを評価すると、その結果を再利用することができるということです。(一般的な副次式は、WITH 句内で定義される副次式に制限されない点に注意してください。)

構文

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
```

ここで、common_table_expression は、非再帰的または再帰的のいずれかになります。非再帰形式は次のとおりです。

```
CTE_table_name [ ( column_name [, ...] ) ] AS ( query )
```

以下は、common_table_expression の再帰形式です。

```
CTE_table_name (column_name [, ...] ) AS ( recursive_query )
```

パラメータ

RECURSIVE

クエリを再帰的な CTE として識別するキーワード。WITH 句で定義された common_table_expression が再帰的である場合、このキーワードは必須です。WITH 句に複数の再帰的な CTE が含まれている場合でも、WITH キーワードの直後に RECURSIVE キーワードを指定できるのは 1 回だけです。一般に、再帰的な CTE は、2 つの部分で構成される UNION ALL サブクエリです。

common_table_expression

FROM 句 で参照できる一時テーブルを定義し、それが属するクエリの実行中にのみ使用されます。

CTE_table_name

WITH 句のサブクエリの結果を定義する一時テーブルの一意な名前。単一の WITH 句内で重複する名前を使用することはできません。各サブクエリには、FROM 句で参照可能なテーブル名を付ける必要があります。

column_name

WITH 句サブクエリの出力列名を、カンマで区切ったリスト。指定された列名の数は、サブクエリで定義した列数以下でなければなりません。非再帰的な CTE の場合、column_name 句はオプションです。再帰的な CTE の場合、column_name リストが必要です。

query

Amazon Redshift がサポートする任意の SELECT クエリ。「SELECT」を参照してください。

recursive_query

2 つの SELECT サブクエリから構成される UNION ALL クエリ。

- 最初の SELECT サブクエリには、同じ CTE_table_name への再帰リファレンスがありません。再帰の最初のシードである結果セットを返します。この部分は、初期メンバーまたはシードメンバーと呼ばれます。
- 2 番目の SELECT サブクエリは、FROM 句で同じ CTE_table_name を参照します。これは、再帰メンバーと呼びます。recursive_query には、recursive_query を終了するための WHERE 条件が含まれています。

使用に関する注意事項

次の SQL ステートメントで WITH 句を使用できます。

- SELECT
- SELECT INTO
- CREATE TABLE AS
- CREATE VIEW
- DECLARE

- EXPLAIN
- INSERT INTO...SELECT
- PREPARE
- UPDATE (WHERE 句のサブクエリ内。サブクエリで再帰的な CTE を定義することはできません。再帰的な CTE は、UPDATE 句の前に配置する必要があります)。
- DELETE

WITH 句を含んでいるクエリの FROM 句が、WITH 句によって定義されたテーブルを参照していない場合、含まれている WITH 句は無視された上でクエリは通常どおり実行されます。

WITH 句のサブクエリで定義されたテーブルは、WITH 句が開始した SELECT クエリの範囲でのみ参照可能です。例えば、このようなテーブルは、SELECT リスト、WHERE 句、または HAVING 句内のサブクエリの FROM 句で参照できます。サブクエリ内で WITH 句を使用し、メインクエリまたは別のサブクエリの FROM 句でそのテーブルを参照することはできません。このクエリパターンを使用すると、WITH 句のテーブルに対して、`relation table_name doesn't exist`という形式のエラーメッセージが発生します。

WITH 句のサブクエリ内で、別の WITH 句を指定することはできません。

WITH 句のサブクエリによって定義されたテーブルに対して、前方参照を作成することはできません。例えば次のクエリでは、テーブル W1 の定義内でテーブル W2 への前方参照を設定しているため、エラーが帰されます。

```
with w1 as (select * from w2), w2 as (select * from w1)
select * from sales;
ERROR:  relation "w2" does not exist
```

WITH 句のサブクエリは、SELECT INTO ステートメントを構成できません。しかし、SELECT INTO ステートメント内で WITH 句を使用することは可能です。

再帰的なテーブル共通式

再帰共通テーブル式 (CTE) はそれ自体を参照する CTE です。再帰的な CTE は、従業員とマネージャー間のレポート関係を示す組織図などの階層データのクエリに役立ちます。「例: 再帰的な CTE」を参照してください。

もう 1 つの一般的な用途は、製品が多くのコンポーネントで構成され、各コンポーネント自体も他のコンポーネントまたはサブアセンブリで構成されている場合のマルチレベルの部品表です。

再帰クエリの 2 番目の SELECT サブクエリに WHERE 句を含めることで、再帰の深さを制限する必要があります。例については、「例: 再帰的な CTE」を参照してください。この制限を行わない場合は、次のようなエラーが発生する可能性があります。

- `Recursive CTE out of working buffers.`
- `Exceeded recursive CTE max rows limit, please add correct CTE termination predicates or change the max_recursion_rows parameter.`

> **Note**
>
> `max_recursion_rows` は、無限再帰ループを防ぐために再帰 CTE が返すことができる最大行数を設定するパラメータです。これをデフォルトよりも大きな値に変更しないことをお勧めします。これにより、クエリの無限再帰の問題がクラスター内で過剰なスペースを占有することを防ぎます。

再帰的な CTE の結果に対するソート順と制限を指定できます。再帰的な CTE の最終結果に、group by オプションと distinct オプションを含めることができます。

サブクエリ内で、WITH RECURSIVE 句を指定することはできません。recursive_query メンバーには、order by 句または limit 句を含めることはできません。

例

次の例では、WITH 句を含む最もシンプルなケースを示します。VENUECOPY という名前の WITH クエリは、VENUE テーブルからすべての行を選択します。次にメインクエリでは、VENUECOPY からすべての行を選択します。VENUECOPY テーブルは、このクエリの有効期間中だけ存在します。

```
with venuecopy as (select * from venue)
select * from venuecopy order by 1 limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
1 | Toyota Park                | Bridgeview      | IL         |          0
2 | Columbus Crew Stadium      | Columbus        | OH         |          0
3 | RFK Stadium                | Washington      | DC         |          0
4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
5 | Gillette Stadium           | Foxborough      | MA         |      68756
```

```
6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
7 | BMO Field                  | Toronto         | ON         |          0
8 | The Home Depot Center      | Carson          | CA         |          0
9 | Dick's Sporting Goods Park | Commerce City   | CO         |          0
v     10 | Pizza Hut Park             | Frisco          | TX         |          0
(10 rows)
```

次の例では、VENUE_SALES と TOP_VENUES という名前の 2 つのテーブルを生成する WITH 句を示します。2 番目の WITH 句テーブルは最初のテーブルから選択します。次に、メインクエリブロックの WHERE 句には、TOP_VENUES テーブルを制約するサブクエリが含まれています。

```
with venue_sales as
(select venuename, venuecity, sum(pricepaid) as venuename_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
group by venuename, venuecity),

top_venues as
(select venuename
from venue_sales
where venuename_sales > 800000)

select venuename, venuecity, venuestate,
sum(qtysold) as venue_qty,
sum(pricepaid) as venue_sales
from sales, venue, event
where venue.venueid=event.venueid and event.eventid=sales.eventid
and venuename in(select venuename from top_venues)
group by venuename, venuecity, venuestate
order by venuename;
```

```
        venuename       |   venuecity   | venuestate | venue_qty | venue_sales
------------------------+---------------+------------+-----------+-------------
August Wilson Theatre   | New York City | NY         |      3187 |  1032156.00
Biltmore Theatre        | New York City | NY         |      2629 |   828981.00
Charles Playhouse       | Boston        | MA         |      2502 |   857031.00
Ethel Barrymore Theatre | New York City | NY         |      2828 |   891172.00
Eugene O'Neill Theatre  | New York City | NY         |      2488 |   828950.00
Greek Theatre           | Los Angeles   | CA         |      2445 |   838918.00
Helen Hayes Theatre     | New York City | NY         |      2948 |   978765.00
Hilton Theatre          | New York City | NY         |      2999 |   885686.00
Imperial Theatre        | New York City | NY         |      2702 |   877993.00
```

```
Lunt-Fontanne Theatre    | New York City | NY          |      3326 |  1115182.00
Majestic Theatre         | New York City | NY          |      2549 |   894275.00
Nederlander Theatre      | New York City | NY          |      2934 |   936312.00
Pasadena Playhouse       | Pasadena      | CA          |      2739 |   820435.00
Winter Garden Theatre    | New York City | NY          |      2838 |   939257.00
(14 rows)
```

次の 2 つの例は、WITH 句サブクエリに基づいた、テーブル参照の範囲に関するルールをデモンストレーションしています。最初のクエリは実行されますが、2 番目のクエリは予想どおりのエラーが発生して失敗します。最初のクエリには、メインクエリの SELECT リスト内に WITH 句サブクエリが存在します。WITH 句によって定義されるテーブル (HOLIDAYS) は、SELECT リストのサブクエリの FROM 句で参照されます。

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join date on sales.dateid=date.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;

caldate   | daysales | dec25sales
-----------+----------+------------
2008-12-25 | 70402.00 |   70402.00
2008-12-31 | 12678.00 |   70402.00
(2 rows)
```

2 番目のクエリは SELECT リストのサブクエリ内だけでなく、メインクエリ内の HOLIDAYS テーブルを参照しようとしたため、失敗しました。メインクエリの参照は範囲外です。

```
select caldate, sum(pricepaid) as daysales,
(with holidays as (select * from date where holiday ='t')
select sum(pricepaid)
from sales join holidays on sales.dateid=holidays.dateid
where caldate='2008-12-25') as dec25sales
from sales join holidays on sales.dateid=holidays.dateid
where caldate in('2008-12-25','2008-12-31')
group by caldate
order by caldate;
```

```
ERROR:  relation "holidays" does not exist
```

例: 再帰的な CTE

次に示すのは、John に直接的または間接的に報告する従業員を返す再帰的な CTE の例です。再帰クエリには、再帰の深さを 4 レベル未満に制限する WHERE 句が含まれています。

```
--create and populate the sample table
  create table employee (
  id int,
  name varchar (20),
  manager_id int
  );

  insert into employee(id, name, manager_id)  values
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofía', 102),
(205, 'Zhang', 104);

--run the recursive query
  with recursive john_org(id, name, manager_id, level) as
( select id, name, manager_id, 1 as level
  from employee
  where name = 'John'
  union all
  select e.id, e.name, e.manager_id, level + 1 as next_level
  from employee e, john_org j
  where e.manager_id = j.id and level < 4
  )
 select distinct id, name, manager_id from john_org order by manager_id;
```

以下は、クエリの結果です。

```
  id        name      manager_id
------+-----------+--------------
 101    John          100
 102    Jorge         101
 103    Kwaku         101
 110    Liu           101
 201    Sofía         102
 106    Mateo         102
 110    Nikki         103
 104    Paulo         103
 105    Richard       103
 120    Saanvi        104
 200    Shirley       104
 205    Zhang         104
```

以下は、John が所属する部門の組織図です。

SELECT リスト

トピック

- 構文
- パラメータ
- 使用に関する注意事項
- 例

SELECT リストは、クエリが返す列、機能、および式を指定します。このリストは、クエリの出力を表しています。

SQL 関数の詳細については、「SQL 関数リファレンス」を参照してください。式の詳細については、「条件式」を参照してください。

構文

```
SELECT
[ TOP number ]
[ ALL | DISTINCT ] * | expression [ AS column_alias ] [, ...]
```

パラメータ

TOP number

TOP は引数として正の整数を取り、クライアントに返される行数を定義します。TOP 句に関する動作は、LIMIT 句に関する動作と同じです。返される行の数は固定されていますが、行のセットは固定されていません。一貫した行のセットを返すには、TOP または LIMIT を ORDER BY 句と組み合わせて使用します。

ALL

DISTINCT を指定しない場合、デフォルトの動作を定義する冗長キーワード。`SELECT ALL *` は、`SELECT *`と同じ意味です (すべての列のすべての行を選択し、重複を維持します)。

DISTINCT

1 つまたは複数の列の一致する値に基づいて、結果セットから重複する行を削除するオプション。

> **Note**
>
> アプリケーションが無効な外部キーまたはプライマリキーを許可する場合、クエリが不正な結果を返す可能性があります。例えば、プライマリキー列に含まれるすべての値が一意でない場合、SELECT DISTINCT クエリが重複した行を返すことがあります。詳細については、「テーブル制約の定義」を参照してください。

* (アスタリスク)

テーブルのコンテンツ全体を返します (すべての列とすべての行)。

expression

クエリによって参照されるテーブル内に存在する 1 つまたは複数の列から構成される式。式には、SQL 関数を含めることができます。次に例を示します。

```
avg(datediff(day, listtime, saletime))
```

AS column_alias

最終的な結果セットに使われる列のテンポラリ名。AS キーワードはオプションです。次に例を示します。

```
avg(datediff(day, listtime, saletime)) as avgwait
```

シンプルな列名ではない式に対して、エイリアスを指定しない場合、結果セットはその列に対してデフォルト名を適用します。

> **Note**
>
> エイリアスは、ターゲットリストで定義された直後に認識されます。同じターゲットリストで、その後に定義されている他の式でエイリアスを使用できます。以下に示しているのはこのポリシーの例です。
>
> ```
> select clicks / impressions as probability, round(100 * probability, 1) as
>  percentage from raw_data;
> ```
>
> エイリアスの側面参照の利点は、同じターゲットリストでより複雑な式を構築するときに、エイリアスの式を繰り返す必要がないことです。Amazon Redshift がこの種類の参照を解析するときは、前に定義されたエイリアスをインライン展開します。`FROM` 句に定義された名前が、以前にエイリアスを作成した式と同じである列がある場合、`FROM`句の列が優先されます。例えば、上記のクエリで、テーブルの raw_data に「probability」という名前の列がある場合、ターゲットリストの 2 番目の式の「probability」は、エイリアス名「probability」ではなく列を参照します。

使用に関する注意事項

TOP は SQL 式です。LIMIT 動作に対する選択肢を提供します。TOP と LIMIT を同じクエリで使用することはできません。

例

次の例では、SALES テーブルから 10 行を返します。クエリは TOP 句を使用していますが、ORDER BY 句が指定されていないため、予想できない行セットが返されます。

```
select top 10 *
from sales;
```

次のクエリは機能的には同じですが、TOP 句の代わりに LIMIT 句を使用します。

```
select *
from sales
limit 10;
```

次の例では、TOP 句を使用して、SALES テーブルから最初の 10 行を QTYSOLD 列の降順にソートして返します。

```
select top 10 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
--------+----------
8 |      518
8 |      520
8 |      574
8 |      718
8 |      868
8 |     2663
8 |     3396
8 |     3726
8 |     5250
8 |     6216
(10 rows)
```

次の例では、SALES テーブルから、最初の 2 つの QTYSOLD 値と SELLERID 値を、QTYSOLD 列をソートして返します。

```
select top 2 qtysold, sellerid
from sales
order by qtysold desc, sellerid;

qtysold | sellerid
--------+----------
8 |      518
8 |      520
(2 rows)
```

次の例は、CATEGORY テーブルからの異なるカテゴリグループのリストを示しています。

```
select distinct catgroup from category
```

```
order by 1;

catgroup
----------
Concerts
Shows
Sports
(3 rows)

--the same query, run without distinct
select catgroup from category
order by 1;

catgroup
----------
Concerts
Concerts
Concerts
Shows
Shows
Shows
Sports
Sports
Sports
Sports
Sports
(11 rows)
```

次の例では、2008 年 12 月の週の数の独自セットを返します。DISTINCT 句がないと、このステートメントは 31 行、つまりその月の各日に対して 1 行を返します。

```
select distinct week, month, year
from date
where month='DEC' and year=2008
order by 1, 2, 3;

week | month | year
-----+-------+------
49 | DEC   | 2008
50 | DEC   | 2008
51 | DEC   | 2008
52 | DEC   | 2008
53 | DEC   | 2008
```

```
(5 rows)
```

FROM 句

クエリ内の FROM 句は、データの選択下のテーブル参照 (テーブル、ビュー、サブクエリ) を一覧表示します。複数のテーブル参照が一覧表示されている場合、FROM 句または WHERE 句のいずれかの適切な構文を使って、テーブル参照を結合する必要があります。結合基準を指定していない場合、クエリはクロス結合 (デカルト積) として処理されます。

トピック

- 構文
- パラメータ
- 使用に関する注意事項
- PIVOT と UNPIVOT の例
- JOIN 句の例

構文

```
FROM table_reference [, ...]
```

ここで table_reference は、次のいずれかになります。

```
with_subquery_table_name [ table_alias ]
table_name [ * ] [ table_alias ]
( subquery ) [ table_alias ]
table_reference [ NATURAL ] join_type table_reference
   [ ON join_condition | USING ( join_column [, ...] ) ]
table_reference PIVOT (
   aggregate(expr) [ [ AS ] aggregate_alias ]
   FOR column_name IN ( expression [ AS ] in_alias [, ...] )
) [ table_alias ]
table_reference UNPIVOT [ INCLUDE NULLS | EXCLUDE NULLS ] (
   value_column_name
   FOR name_column_name IN ( column_reference [ [ AS ]
   in_alias ] [, ...] )
) [ table_alias ]
UNPIVOT expression AS value_alias [ AT attribute_alias ]
```

オプションの table_alias を使用して、テーブルと複雑なテーブル参照 (および必要に応じてその列) に、次のような一時名を付けることができます。

```
[ AS ] alias [ ( column_alias [, ...] ) ]
```

パラメータ

with_subquery_table_name

WITH 句 のサブクエリで定義されるテーブル。

table_name

テーブルまたはビューの名前。

alias

テーブルまたはビューの一時的な代替名。エイリアスは、サブクエリから生成されたテーブルに対して提供する必要があります。他のテーブル参照では、エイリアスはオプションです。AS キーワードは常にオプションです。テーブルエイリアスは、WHERE 句など、クエリの別の部分のテーブルを識別するため、便利なショートカットを提供します。次に例を示します。

```
select * from sales s, listing l
where s.listid=l.listid
```

column_alias

テーブルまたはビュー内の列に対する一時的な代替名。

subquery

テーブルに対して評価を実行するクエリ式。テーブルは、クエリの有効期間中のみ存在し、通常は名前またはエイリアスが与えられます。ただし、エイリアスは必須ではありません。また、サブクエリから生成されたテーブルに対して、列名を定義することもできます。サブクエリの結果を他のテーブルに結合する場合、および列をクエリ内のどこかで選択または拘束する場合、列のエイリアスの命名は重要です。

サブクエリには ORDER BY 句が含まれることがありますが、LIMIT または OFFSET 句も併せて指定しない場合、この句には効力がありません。

NATURAL

2 つのテーブル内で同じ名前を付けられた列のペアをすべて結合列として、自動的に使用する結合を定義します。明示的な結合条件は必要ありません。例えば、CATEGORY と EVENT の両

方のテーブルに CATID という名前の列が存在する場合、これらのテーブルの NATURAL 結合は CATID 列による結合です。

> **Note**
>
> NATURAL 結合を指定しても、結合対象のテーブルに同じ名前の列ペアが存在しない場合、クエリはデフォルト設定のクロス結合になります。

join_type

以下のいずれかの結合タイプを指定します。

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

クロス結合は非限定の結合で、2 つの表のデカルト積を返します。

内部結合と外部結合は限定結合です。これらの結合は、FROM 句の ON または USING 構文、または WHERE 句条件を使った (Natural 結合での) 黙示的な結合です。

内部結合は、結合条件、また結合列のリストに基づいて、一致する行だけを返します。外部結合は、同等の内部結合が返すすべての行に加え、「左側の」表、「右側の」表、または両方の表から一致しない行を返します。左の表は最初に一覧表示された表で、右の表は 2 番目に一覧表示された表です。一致しない行には、出力列のギャップを埋めるため、NULL が含まれます。

ON join_condition

結合列を ON キーワードに続く条件として記述する、結合タイプの指定。次に例を示します。

```
sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
```

USING (join_column [, ...])

結合列をかっこで一覧表示する結合の指定タイプ。複数の結合列を指定する場合、カンマによって区切ります。USING キーワードは、リストより前に指定する必要があります。例:

```
sales join listing
using (listid,eventid)
```

PIVOT

表形式のデータを読みやすい形式で表現するために、出力を行から列に変更します。出力が複数の列にわたって水平に表されます。PIVOT は、(集計式を使用して出力形式を指定する)集計が含まれる GROUP BY クエリに似ています。ただし GROUP BY とは対照的に、結果は行ではなく列で返されます。

PIVOT および UNPIVOT を使用してクエリする方法の例については、「PIVOT と UNPIVOT の例」をご確認ください。

UNPIVOT

UNPIVOT による列から行への変換 – この演算子は、入力テーブルまたはクエリ結果の結果列を行に変換して、出力を読みやすくします。UNPIVOT は、入力列のデータを 2 つの結果列 (name 列と value 列) として統合します。name 列には、入力の列名が行エントリとして格納されています。value 列には、集計の結果など、入力列の値が含まれます。例えばこれは、さまざまなカテゴリについての項目数などです。

UNPIVOT (SUPER) によるオブジェクトのピボット解除 – オブジェクトのピボット解除を実行できます。ここで、expression は別の FROM 句項目を参照する SUPER 式です。詳細については、「オブジェクトのピボット解除」を参照してください。また、JSON 形式のデータなど、半構造化データをクエリする方法を示す例も含まれています。

使用に関する注意事項

列を結合するには、データ型に互換性がなければなりません。

NATURAL または USING 結合は、中間結果セットの結合列の各ペアの一方だけを保持します。

ON 構文を使った結合は、中間結果セットの両方の結合列を保持します。

「WITH 句」も参照してください。

PIVOT と UNPIVOT の例

PIVOT と UNPIVOT は FROM 句のパラメータで、クエリ出力をそれぞれ行から列、列から行にローテーションします。クエリ結果は読みやすい表形式で示されます。次の例では、テストデータとクエリを使用してその使用方法を示します。

これらのパラメータおよび他のパラメータの詳細については、「FROM 句」を参照してください。

PIVOT の例

サンプルテーブルとデータをセットアップし、それを使用して後続のサンプルクエリを実行します。

```
CREATE TABLE part (
    partname varchar,
    manufacturer varchar,
    quality int,
    price decimal(12, 2)
);

INSERT INTO part VALUES ('prop', 'local parts co', 2, 10.00);
INSERT INTO part VALUES ('prop', 'big parts co', NULL, 9.00);
INSERT INTO part VALUES ('prop', 'small parts co', 1, 12.00);

INSERT INTO part VALUES ('rudder', 'local parts co', 1, 2.50);
INSERT INTO part VALUES ('rudder', 'big parts co', 2, 3.75);
INSERT INTO part VALUES ('rudder', 'small parts co', NULL, 1.90);

INSERT INTO part VALUES ('wing', 'local parts co', NULL, 7.50);
INSERT INTO part VALUES ('wing', 'big parts co', 1, 15.20);
INSERT INTO part VALUES ('wing', 'small parts co', NULL, 11.80);
```

price で AVG を集計した partname の PIVOT。

```
SELECT *
FROM (SELECT partname, price FROM part) PIVOT (
    AVG(price) FOR partname IN ('prop', 'rudder', 'wing')
);
```

このクエリでは次の出力が生成されます。

```
  prop   |  rudder  |  wing
---------+----------+---------
 10.33   | 2.71     | 11.50
```

前の例では、結果が列に変換されています。次の例は、平均価格を列ではなく行で返す GROUP BY クエリを示しています。

```
SELECT partname, avg(price)
```

```
FROM (SELECT partname, price FROM part)
WHERE partname IN ('prop', 'rudder', 'wing')
GROUP BY partname;
```

このクエリでは次の出力が生成されます。

```
 partname |  avg
----------+-------
 prop     | 10.33
 rudder   |  2.71
 wing     | 11.50
```

manufacturer を暗黙の列として扱う PIVOT の例。

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) FOR quality IN (1, 2, NULL)
);
```

このクエリでは次の出力が生成されます。

```
 manufacturer      | 1  | 2  | null
-------------------+----+----+------
 local parts co    | 1  | 1  |  1
 big parts co      | 1  | 1  |  1
 small parts co    | 1  | 0  |  2
```

PIVOT 定義で参照されていない入力テーブルの列は、結果テーブルに暗黙的に追加されます。前出の例での manufacturer 列が、これに当てはまります。またこの例からは、NULLが IN 演算子で有効な値であることもわかります。

上記の例の PIVOT では、GROUP BYを含む下記のクエリと同様の情報を返します。相違点は、列 2 と manufacturer small parts co に対し、PIVOT が値 0 を返していることです。GROUP BY クエリには対応する行が含まれていません。行が特定の列に入力するデータを持たない場合、ほとんどのケースで PIVOT は NULL を挿入します。ただし、カウント集計は NULL を返さないので、0 がデフォルト値として使用されます。

```
SELECT manufacturer, quality, count(*)
FROM (SELECT quality, manufacturer FROM part)
WHERE quality IN (1, 2) OR quality IS NULL
GROUP BY manufacturer, quality
```

```
ORDER BY manufacturer;
```

このクエリでは次の出力が生成されます。

```
 manufacturer        | quality | count
---------------------+---------+-------
 big parts co        |         |     1
 big parts co        |       2 |     1
 big parts co        |       1 |     1
 local parts co      |       2 |     1
 local parts co      |       1 |     1
 local parts co      |         |     1
 small parts co      |       1 |     1
 small parts co      |         |     2
```

PIVOT 演算子は、集計式および IN 演算子の各値に関して、オプションのエイリアスを受け入れます。エイリアスは、列名をカスタマイズするために使用されます。集計のエイリアスがない場合は、INリストのエイリアスのみが使用されます。それ以外の場合は、集計のエイリアスが列名に (名前を区切るために) アンダースコアとともに追加されます。

```
SELECT *
FROM (SELECT quality, manufacturer FROM part) PIVOT (
    count(*) AS count FOR quality IN (1 AS high, 2 AS low, NULL AS na)
);
```

このクエリでは次の出力が生成されます。

```
 manufacturer      | high_count  | low_count | na_count
-------------------+-------------+-----------+----------
 local parts co    |           1 |         1 |        1
 big parts co      |           1 |         1 |        1
 small parts co    |           1 |         0 |        2
```

次のサンプルテーブルとデータを設定し、これらを使用して後続のサンプルクエリを実行します。データは、複数のホテルの予約日を表しています。

```
CREATE TABLE bookings (
    booking_id int,
    hotel_code char(8),
    booking_date date,
    price decimal(12, 2)
```

```
);

INSERT INTO bookings VALUES (1, 'FOREST_L', '02/01/2023', 75.12);
INSERT INTO bookings VALUES (2, 'FOREST_L', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (3, 'FOREST_L', '02/04/2023', 85.54);

INSERT INTO bookings VALUES (4, 'FOREST_L', '02/08/2023', 75.00);
INSERT INTO bookings VALUES (5, 'FOREST_L', '02/11/2023', 75.00);
INSERT INTO bookings VALUES (6, 'FOREST_L', '02/14/2023', 90.00);

INSERT INTO bookings VALUES (7, 'FOREST_L', '02/21/2023', 60.00);
INSERT INTO bookings VALUES (8, 'FOREST_L', '02/22/2023', 85.00);
INSERT INTO bookings VALUES (9, 'FOREST_L', '02/27/2023', 90.00);

INSERT INTO bookings VALUES (10, 'DESERT_S', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (11, 'DESERT_S', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (12, 'DESERT_S', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (13, 'DESERT_S', '02/05/2023', 75.00);
INSERT INTO bookings VALUES (14, 'DESERT_S', '02/06/2023', 34.00);
INSERT INTO bookings VALUES (15, 'DESERT_S', '02/09/2023', 85.00);

INSERT INTO bookings VALUES (16, 'DESERT_S', '02/12/2023', 23.00);
INSERT INTO bookings VALUES (17, 'DESERT_S', '02/13/2023', 76.00);
INSERT INTO bookings VALUES (18, 'DESERT_S', '02/14/2023', 85.00);

INSERT INTO bookings VALUES (19, 'OCEAN_WV', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (20, 'OCEAN_WV', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (21, 'OCEAN_WV', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (22, 'OCEAN_WV', '02/06/2023', 75.00);
INSERT INTO bookings VALUES (23, 'OCEAN_WV', '02/09/2023', 34.00);
INSERT INTO bookings VALUES (24, 'OCEAN_WV', '02/12/2023', 85.00);

INSERT INTO bookings VALUES (25, 'OCEAN_WV', '02/13/2023', 23.00);
INSERT INTO bookings VALUES (26, 'OCEAN_WV', '02/14/2023', 76.00);
INSERT INTO bookings VALUES (27, 'OCEAN_WV', '02/16/2023', 85.00);

INSERT INTO bookings VALUES (28, 'CITY_BLD', '02/01/2023', 98.00);
INSERT INTO bookings VALUES (29, 'CITY_BLD', '02/02/2023', 75.00);
INSERT INTO bookings VALUES (30, 'CITY_BLD', '02/04/2023', 85.00);

INSERT INTO bookings VALUES (31, 'CITY_BLD', '02/12/2023', 75.00);
INSERT INTO bookings VALUES (32, 'CITY_BLD', '02/13/2023', 34.00);
```

```
INSERT INTO bookings VALUES (33, 'CITY_BLD', '02/17/2023', 85.00);

INSERT INTO bookings VALUES (34, 'CITY_BLD', '02/22/2023', 23.00);
INSERT INTO bookings VALUES (35, 'CITY_BLD', '02/23/2023', 76.00);
INSERT INTO bookings VALUES (36, 'CITY_BLD', '02/24/2023', 85.00);
```

このサンプルクエリでは、予約レコードを集計し、週ごとの合計を示します。各週の終了日が列名になります。

```
SELECT * FROM
    (SELECT
       booking_id,
       (date_trunc('week', booking_date::date) + '5 days'::interval)::date as enddate,
       hotel_code AS "hotel code"
FROM bookings
) PIVOT (
    count(booking_id) FOR enddate IN ('2023-02-04','2023-02-11','2023-02-18')
);
```

このクエリでは次の出力が生成されます。

```
 hotel code | 2023-02-04  | 2023-02-11 | 2023-02-18
------------+-------------+------------+----------
 FOREST_L   |           3 |          2 |        1
 DESERT_S   |           4 |          3 |        2
 OCEAN_WV   |           3 |          3 |        3
 CITY_BLD   |           3 |          1 |        2
```

Amazon Redshift は、複数の列を組み合わせたクロス集計をサポートしていません。ただし、次のようなクエリを使用して、PIVOT による集計と同様に、行データを列に変換できます。ここでは、前の例と同じ予約サンプルデータを使用します。

```
SELECT
  booking_date,
  MAX(CASE WHEN hotel_code = 'FOREST_L' THEN 'forest is booked' ELSE '' END) AS
 FOREST_L,
  MAX(CASE WHEN hotel_code = 'DESERT_S' THEN 'desert is booked' ELSE '' END) AS
 DESERT_S,
  MAX(CASE WHEN hotel_code = 'OCEAN_WV' THEN 'ocean is booked' ELSE '' END)  AS
 OCEAN_WV
FROM bookings
```

```
GROUP BY booking_date
ORDER BY booking_date asc;
```

サンプルクエリを実行すると、予約されたホテルを示す短いフレーズの横に予約日が表示されます。

```
 booking_date  | forest_l         | desert_s         | ocean_wv
---------------+------------------+------------------+--------------------
 2023-02-01    | forest is booked | desert is booked |  ocean is booked
 2023-02-02    | forest is booked | desert is booked |  ocean is booked
 2023-02-04    | forest is booked | desert is booked |  ocean is booked
 2023-02-05    |                  | desert is booked |
 2023-02-06    |                  | desert is booked |
```

PIVOT の使用に関する注意事項を以下に示します。

- PIVOT を適用可能なのはテーブル、サブクエリ、および共通テーブル式 (CTE) です。JOIN 式、再帰的 CTE、PIVOT、および UNPIVOT 式のいずれにも PIVOT を適用することはできません。さらに、ネストされていない SUPER 式、ならびに Redshift Spectrum のネストされたテーブルもサポートされません。
- PIVOT では COUNT、SUM、MIN、MAX、および AVG 集計関数がサポートされています。
- PIVOT 集計式は、サポートされた集計関数に対する呼び出しである必要があります。集計の上位にある複雑な式はサポートされていません。集計引数には、PIVOT入力テーブル以外のテーブルへの参照を含めることはできません。親クエリに対する相関参照もサポートされません。集計引数にはサブクエリを含めることができます。これらは内部的に相関させることも、PIVOT 入力テーブル上で相関させることもできます。
- PIVOT IN リスト値は、列参照またはサブクエリにすることはできません。各値は、FOR列参照と型互換である必要があります。
- IN リスト値にエイリアスがない場合には、PIVOT によりデフォルトの列名が生成されます。IN 値が 'abc' や 5 などの定数の場合、デフォルトの列名自体も定数となります。いずれの複雑な式でも、列名は Amazon Redshift で標準のデフォルト名 (?column? など) になります。

UNPIVOT の例

サンプルデータを設定し、それを使用して後続の例を実行します。

```
CREATE TABLE count_by_color (quality varchar, red int, green int, blue int);
```

```
INSERT INTO count_by_color VALUES ('high', 15, 20, 7);
INSERT INTO count_by_color VALUES ('normal', 35, NULL, 40);
INSERT INTO count_by_color VALUES ('low', 10, 23, NULL);
```

red、green、blue の入力列で UNPIVOT を実行します。

```
SELECT *
FROM (SELECT red, green, blue FROM count_by_color) UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

このクエリでは次の出力が生成されます。

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
 red   |  10
 green |  20
 green |  23
 blue  |   7
 blue  |  40
```

デフォルトでは、入力列の NULL 値はスキップされ、これに対する結果行は生成されません。

次に、INCLUDE NULLSを含む UNPIVOT の例を示します。

```
SELECT *
FROM (
    SELECT red, green, blue
    FROM count_by_color
) UNPIVOT INCLUDE NULLS (
    cnt FOR color IN (red, green, blue)
);
```

その出力を次に示します。

```
 color | cnt
-------+-----
 red   |  15
 red   |  35
```

```
 red   |  10
 green |  20
 green |
 green |  23
 blue  |   7
 blue  |  40
 blue  |
```

INCLUDING NULLS パラメータが設定されている場合には、NULL入力値は結果行を生成します。

暗黙の列として quality を含む The following query shows UNPIVOT。

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red, green, blue)
);
```

このクエリでは次の出力が生成されます。

```
 quality | color | cnt
---------+-------+-----
 high    | red   |  15
 normal  | red   |  35
 low     | red   |  10
 high    | green |  20
 low     | green |  23
 high    | blue  |   7
 normal  | blue  |  40
```

UNPIVOT 定義で参照されていない入力テーブルの列は、結果テーブルに暗黙的に追加されます。この例での quality 列が、これに当てはまります。

次に、INリスト内の値のためのエイリアスを含む UNPIVOT の例を示します。

```
SELECT *
FROM count_by_color UNPIVOT (
    cnt FOR color IN (red AS r, green AS g, blue AS b)
);
```

前述のクエリでは、次の出力が生成されます。

```
 quality | color | cnt
---------+-------+-----
 high    | r     |  15
 normal  | r     |  35
 low     | r     |  10
 high    | g     |  20
 low     | g     |  23
 high    | b     |   7
 normal  | b     |  40
```

UNPIVOT 演算子は、INリストの各値に関するオプションのエイリアスを受け入れます。各エイリアスでは、それぞれの value 列にあるデータをカスタマイズできますます。

以下に、UNPIVOTの使用に関する注意事項を示します。

- UNPIVOT を適用可能なのはテーブル、サブクエリ、および共通テーブル式 (CTE) です。JOIN 式、再帰的 CTE、PIVOT、および UNPIVOT 式のいずれにも UNPIVOT を適用することはできません。さらに、ネストされていない SUPER 式、ならびに Redshift Spectrum のネストされたテーブルもサポートされません。
- UNPIVOT IN リストには、入力テーブルの列への参照のみを含める必要があります。IN リスト列には、その全体との互換性がある共通の型が必要です。UNPIVOT 値列には、下記の共通の型があります。UNPIVOT 名前列の型は VARCHAR です。
- IN リストの値がエイリアスを持たない場合、UNPIVOTでは、デフォルト値として列名が使用されます。

JOIN 句の例

SQL JOIN 句は、共通のフィールドに基づいて 2 つ以上のテーブルのデータを結合するために使用されます。結果は、指定した結合方法によって変わる場合もあります。JOIN 句の詳細については、「パラメータ」を参照してください。

次の例では、TICKITサンプルデータのデータを使用します。データベーススキーマの詳細については、「サンプルデータベース」を参照してください。サンプルデータをロードする方法については、「Amazon Redshift 入門ガイド」の「データのロード」を参照してください。

次のクエリは、LISTING テーブルと SALES テーブル間の内部結合です (JOIN キーワードを除く)。ここで、LISTING テーブルの LISTID は 1～5 です。このクエリは、LISTING テーブル (左のテーブル) と SALES テーブル (右のテーブル) の LISTID 列の値と一致します。結果は、LISTID 1、4、および 5 が基準に一致することを示しています。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing, sales
where listing.listid = sales.listid
and listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

次のクエリは左外部結合です。左と右の外部結合は、もう一方のテーブルに一致するものが見つからない場合に、結合したどちらかのテーブルの値を保持します。左のテーブルと右のテーブルは、構文に一覧表示されている最初と 2 番目のテーブルです。NULL 値は、結果セットの「ギャップ」を埋めるために使われます。このクエリは、LISTING テーブル (左のテーブル) と SALES テーブル (右のテーブル) の LISTID 列の値と一致します。結果は、LISTID 2 および 3 にセールスがないことを示しています。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing left outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

次のクエリは右外部結合です。このクエリは、LISTING テーブル (左のテーブル) と SALES テーブル (右のテーブル) の LISTID 列の値と一致します。結果は、LISTID 1、4、および 5 が基準に一致することを示しています。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing right outer join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
```

```
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

次のクエリは完全結合です。完全結合は、もう一方のテーブルに一致するものが見つからない場合に、結合したテーブルの値を保持します。左のテーブルと右のテーブルは、構文に一覧表示されている最初と 2 番目のテーブルです。NULL 値は、結果セットの「ギャップ」を埋めるために使われます。このクエリは、LISTING テーブル (左のテーブル) と SALES テーブル (右のテーブル) の LISTID 列の値と一致します。結果は、LISTID 2 および 3 にセールスがないことを示しています。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     2 | NULL   | NULL
     3 | NULL   | NULL
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

次のクエリは完全結合です。このクエリは、LISTING テーブル (左のテーブル) と SALES テーブル (右のテーブル) の LISTID 列の値と一致します。セールスがない行 (LISTID 2 および 3) のみが結果に表示されます。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from listing full join sales on sales.listid = listing.listid
where listing.listid between 1 and 5
and (listing.listid IS NULL or sales.listid IS NULL)
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
```

```
      2 | NULL    | NULL
      3 | NULL    | NULL
```

次の例は、ON 句を含む内部結合です。この場合、NULL 行は返されません。

```
select listing.listid, sum(pricepaid) as price, sum(commission) as comm
from sales join listing
on sales.listid=listing.listid and sales.eventid=listing.eventid
where listing.listid between 1 and 5
group by 1
order by 1;

listid | price  |  comm
-------+--------+--------
     1 | 728.00 | 109.20
     4 |  76.00 |  11.40
     5 | 525.00 |  78.75
```

次のクエリは、LISTING テーブルと SALES テーブルのクロス結合またはデカルト結合で、結果を制限する述語が使用されています。このクエリは、両方のテーブルの LISTID 1、2、3、4、および 5 について、SALES テーブルと LISTING テーブルの LISTID 列の値と一致します。結果は、20 行が基準に一致することを示しています。

```
select sales.listid as sales_listid, listing.listid as listing_listid
from sales cross join listing
where sales.listid between 1 and 5
and listing.listid between 1 and 5
order by 1,2;

sales_listid | listing_listid
-------------+---------------
1            | 1
1            | 2
1            | 3
1            | 4
1            | 5
4            | 1
4            | 2
4            | 3
4            | 4
4            | 5
5            | 1
```

```
5             | 1
5             | 2
5             | 2
5             | 3
5             | 3
5             | 4
5             | 4
5             | 5
5             | 5
```

次の例は、2 つのテーブル間の自然結合です。この場合、listid、sellerid、eventid、および dateid の各列は、両方のテーブルで同じ名前とデータ型を持つため、結合列として使用されます。結果は 5 行に制限されます。

```
select listid, sellerid, eventid, dateid, numtickets
from listing natural join sales
order by 1
limit 5;

listid | sellerid  | eventid | dateid | numtickets
-------+-----------+---------+--------+-----------
113    | 29704     | 4699    | 2075   | 22
115    | 39115     | 3513    | 2062   | 14
116    | 43314     | 8675    | 1910   | 28
118    | 6079      | 1611    | 1862   | 9
163    | 24880     | 8253    | 1888   | 14
```

次の例は、USING 句が含まれている 2 つのテーブル間の結合です。この場合、listid と eventid の各列が結合列として使用されます。結果は 5 行に制限されます。

```
select listid, listing.sellerid, eventid, listing.dateid, numtickets
from listing join sales
using (listid, eventid)
order by 1
limit 5;

listid | sellerid | eventid | dateid | numtickets
-------+----------+---------+--------+-----------
1      | 36861    | 7872    | 1850   | 10
4      | 8117     | 4337    | 1970   | 8
5      | 1616     | 8647    | 1963   | 4
5      | 1616     | 8647    | 1963   | 4
```

```
6        | 47402     | 8240     | 2053    | 18
```

次のクエリは、FROM 句の 2 つのサブクエリを内部結合したものです。このクエリは、イベント (コンサートとショー) の異なるカテゴリのチケットの販売数と売れ残り数を検出します。この FROM 句サブクエリはテーブルサブクエリです。これらは、複数の列と行を返すことができます。

```
select catgroup1, sold, unsold
from
(select catgroup, sum(qtysold) as sold
from category c, event e, sales s
where c.catid = e.catid and e.eventid = s.eventid
group by catgroup) as a(catgroup1, sold)
join
(select catgroup, sum(numtickets)-sum(qtysold) as unsold
from category c, event e, sales s, listing l
where c.catid = e.catid and e.eventid = s.eventid
and s.listid = l.listid
group by catgroup) as b(catgroup2, unsold)

on a.catgroup1 = b.catgroup2
order by 1;

catgroup1 |  sold  | unsold
----------+--------+--------
Concerts  | 195444 |1067199
Shows     | 149905 | 817736
```

WHERE 句

WHERE 句には、テーブルの結合またはテーブル内の列への述語の適用のいずれかを実行する条件が含まれています。テーブルは、WHERE 句または FROM 句のいずれかの適切な構文を使うことで結合できます。外部結合基準は、FROM 句で指定する必要があります。

構文

```
[ WHERE condition ]
```

condition

結合条件やテーブル列に関する述語など、ブール値結果に関する検索条件 次の例は有効な join の条件です。

```
sales.listid=listing.listid
sales.listid<>listing.listid
```

次の例は、テーブル内の列の有効な条件です。

```
catgroup like 'S%'
venueseats between 20000 and 50000
eventname in('Jersey Boys','Spamalot')
year=2008
length(catdesc)>25
date_part(month, caldate)=6
```

条件は単純にすることも複雑することもできます。複雑な条件の場合、かっこを使って、論理ユニットを分離します。次の例では、結合条件をかっこによって囲みます。

```
where (category.catid=event.catid) and category.catid in(6,7,8)
```

使用に関する注意事項

WHERE 句でエイリアスを使って、SELECT リスト式を参照することができます。

WHERE 句内の集計関数の結果を制限することはできません。その目的には、HAVING 句を使用してください。

WHERE 句内で制限されている列は、FROM 句内のテーブル参照から生成する必要があります。

例

次のクエリは、SALES テーブルと EVENT テーブルの結合条件、EVENTNAME 列に関する述語、STARTTIME 列に関する 2 つの述語など、複数の WHERE 句制限の組み合わせを使用します。

```
select eventname, starttime, pricepaid/qtysold as costperticket, qtysold
from sales, event
where sales.eventid = event.eventid
and eventname='Hannah Montana'
and date_part(quarter, starttime) in(1,2)
and date_part(year, starttime) = 2008
order by 3 desc, 4, 2, 1 limit 10;

eventname    |      starttime      |   costperticket   | qtysold
```

```
-----------------+---------------------+-------------------+---------
Hannah Montana | 2008-06-07 14:00:00 |     1706.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |     1658.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1479.00000000 |       3
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       1
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       2
Hannah Montana | 2008-06-07 14:00:00 |     1163.00000000 |       4
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       1
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       2
Hannah Montana | 2008-05-01 19:00:00 |      497.00000000 |       4
(10 rows)
```

WHERE 句の Oracle スタイルの外部結合

Oracle との互換性を目的として、Amazon Redshift は WHERE 句結合条件での Oracle の外部結合演算子 (+) をサポートします。この演算子は、外部結合条件の定義でのみ使用することを意図しています。他のコンテキストで使用しないでください。この演算子をその他の目的に使用すると、ほとんどの場合、メッセージを表示せずに無視します。

外部結合は、同等の内部結合が返す行と同じ行をすべて返します。それに加え、1 つまたは両方のテーブルから一致しない行も返します。FROM 句では、left、right、full の外部結合を指定できます。WHERE 句では、left または right の外部結合だけを指定できます。

TABLE1 と TABLE2 を外部結合し、TABLE1 (左側の外部結合) から一致しない行を返すには、FROM 句内で `TABLE1 LEFT OUTER JOIN TABLE2` を指定するか、WHERE 句内で TABLE2 からのすべての結合列に (+) 演算子を適用します。TABLE2 と一致する行がない TABLE1 のすべての行の場合、TABLE2 からの列を含む SELECT リスト式に対するクエリ結果には Null が含まれます。

TABLE1 と一致する行がない TABLE2 のすべての行に対して同じ動作を生成するには、FROM 句で `TABLE1 RIGHT OUTER JOIN TABLE2` を指定するか、WHERE 句内で TABLE1 からのすべての結合列に (+) 演算子を適用します。

基本構文

```
[ WHERE {
[ table1.column1 = table2.column1(+) ]
[ table1.column1(+) = table2.column1 ]
}
```

最初の条件は、以下と同等です。

```
from table1 left outer join table2
on table1.column1=table2.column1
```

2 番目の条件は、以下と同等です。

```
from table1 right outer join table2
on table1.column1=table2.column1
```

> **Note**
>
> ここに示す構文は、結合列の 1 ペアを介した等価結合のシンプルなケースを示しています。ただし、他のタイプの比較条件と結合列の複数のペアも有効です。

例えば、次の WHERE 句は、列の 2 つのペアを介して、外部結合を定義します。(+) 演算子は、両方の条件の同じテーブルにアタッチする必要があります。

```
where table1.col1 > table2.col1(+)
and table1.col2 = table2.col2(+)
```

使用に関する注意事項

可能な場合は、WHERE 句の (+) 演算子の代わりに、標準の FROM 句 Outer JOIN 構文を使用してください。(+) 演算子を含んでいるクエリは、次の規則に依存します。

- (+) 演算子は WHERE 句内と、テーブルまたはビューから列への参照でのみ使用できます。
- 式に (+) 演算子を適用することはできません。ただし、式に (+) 演算子を使用する列を含めることはできます。例えば、次の結合条件は、構文エラーを返します。

  ```
  event.eventid*10(+)=category.catid
  ```

  ただし、次の結合条件は有効です。

  ```
  event.eventid(+)*10=category.catid
  ```

- FROM 句結合構文も含むクエリブロック内で (+) 演算子を使用することはできません。

- 2 つのテーブルを複数の結合条件を介して結合する場合、(+) 演算子をこれらの条件のすべてで使用するか、すべてで使用しないことが必要です。構文スタイルが混在する結合は、内部結合として実行され、警告は出力されません。
- 外部クエリ内のテーブルを内部クエリによって生成されたテーブルに結合する場合、(+) 演算子は外部結合を生成しません。
- (+) 演算子を使って、テーブルを自分自身に外部結合するには、FROM 句内でテーブルエイリアスを定義し、結合条件でそのエイリアスを参照する必要があります。

```
select count(*)
from event a, event b
where a.eventid(+)=b.catid;

count
-------
8798
(1 row)
```

- (+) 演算子を含む結合条件を、OR 条件または IN 条件と組み合わせることはできません。次に例を示します。

```
select count(*) from sales, listing
where sales.listid(+)=listing.listid or sales.salesid=0;
ERROR:  Outer join operator (+) not allowed in operand of OR or IN.
```

- 3 つ以上のテーブルを外部結合する WHERE 句では、(+) 演算子は指定されたテーブルに一度して適用できません。次の例では、2 つの連続する結合で (+) 演算子を使って SALES テーブルを参照することはできません。

```
select count(*) from sales, listing, event
where sales.listid(+)=listing.listid and sales.dateid(+)=date.dateid;
ERROR:  A table may be outer joined to at most one other table.
```

- WHERE 句の外部結合条件で TABLE2 からの列を定数と比較する場合、(+) 演算子を列に適用します。演算子を含めない場合、TABLE1 から外部結合された行 (制限された行に対して null を含んでいる行) は削除されます。以下の「例」のセクションを参照してください。

例

次の結合クエリは、LISTID 列を介した SALES テーブルと LISTING テーブルの左外部結合を指定します。

```
select count(*)
from sales, listing
where sales.listid = listing.listid(+);

count
--------
172456
(1 row)
```

次の同等クエリは同じ結果を提供しますが、FROM 句結合構文を使用します。

```
select count(*)
from sales left outer join listing on sales.listid = listing.listid;

count
--------
172456
(1 row)
```

すべての一覧が販売に含まれるわけではないため、SALES テーブルに LISTING テーブル内のすべてのリストのレコードが含まれているわけではありません。次のクエリでは、SALES と LISTING を外部結合し、SALES テーブルが指定されたリスト ID に対して販売がないことをレポートした場合でも、LISTING からの行を返します。SALES テーブルから生成された PRICE と COMM 列には、一致しない行の結果セットにヌルが格納されています。

```
select listing.listid, sum(pricepaid) as price,
sum(commission) as comm
from listing, sales
where sales.listid(+) = listing.listid and listing.listid between 1 and 5
group by 1 order by 1;

listid | price  |  comm
--------+--------+--------
1 | 728.00 | 109.20
2 |        |
3 |        |
4 |  76.00 |  11.40
5 | 525.00 |  78.75
(5 rows)
```

WHERE 句の結合演算子を使用する場合、FROM 句のテーブルの順番は問題になりません。

WHERE 句のより複雑な外部結合条件の例としては、2 つのテーブル列間の比較と定数との比較による条件のケースがあります。

```
where category.catid=event.catid(+) and eventid(+)=796;
```

(+) 演算子は 2 つの場所で使用されることに注意してください。1 つ目はテーブル間の等価比較、2 つ目は EVENTID 列への条件の比較においてです。この構文の結果は、EVENTID に関する制限が評価される際、外部結合された行に保存されます。EVENTID 制限から (+) 演算子を削除すると、クエリはこの制限を、外部結合条件ではなく、フィルタとして処理します。この場合、EVENTID にヌルが格納されている外部結合された行は、結果セットから削除されます。

ここに、この動作を表現する完全なクエリを示します。

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid(+)=796;

catname | catgroup | eventid
-----------+----------+---------
Classical | Concerts |
Jazz | Concerts |
MLB | Sports   |
MLS | Sports   |
Musicals | Shows    | 796
NBA | Sports   |
NFL | Sports   |
NHL | Sports   |
Opera | Shows    |
Plays | Shows    |
Pop | Concerts |
(11 rows)
```

FROM 句構文を使用した同等クエリは次のとおりです。

```
select catname, catgroup, eventid
from category left join event
on category.catid=event.catid and eventid=796;
```

このクエリの WHERE 句バージョンから 2 番目の (+) 演算子を削除した場合、1 行だけが返されます (eventid=796 となる行)。

```
select catname, catgroup, eventid
from category, event
where category.catid=event.catid(+) and eventid=796;

catname | catgroup | eventid
-----------+----------+---------
Musicals | Shows    | 796
(1 row)
```

GROUP BY 句

GROUP BY 句は、クエリのグループ化列を特定します。クエリが SUM、AVG、COUNT などの標準関数を使って集計する場合、グループ化列を宣言する必要があります。詳細については、「集計関数」を参照してください。

構文

```
GROUP BY group_by_clause [, ...]

group_by_clause := {
    expr |
    GROUPING SETS ( () | group_by_clause [, ...] ) |
    ROLLUP ( expr [, ...] ) |
    CUBE ( expr [, ...] )
    }
```

パラメータ

expr

列または式のリストは、クエリの SELECT リストの非集計式のリストと一致する必要があります。例えば、次のシンプルなクエリを考慮してみます。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by listid, eventid
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
```

```
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

このクエリでは、選択されたリストは 2 つの集計式で構成されています。最初の式は SUM 関数を使用し、2 番目の式は COUNT 関数を使用します。残りの 2 つの例 (LISTID と EVENTID) は、グループ化列として宣言する必要があります。

GROUP BY 句の式は、序数を使用することで、SELECT リストを参照することもできます。例えば、前の例は、次のように短縮できます。

```
select listid, eventid, sum(pricepaid) as revenue,
count(qtysold) as numtix
from sales
group by 1,2
order by 3, 4, 2, 1
limit 5;

listid | eventid | revenue | numtix
-------+---------+---------+--------
89397  |      47 |   20.00 |      1
106590 |      76 |   20.00 |      1
124683 |     393 |   20.00 |      1
103037 |     403 |   20.00 |      1
147685 |     429 |   20.00 |      1
(5 rows)
```

GROUPING SETS/ROLLUP/CUBE

集計拡張機能 GROUPING SETS、ROLLUP、CUBE を使用すると、1 つのステートメントで複数の GROUP BY オペレーションを実行できます。集計拡張機能および関連する関数の詳細については、「集計拡張機能」を参照してください。

集計拡張機能

Amazon Redshift では、1 つのステートメントで複数の GROUP BY オペレーションの処理を実行するための集計拡張機能をサポートしています。

集計拡張機能の例では、ある電子機器会社の売上データを格納する orders テーブルを使用しています。orders は以下を使用して作成できます。

```
CREATE TABLE ORDERS (
    ID INT,
    PRODUCT CHAR(20),
    CATEGORY CHAR(20),
    PRE_OWNED CHAR(1),
    COST DECIMAL
);

INSERT INTO ORDERS VALUES
    (0, 'laptop',       'computers',    'T', 1000),
    (1, 'smartphone',   'cellphones',   'T', 800),
    (2, 'smartphone',   'cellphones',   'T', 810),
    (3, 'laptop',       'computers',    'F', 1050),
    (4, 'mouse',        'computers',    'F', 50);
```

GROUPING SETS

1 つのステートメントで 1 つ以上のグループ化セットを計算します。グループ化セットとは、1 つの GROUP BY 句のセットで、クエリの結果セットをグループ化できる 0 個以上の列のセットです。GROUP BY GROUPING SETS は、異なる列でグループ化された 1 つの結果セットに対して UNION ALL クエリを実行することに相当します。例えば、GROUP BY GROUPING SETS((a), (b)) は、GROUP BY a UNION ALL GROUP BY b と同等です。

次の例では、製品のカテゴリと販売された製品の種類の両方に従ってグループ化された注文テーブルの製品のコストを返します。

```

SELECT category, product, sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(category, product);

       category       |       product        | total
----------------------+----------------------+-------
 computers            |                      |  2100
 cellphones           |                      |  1610
                      | laptop               |  2050
                      | smartphone           |  1610
                      | mouse                |    50
```

```
(5 rows)
```

ROLLUP

前の列が後続の列の親と見なされる階層を前提としています。ROLLUP は、指定された列ごとにデータをグループ化し、グループ化された行に加えて、グループ化列の全レベルの合計を表す追加の小計行を返します。例えば、GROUP BY ROLLUP((a), (b)) を使用すると、b が a のサブセクションであると仮定して、最初に a でグループ化された結果セットを返し、次に b でグループ化された結果セットを返すことができます。また、ROLLUP では、列をグループ化せずに結果セット全体を含む行を返します。

GROUP BY ROLLUP((a), (b)) は、GROUP BY GROUPING SETS((a,b), (a), ()) と同等です。

次の例では、最初にカテゴリ別にグループ化された注文テーブルの製品のコストを返し、次にカテゴリが細分化された製品を返します。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY ROLLUP(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      |                      |  3710
(6 rows)
```

CUBE

指定した列ごとにデータをグループ化し、グループ化された行に加えて、グループ化列の全レベルの合計を表す追加の小計行を返します。CUBE は ROLLUP と同じ行を返しますが、ROLLUP の対象とならないグループ列のすべての組み合わせで小計行を追加します。例えば、GROUP BY CUBE ((a), (b)) を使用すると、b が a のサブセクションであると仮定して、最初に a でグループ化された結果セットを返し、次に b でグループ化された結果セット、さらに b のみでグループ化された結果セットを返すことができます。また、CUBE では、列をグループ化せずに結果セット全体を含む行を返します。

GROUP BY CUBE((a), (b)) は GROUP BY GROUPING SETS((a, b), (a), (b), ()) と同等です。

次の例では、最初にカテゴリ別にグループ化された注文テーブルの製品のコストを返し、次にカテゴリが細分化された製品を返します。前述の ROLLUP の例とは異なり、このステートメントはグループ化列のすべての組み合わせの結果を返します。

```
SELECT category, product, sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 1,2;

       category       |       product        | total
----------------------+----------------------+-------
 cellphones           | smartphone           |  1610
 cellphones           |                      |  1610
 computers            | laptop               |  2050
 computers            | mouse                |    50
 computers            |                      |  2100
                      | laptop               |  2050
                      | mouse                |    50
                      | smartphone           |  1610
                      |                      |  3710
(9 rows)
```

GROUPING/GROUPING_ID functions

ROLLUP と CUBE は、小計行を示す NULL 値を結果セットに追加します。例えば、GROUP BY ROLLUP((a), (b)) は、b グループ列の値が NULL である行を 1 つ以上返し、それらが a グループ列のフィールドの小計であることを示します。これらの NULL 値は、返されるタプルの形式を満たすためにのみ機能します。

それ自体に NULL 値が格納されているリレーションに対して ROLLUP と CUBE を使用して GROUP BY オペレーションを実行すると、同じグループ化列を持つように見える行を含む結果セットが生成されることがあります。前の例に戻ると、b のグループ化列に NULL 値が格納されている場合、GROUP BY ROLLUP((a), (b)) は、小計ではない b のグループ化列に値が NULL の行を返します。

ROLLUP と CUBE によって作成された NULL 値と、テーブル自体に格納されている NULL 値を区別するには、GROUPING 関数またはそのエイリアスである GROUPING_ID を使用できます。GROUPING は、引数として 1 つのグループ化セットを取り、結果セットの各行について、その位置のグループ化列に対応する 0 または 1 ビットの値を返し、その値を整数に変換します。その位置の値が集計拡張によって作成された NULL 値の場合、GROUPING は 1 を返します。格納されている NULL 値を含む他のすべての値に対しては 0 を返します。

例えば、GROUPING(category, product) は、その行のグループ化列の値に応じて、特定の行について次の値を返すことができます。この例では、テーブル内のすべての NULL 値は、集計拡張によって作成された NULL 値です。

カテゴリ列	製品列	GROUPING 関数のビット値	10 進値
null でない	null でない	00	0
null でない	NULL	01	1
NULL	null でない	10	2
NULL	NULL	11	3

GROUPING 関数は、クエリの SELECT リスト部分に次の形式で表示されます。

```
SELECT ... [GROUPING( expr )...] ...
  GROUP BY ... {CUBE | ROLLUP| GROUPING SETS} ( expr ) ...
```

次の例は、前述の CUBE の例と同じですが、グループ化セットに GROUPING 関数が追加されています。

```
SELECT category, product,
       GROUPING(category) as grouping0,
       GROUPING(product) as grouping1,
       GROUPING(category, product) as grouping2,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product) ORDER BY 3,1,2;

       category       |       product        | grouping0 | grouping1 | grouping2 |
 total
----------------------+----------------------+-----------+-----------+-----------
+-------
 cellphones           | smartphone           |         0 |         0 |         0 |
 1610
 cellphones           |                      |         0 |         1 |         1 |
 1610
```

```
 computers              | laptop                |         0 |          0 |          0 |
 2050
 computers              | mouse                 |         0 |          0 |          0 |
 50
 computers              |                       |         0 |          1 |          1 |
 2100
                        | laptop                |         1 |          0 |          2 |
 2050
                        | mouse                 |         1 |          0 |          2 |
 50
                        | smartphone            |         1 |          0 |          2 |
 1610
                        |                       |         1 |          1 |          3 |
 3710
(9 rows)
```

部分的な ROLLUP および CUBE

ROLLUP および CUBE オペレーションは、小計の一部のみで実行できます。

部分的な ROLLUP および CUBE オペレーションの構文は次のとおりです。

```
GROUP BY expr1, { ROLLUP | CUBE }(expr2, [, ...])
```

ここで、GROUP BY 句は expr2 以降のレベルの小計行のみを作成します。

次の例は、注文テーブルでの ROLLUP および CUBE オペレーションの一部を示しています。最初に製品が中古品かどうかでグループ化し、次にカテゴリ列と製品列で ROLLUP と CUBE を実行します。

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, ROLLUP(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
```

```
 F          | computers            |                      |        2 |  1100
 T          | cellphones           |                      |        2 |  1610
 T          | computers            |                      |        2 |  1000
 F          |                      |                      |        6 |  1100
 T          |                      |                      |        6 |  2610
(9 rows)

SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY pre_owned, CUBE(category, product) ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F          | computers            | laptop               |        0 |  1050
 F          | computers            | mouse                |        0 |    50
 T          | cellphones           | smartphone           |        0 |  1610
 T          | computers            | laptop               |        0 |  1000
 F          | computers            |                      |        2 |  1100
 T          | cellphones           |                      |        2 |  1610
 T          | computers            |                      |        2 |  1000
 F          |                      | laptop               |        4 |  1050
 F          |                      | mouse                |        4 |    50
 T          |                      | laptop               |        4 |  1000
 T          |                      | smartphone           |        4 |  1610
 F          |                      |                      |        6 |  1100
 T          |                      |                      |        6 |  2610
(13 rows)
```

「pre_owned」列は ROLLUP および CUBE オペレーションには含まれないため、他のすべての行を含む総計行はありません。

連結グループ

複数の GROUPING SETS/ROLLUP/CUBE 句を連結して、さまざまなレベルの小計を計算できます。連結グループでは、指定されたグループ化セットの直積集合を返します。

GROUPING SETS/ROLLUP/CUBE 句を連結する構文は次のとおりです。

```
GROUP BY {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...]),
         {ROLLUP|CUBE|GROUPING SETS}(expr1[, ...])[, ...]
```

次の例を考慮して、小さな連結グループがどのようにして大きな最終結果セットを生成できるかを確認してください。

```
SELECT pre_owned, category, product,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY CUBE(category, product), GROUPING SETS(pre_owned, ())
ORDER BY 4,1,2,3;

 pre_owned |       category       |       product        | group_id | total
-----------+----------------------+----------------------+----------+-------
 F         | computers            | laptop               |        0 |  1050
 F         | computers            | mouse                |        0 |    50
 T         | cellphones           | smartphone           |        0 |  1610
 T         | computers            | laptop               |        0 |  1000
           | cellphones           | smartphone           |        1 |  1610
           | computers            | laptop               |        1 |  2050
           | computers            | mouse                |        1 |    50
 F         | computers            |                      |        2 |  1100
 T         | cellphones           |                      |        2 |  1610
 T         | computers            |                      |        2 |  1000
           | cellphones           |                      |        3 |  1610
           | computers            |                      |        3 |  2100
 F         |                      | laptop               |        4 |  1050
 F         |                      | mouse                |        4 |    50
 T         |                      | laptop               |        4 |  1000
 T         |                      | smartphone           |        4 |  1610
           |                      | laptop               |        5 |  2050
           |                      | mouse                |        5 |    50
           |                      | smartphone           |        5 |  1610
 F         |                      |                      |        6 |  1100
 T         |                      |                      |        6 |  2610
           |                      |                      |        7 |  3710
(22 rows)
```

ネストされたグループ

GROUPING SETS/ROLLUP/CUBE オペレーションを GROUPING SETS expr として使用して、ネストされたグループを作成できます。ネストされた GROUPING SETS 内のサブグループはフラット化されます。

ネストされたグループの構文は次のとおりです。

```
GROUP BY GROUPING SETS({ROLLUP|CUBE|GROUPING SETS}(expr[, ...])[, ...])
```

次の例を考えます。

```
SELECT category, product, pre_owned,
       GROUPING(category, product, pre_owned) as group_id,
       sum(cost) as total
FROM orders
GROUP BY GROUPING SETS(ROLLUP(category), CUBE(product, pre_owned))
ORDER BY 4,1,2,3;

       category       |       product        | pre_owned | group_id | total
----------------------+----------------------+-----------+----------+-------
 cellphones           |                      |           |        3 |  1610
 computers            |                      |           |        3 |  2100
                      | laptop               | F         |        4 |  1050
                      | laptop               | T         |        4 |  1000
                      | mouse                | F         |        4 |    50
                      | smartphone           | T         |        4 |  1610
                      | laptop               |           |        5 |  2050
                      | mouse                |           |        5 |    50
                      | smartphone           |           |        5 |  1610
                      |                      | F         |        6 |  1100
                      |                      | T         |        6 |  2610
                      |                      |           |        7 |  3710
                      |                      |           |        7 |  3710
(13 rows)
```

ROLLUP(category) と CUBE(product, pre_owned) の両方にグループ化セット () が含まれているため、総計を表す行が重複していることに注意してください。

使用に関する注意事項

- GROUP BY 句は最大 64 のグループ化セットをサポートします。ROLLUP と CUBE、または GROUPING SETS、ROLLUP、および CUBE の組み合わせの場合、この制限はグループ化セットの暗黙の数に適用されます。例えば、GROUP BY CUBE((a), (b)) は、2 つではなく 4 つのグループ化セットとしてカウントされます。
- 集計拡張機能を使用する場合、定数をグループ化列として使用することはできません。
- 重複する列を含むグループ化セットを作成することはできません。

HAVING 句

HAVING 句は、クエリが返す中間グループ結果セットに条件を適用します。

構文

```
[ HAVING condition ]
```

例えば、SUM 関数の結果を制限できます。

```
having sum(pricepaid) >10000
```

HAVING 条件は、すべての WHERE 句条件が適用され、GROUP BY オペレーションが完了してから適用されます。

条件自体は、WHERE 句の条件と同じ形式になります。

使用に関する注意事項

- HAVING 句条件内で参照される列は、グループ化列または集計関数の結果を参照する列のいずれかでなければなりません。
- HAVING 句では、以下の項目を指定することはできません。
  - SELECT リスト項目を参照する序数。序数が使用できるのは、GROUP BY 句または ORDER BY 句だけです。

例

次のクエリは、すべてのイベントに対するチケットの合計販売を名前別に計算し、販売合計が 800,000 ドルに達しなかったイベントを削除します。HAVING 条件は、SELECT リスト内の集計関数の結果に適用されます。`sum(pricepaid)`)

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(pricepaid) > 800000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
```

```
Mamma Mia!        | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

次のクエリは、同じような結果セットを計算します。ただしこの場合、SELECT リスト sum(qtysold) で指定されていない集計に対して HAVING 条件が適用されます。2,000 枚を超えるチケットを販売しなかったイベントは、最終結果から削除されます。

```
select eventname, sum(pricepaid)
from sales join event on sales.eventid = event.eventid
group by 1
having sum(qtysold) >2000
order by 2 desc, 1;

eventname        |    sum
-----------------+-----------
Mamma Mia!        | 1135454.00
Spring Awakening |  972855.00
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
Chicago          |  790993.00
Spamalot         |  714307.00
```

次のクエリは、すべてのイベントに対するチケットの合計販売を名前別に計算し、販売合計が 800,000 ドルに達しなかったイベントを削除します。HAVING 条件は、sum(pricepaid) の pp エイリアスを使用して SELECT リスト内の集計関数の結果に適用されます。

```
select eventname, sum(pricepaid) as pp
from sales join event on sales.eventid = event.eventid
group by 1
having pp > 800000
order by 2 desc, 1;

eventname        |    pp
-----------------+-----------
Mamma Mia!        | 1135454.00
Spring Awakening |  972855.00
```

```
The Country Girl |  910563.00
Macbeth          |  862580.00
Jersey Boys      |  811877.00
Legally Blonde   |  804583.00
```

QUALIFY 句

QUALIFY 句は、ユーザーが指定した検索条件に従って、以前に計算されたウィンドウ関数の結果をフィルタリングします。この句を使用すると、サブクエリを使用せずに、ウィンドウ関数の結果にフィルタリング条件を適用できます。

これは、WHERE 句から行をさらに絞り込む条件を適用する HAVING 句に似ています。QUALIFY と HAVING の違いは、QUALIFY 句からフィルタリングされた結果は、データに対してウィンドウ関数を実行した結果に基づいている可能性があることです。1 つのクエリで QUALIFY 句と HAVING 句の両方を使用できます。

構文

```
QUALIFY condition
```

> **Note**
>
> FROM 句の直後に QUALIFY 句を使用する場合、FROM リレーション名には QUALIFY 句の前にエイリアスを指定する必要があります。

例

このセクションの例では下のサンプルデータを使用します。

```
create table store_sales (ss_sold_date date, ss_sold_time time,
               ss_item text, ss_sales_price float);
insert into store_sales values ('2022-01-01', '09:00:00', 'Product 1', 100.0),
                               ('2022-01-01', '11:00:00', 'Product 2', 500.0),
                               ('2022-01-01', '15:00:00', 'Product 3', 20.0),
                               ('2022-01-01', '17:00:00', 'Product 4', 1000.0),
                               ('2022-01-01', '18:00:00', 'Product 5', 30.0),
                               ('2022-01-02', '10:00:00', 'Product 6', 5000.0),
                               ('2022-01-02', '16:00:00', 'Product 7', 5.0);
```

次の例は、その日の 12:00 以降に販売される最も高価な 2 つの商品を検索する方法を示しています。

```
SELECT *
FROM store_sales ss
WHERE ss_sold_time > time '12:00:00'
QUALIFY row_number()
OVER (PARTITION BY ss_sold_date ORDER BY ss_sales_price DESC) <= 2


 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 17:00:00     | Product 4 |           1000
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

次に、毎日最後に販売された商品を検索できます。

```
SELECT *
FROM store_sales ss
QUALIFY last_value(ss_item)
OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
      ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) = ss_item;

ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price
--------------+--------------+-----------+----------------
 2022-01-01   | 18:00:00     | Product 5 |             30
 2022-01-02   | 16:00:00     | Product 7 |              5
```

次の例では、前のクエリ (毎日最後に販売された商品) と同じ記録が返されますが、QUALIFY 句は使用しません。

```
SELECT * FROM (
  SELECT *,
  last_value(ss_item)
  OVER (PARTITION BY ss_sold_date ORDER BY ss_sold_time ASC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) ss_last_item
  FROM store_sales ss
)
WHERE ss_last_item = ss_item;

 ss_sold_date | ss_sold_time |  ss_item  | ss_sales_price | ss_last_item
```

```
--------------+--------------+-----------+----------------+--------------
 2022-01-02    | 16:00:00      | Product 7 |               5 | Product 7
 2022-01-01    | 18:00:00      | Product 5 |              30 | Product 5
```

UNION、INTERSECT、または EXCEPT

トピック

- 構文
- パラメータ
- セット演算の評価の順番
- 使用に関する注意事項
- UNION クエリの例
- UNION ALL クエリの例
- INTERSECT クエリの例
- EXCEPT クエリの例

UNION、INTERSECT、または EXCEPT セット演算子は、2 つの個別のクエリ式の比較とマージに使われます。例えば、ウェブサイトで購入者と販売者の両方を兼ねているが、ユーザー名が別々の列または表に格納されているユーザーを確認するには、これら 2 種類のユーザーの積集合を求めます。購入者ではあるが販売者ではないウェブユーザーを確認するには、EXCEPT 演算子を使用すると、2 つユーザーリストの差を見つけることができます。役割とは無関係に、すべてのユーザーのリストを作成する場合、UNION 演算子を使用できます。

構文

```
query
{ UNION [ ALL ] | INTERSECT | EXCEPT | MINUS }
query
```

パラメータ

query

UNION、INTERSECT、または EXCEPT 演算子の後に続く 2 番目のクエリ式に、SELECT リストの形式で対応するクエリ式。2 つの式は、互換性のあるデータ型の出力列を同数含んでいる必

要があります。そうでない場合、2 つの結果セットの比較とマージはできません。データ型の複数のカテゴリ間で黙示的な変換を許可しない演算を設定します。詳細については「型の互換性と変換」を参照してください。

クエリ式の数を上限なしに含むクエリを構築して、そのクエリを任意の組み合わせで UNION、INTERSECT、および EXCEPT 演算子に関連付けることができます。例えば、テーブル T1、T2、および T3 に互換性のある列セットが含まれていると想定した場合、次のクエリ構造は有効です。

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

UNION

行が片方の式から生成されたか、両方の式から生成されたかにかかわらず、2 つのクエリ式からの行を返す演算を設定します。

INTERSECT

2 つのクエリ式から生成される行を返す演算を設定します。両方の式によって返されない行は破棄されます。

EXCEPT | MINUS

2 つのクエリ式の一方から生成される行を返す演算を設定します。結果を制限するため、行は最初の結果テーブルに存在し、2 番目のテーブルには存在しない必要があります。MINUS と EXCEPT はまったく同じ意味です。

ALL

ALL キーワードは、UNION が生成する重複行を保持します。ALL キーワードを使用しないよう場合のデフォルトの動作は、このような重複を破棄します。INTERSECT ALL、EXCEPT ALL、および MINUS ALL はサポートされません。

セット演算の評価の順番

UNION および EXCEPT セット演算子は左結合です。優先順位の決定でかっこを指定しなかった場合、これらのセット演算子の組み合わせは、左から右に評価されます。例えば、次のクエリで

は、T1 と T2 の UNION が最初に評価され、次に UNION の結果に対して、EXCEPT 演算が実行されます。

```
select * from t1
union
select * from t2
except
select * from t3
order by c1;
```

同じクエリ内で演算子の組み合わせを使用した場合、INTERSECT 演算子は UNION および EXCEPT よりも優先されます。例えば、次のクエリは T2 と T3 の積集合を評価し、その結果を T1 を使って結合します。

```
select * from t1
union
select * from t2
intersect
select * from t3
order by c1;
```

かっこを追加することで、評価の順番を変更することができます。次のケースでは、T1 と T2 の結合結果と T3 の積集合を求めます。このクエリでは異なる結果が生成されます。

```
(select * from t1
union
select * from t2)
intersect
(select * from t3)
order by c1;
```

使用に関する注意事項

- セット演算クエリの結果で返される列名は、最初のクエリ式のテーブルからの列名 (またはエイリアス) です。これらの列名は、列内の値はセット演算子の両方のテーブルから生成されるという点で誤解を生む可能性があるため、結果セットには意味のあるエイリアスを付けることをお勧めします。
- セット演算子より前に記述されたクエリ式には、ORDER BY 句を含めないでください。ORDER BY 句は、セット演算子を含むクエリの最後に使用することで、意味のあるソート結果が得られま

す。この場合、ORDER BY 句は、すべてのセット演算の最終結果に適用されます。最も外側のクエリには、標準の LIMIT 句および OFFSET 句を含めることもできます。

- セット演算子クエリが 10 進数の結果を返した場合、同じ精度とスケールで対応する結果列を返すように奨励されます。例えば、T1.REVENUE が DECIMAL(10,2) 列で T2.REVENUE が DECIMAL(8,4) 列の次のクエリでは、DECIMAL(12,4) への結果も 10 進数であることが奨励されます。

  ```
  select t1.revenue union select t2.revenue;
  ```

  スケールは 4 になります。2 つの列の最大のスケールです。精度は 12 です。T1.REVENUE は小数点の左側に 8 桁必要であるからです (12 - 4 = 8)。このような奨励により、UNION の両側からのすべての値が結果に適合します。64 ビットの値の場合、最大結果精度は 19 で、最大結果スケールは 18 です。128 ビットの値の場合、最大結果精度は 38 で、最大結果スケールは 37 です。

  生成されるデータ型が Amazon Redshift の精度とスケールの上限を超えた場合、クエリはエラーを返します。

- 集合演算で 2 行が同一として扱われるのは、対応する列のペアごとに、2 つのデータ値が等しいまたはどちらも NULL である場合です。例えば、テーブル T1 と T2 の両方に 1 つの列と 1 つの行が含まれていて、両方のテーブルでその行が NULL の場合、これらのテーブルに INTERSECT 演算に実行すると、その行が返されます。

UNION クエリの例

次の UNION クエリでは、SALES テーブルの行が、LISTING テーブルの行とマージされます。各テーブルからは 3 つの互換性のある列が選択されます。この場合、対応する列には同じ名前とデータ型が与えられます。

最終結果セットは、LISTING テーブルの最初の列によってソートされ、LISTID の値が高い 5 つの行に制限されます。

```
select listid, sellerid, eventid from listing
union select listid, sellerid, eventid from sales
order by listid, sellerid, eventid desc limit 5;

listid | sellerid | eventid
--------+----------+---------
1 |    36861 |    7872
2 |    16002 |    4806
3 |    21461 |    4256
```

```
4 |     8117 |    4337
5 |     1616 |    8647
(5 rows)
```

次の例では、どのクエリ式が結果セットの各行を生成したかを確認できるように、UNION クエリの出力にリテラル値を追加する方法を示します。このクエリは、最初のクエリ式からの行を (販売者を意味する) 「B」として識別し、2 番目のクエリ式からの行を (購入者を意味する) 「S」として識別します。

このクエリは 10,000 ドル以上のチケット取引の販売者と購入者を識別します。UNION 演算子の両側の 2 つのクエリ式の違いは、SALES テーブルの結合列だけです。

```
select listid, lastname, firstname, username,
pricepaid as price, 'S' as buyorsell
from sales, users
where sales.sellerid=users.userid
and pricepaid >=10000
union
select listid, lastname, firstname, username, pricepaid,
'B' as buyorsell
from sales, users
where sales.buyerid=users.userid
and pricepaid >=10000
order by 1, 2, 3, 4, 5;

listid | lastname | firstname | username |   price   | buyorsell
--------+----------+-----------+----------+-----------+-----------
209658 | Lamb     | Colette   | VOR15LYI |  10000.00 | B
209658 | West     | Kato      | ELU81XAA |  10000.00 | S
212395 | Greer    | Harlan    | GXO71KOC |  12624.00 | S
212395 | Perry    | Cora      | YWR73YNZ |  12624.00 | B
215156 | Banks    | Patrick   | ZNQ69CLT |  10000.00 | S
215156 | Hayden   | Malachi   | BBG56AKU |  10000.00 | B
(6 rows)
```

次の例では、重複行が検出された場合、その重複行を結果に保持する必要があるため、UNION ALL 演算子を使用します。一連の特定イベント ID では、クエリは各イベントに関連付けられているセールスごとに 0 行以上の行を返し、そのイベントのリスティングごとに 0 行または 1 行を返します。イベント ID は、LISTING テーブルと EVENT テーブルの各行に対して一意ですが、SALES テーブルのイベント ID とリスティング ID の同じ組み合わせに対して、複数のセールスが存在することがあります。

結果セットの 3 番目の列は、行のソースを特定します。その行が SALES テーブルからの行だった場合、SALESROW 列に "Yes" というマークが付きます。(SALESROW は SALES.LISTID のエイリアスです。) その行が LISTING テーブルからの行だった場合、SALESROW 列に "No" というマークが付きます。

この場合、リスティング 500、イベント 7787 の結果セットは、3 つの行から構成されます。つまり、このリスティングとイベントの組み合わせに対して、3 つの異なる取引が実行されたということです。他の 2 つのリスティング (501 と 502) では販売はありません。このため、これらのリスト ID に対してクエリが生成した唯一の行は LISTING テーブル (SALESROW = 'No') から生成されます。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(6 rows)
```

ALL キーワードを付けずに同じクエリを実行した場合、結果には、セールス取引の 1 つだけが保持されます。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
```

```
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

UNION ALL クエリの例

次の例では、重複行が検出された場合、その重複行を結果に保持する必要があるため、UNION ALL 演算子を使用します。一連の特定イベント ID では、クエリは各イベントに関連付けられているセールスごとに 0 行以上の行を返し、そのイベントのリスティングごとに 0 行または 1 行を返します。イベント ID は、LISTING テーブルと EVENT テーブルの各行に対して一意ですが、SALES テーブルのイベント ID とリスティング ID の同じ組み合わせに対して、複数のセールスが存在することがあります。

結果セットの 3 番目の列は、行のソースを特定します。その行が SALES テーブルからの行だった場合、SALESROW 列に "Yes" というマークが付きます。(SALESROW は SALES.LISTID のエイリアスです。) その行が LISTING テーブルからの行だった場合、SALESROW 列に "No" というマークが付きます。

この場合、リスティング 500、イベント 7787 の結果セットは、3 つの行から構成されます。つまり、このリスティングとイベントの組み合わせに対して、3 つの異なる取引が実行されたということです。他の 2 つのリスティング (501 と 502) では販売はありません。このため、これらのリスト ID に対してクエリが生成した唯一の行は LISTING テーブル (SALESROW = 'No') から生成されます。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union all
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
7787 |    500 | Yes
7787 |    500 | Yes
6473 |    501 | No
```

```
5108 |    502 | No
(6 rows)
```

ALL キーワードを付けずに同じクエリを実行した場合、結果には、セールス取引の 1 つだけが保持されます。

```
select eventid, listid, 'Yes' as salesrow
from sales
where listid in(500,501,502)
union
select eventid, listid, 'No'
from listing
where listid in(500,501,502)
order by listid asc;

eventid | listid | salesrow
---------+--------+----------
7787 |    500 | No
7787 |    500 | Yes
6473 |    501 | No
5108 |    502 | No
(4 rows)
```

INTERSECT クエリの例

次の例を最初の UNION の例と比較してみます。2 つの例の違いは使われたセット演算子だけですが、結果は大きく異なります。1 つの行だけが同じになります。

```
235494 |    23875 |    8771
```

これは、両方のテーブルから検出された 5 つの行の制限された結果の唯一の行です。

```
select listid, sellerid, eventid from listing
intersect
select listid, sellerid, eventid from sales
order by listid desc, sellerid, eventid
limit 5;

listid | sellerid | eventid
--------+----------+---------
235494 |    23875 |    8771
235482 |     1067 |    2667
```

```
235479 |     1589 |    7303
235476 |    15550 |     793
235475 |    22306 |    7848
(5 rows)
```

次のクエリでは、3 月にニューヨーク市とロサンジェルスの両方で発生した (チケットが販売された) イベントを検索します。2 つのクエリ式の違いは、VENUECITY 列の制約です。

```
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='Los Angeles'
intersect
select distinct eventname from event, sales, venue
where event.eventid=sales.eventid and event.venueid=venue.venueid
and date_part(month,starttime)=3 and venuecity='New York City'
order by eventname asc;

eventname
----------------------------
A Streetcar Named Desire
Dirty Dancing
Electra
Running with Annalise
Hairspray
Mary Poppins
November
Oliver!
Return To Forever
Rhinoceros
South Pacific
The 39 Steps
The Bacchae
The Caucasian Chalk Circle
The Country Girl
Wicked
Woyzeck
(16 rows)
```

EXCEPT クエリの例

TICKET データベースの CATEGORY テーブルには、次の 11 行が含まれています。

```
 catid | catgroup |  catname  |                  catdesc
```

```
-------+----------+-----------+-------------------------------------------
   1   | Sports   | MLB       | Major League Baseball
   2   | Sports   | NHL       | National Hockey League
   3   | Sports   | NFL       | National Football League
   4   | Sports   | NBA       | National Basketball Association
   5   | Sports   | MLS       | Major League Soccer
   6   | Shows    | Musicals  | Musical theatre
   7   | Shows    | Plays     | All non-musical theatre
   8   | Shows    | Opera     | All opera and light opera
   9   | Concerts | Pop       | All rock and pop music concerts
  10   | Concerts | Jazz      | All jazz singers and bands
  11   | Concerts | Classical | All symphony, concerto, and choir concerts
(11 rows)
```

CATEGORY_STAGE テーブル (ステージングテーブル) には、1 つの追加行が含まれていると想定します。

```
 catid | catgroup |  catname  |                  catdesc
-------+----------+-----------+-------------------------------------------
1 | Sports   | MLB       | Major League Baseball
2 | Sports   | NHL       | National Hockey League
3 | Sports   | NFL       | National Football League
4 | Sports   | NBA       | National Basketball Association
5 | Sports   | MLS       | Major League Soccer
6 | Shows    | Musicals  | Musical theatre
7 | Shows    | Plays     | All non-musical theatre
8 | Shows    | Opera     | All opera and light opera
9 | Concerts | Pop       | All rock and pop music concerts
10 | Concerts | Jazz      | All jazz singers and bands
11 | Concerts | Classical | All symphony, concerto, and choir concerts
12 | Concerts | Comedy    | All stand up comedy performances
(12 rows)
```

2 つのテーブル間の違いを返します。つまり、CATEGORY テーブル内の行ではなく、CATEGORY_STAGE テーブル内の行が返されるということです。

```
select * from category_stage
except
select * from category;

catid | catgroup | catname |             catdesc
-------+----------+---------+----------------------------------
```

```
12 | Concerts | Comedy   | All stand up comedy performances
(1 row)
```

次の同等のクエリでは、同義語の MINUS を使用します。

```
select * from category_stage
minus
select * from category;

catid | catgroup | catname |              catdesc
-------+----------+---------+----------------------------------
12 | Concerts | Comedy   | All stand up comedy performances
(1 row)
```

SELECT 式の順番を逆にすると、クエリは行を返しません。

ORDER BY 句

トピック

- 構文
- パラメータ
- 使用に関する注意事項
- ORDER BY の例

ORDER BY 句は、クエリの結果セットをソートします。

構文

```
[ ORDER BY expression [ ASC | DESC ] ]
[ NULLS FIRST | NULLS LAST ]
[ LIMIT { count | ALL } ]
[ OFFSET start ]
```

パラメータ

expression

通常は、SELECT リスト内の 1 つまたは複数の列を指定することで、クエリ結果セットのソート順を定義する式。結果は、バイナリ UTF-8 順序付けに基づいて返されます。以下を指定することもできます。

- SELECT リストにない列
- クエリが参照するテーブル内に存在する、1 つまたは複数の列で構成される式
- SELECT リストエントリの位置 (SELECT リストが存在しない場合は、テーブルの列の位置) を表す序数
- SELECT リストエントリを定義するエイリアス

ORDER BY 句に複数の式が含まれる場合、結果セットは、最初の式に従ってソートされ、次の最初の式の値と一致する値を持つ行に 2 番目の式が適用されます。以降同様の処理が行われます。

ASC | DESC

次のように、式のソート順を定義するオプション:

- ASC: 昇順 (数値の場合は低から高、文字列の場合は「A」から「Z」など) オプションを指定しない場合、データはデフォルトでは昇順にソートされます。
- DESC: 降順 (数値の場合は高から低、文字列の場合は「Z」から「A」)。

NULLS FIRST | NULLS LAST

NULL 値を NULL 以外の値より先に順序付けするか、NULL 以外の値の後に順序付けするかを指定するオプション。デフォルトでは、NULL 値は昇順ではソートされて最後にランク付けされ、降順ではソートされて最初にランク付けされます。

LIMIT number | ALL

クエリが返すソート済みの行数を制御するオプション。LIMIT 数は正の整数でなければなりません。最大値は 2147483647 です。

LIMIT 0 は行を返しません。この構文は、(行を表示せずに) クエリが実行されているかを確認したり、テーブルから列リストを返すためのテスト目的で使用できます。列リストを返すために LIMIT 0 を使用した場合、ORDER BY 句は重複です。デフォルトは LIMIT ALL です。

OFFSET start

行を返す前に、start の前の行数をスキップするよう指定するオプション。OFFSET 数は正の整数でなければなりません。最大値は 2147483647 です。LIMIT オプションを合わせて使用すると、OFFSET 行は、返される LIMIT 行のカウントを開始する前にスキップされます。LIMIT オプションを使用しない場合、結果セット内の行の数が、スキップされる行の数だけ少なくなります。OFFSET 句によってスキップされた行もスキャンされる必要はあるため、大きい OFFSET 値を使用することは一般的に非効率的です。

使用に関する注意事項

ORDER BY 句を使用すると、次の動作が予想されます。

- ヌル値は他のすべての値よりも「高い」と見なされます。デフォルトの昇順のソートでは、NULL 値は最後に表示されます。この動作を変更するには、NULLS FIRST オプションを使用します。
- クエリに ORDER BY 句が含まれていない場合、結果行が返す行の順番は予想不能です。同じクエリを 2 回実行した場合に、結果セットが返される順番が異なることがあります。
- LIMIT オプションと OFFSET オプションは、ORDER BY 句なしに使用できます。ただし、整合性のある行セットを返すには、これらのオプションを ORDER BY と組み合わせて使用してください。
- Amazon Redshift などの並列システムで ORDER BY が一意の順番を生成しない場合、行の順番は非決定性です。つまり、ORDER BY 式が複数の値を生成する場合、行が返される順番は、システムごと、または Amazon Redshift の実行ごとに異なることがあるということです。
- Amazon Redshift は、ORDER BY 句で文字列リテラルをサポートしていません。

ORDER BY の例

CATEGORY テーブルの 11 の行をすべて、2 番目の列 (CATGROUP) でソートして返します。CATGROUP の値が同じ結果の場合、CATDESC 列の値は文字列の長さによってソートされます。次に CATID 列と CATNAME 列でソートされます。

```
select * from category order by 2, length(catdesc), 1, 3;

catid | catgroup |  catname  |                  catdesc
------+----------+-----------+----------------------------------------
10    | Concerts | Jazz      | All jazz singers and bands
9     | Concerts | Pop       | All rock and pop music concerts
11    | Concerts | Classical | All symphony, concerto, and choir conce
6     | Shows    | Musicals  | Musical theatre
7     | Shows    | Plays     | All non-musical theatre
8     | Shows    | Opera     | All opera and light opera
5     | Sports   | MLS       | Major League Soccer
1     | Sports   | MLB       | Major League Baseball
2     | Sports   | NHL       | National Hockey League
3     | Sports   | NFL       | National Football League
4     | Sports   | NBA       | National Basketball Association
(11 rows)
```

SALES テーブルの選択した列を、QTYSOLD 値の高い順にソートして返します。結果を上位の 10 行に制限します。

```
select salesid, qtysold, pricepaid, commission, saletime from sales
order by qtysold, pricepaid, commission, salesid, saletime desc
limit 10;

salesid | qtysold | pricepaid | commission |      saletime
--------+---------+-----------+------------+---------------------
15401   |       8 |    272.00 |      40.80 | 2008-03-18 06:54:56
61683   |       8 |    296.00 |      44.40 | 2008-11-26 04:00:23
90528   |       8 |    328.00 |      49.20 | 2008-06-11 02:38:09
74549   |       8 |    336.00 |      50.40 | 2008-01-19 12:01:21
130232  |       8 |    352.00 |      52.80 | 2008-05-02 05:52:31
55243   |       8 |    384.00 |      57.60 | 2008-07-12 02:19:53
16004   |       8 |    440.00 |      66.00 | 2008-11-04 07:22:31
489     |       8 |    496.00 |      74.40 | 2008-08-03 05:48:55
4197    |       8 |    512.00 |      76.80 | 2008-03-23 11:35:33
16929   |       8 |    568.00 |      85.20 | 2008-12-19 02:59:33
(10 rows)
```

LIMIT 0 構文を使用することで、列リストを返し、行を返しません。

```
select * from venue limit 0;
venueid | venuename | venuecity | venuestate | venueseats
---------+-----------+-----------+------------+------------
(0 rows)
```

CONNECT BY 句

CONNECT BY 句は、階層内の行間の関係を指定します。CONNECT BY を使用すると、テーブルをそれ自体に結合して階層データを処理することで、階層順に行を選択できます。例えば、組織図を再帰的にループ処理してデータを一覧表示できます。

階層型クエリは次の順序で処理されます。

1. FROM 句に結合がある場合は、最初に処理されます。
2. CONNECT BY 句が評価されます。
3. WHERE 句が評価されます。

構文

```
[START WITH start_with_conditions]
CONNECT BY connect_by_conditions
```

> **Note**
>
> START と CONNECT は予約語ではありませんが、ランタイムの失敗を避けるために、クエリでテーブルのエイリアスとして START と CONNECT を使用する場合は、区切られた識別子 (二重引用符) または AS を使用してください。

```
SELECT COUNT(*)
FROM Employee "start"
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

```
SELECT COUNT(*)
FROM Employee AS start
CONNECT BY PRIOR id = manager_id
START WITH name = 'John'
```

パラメータ

start_with_conditions

階層のルート行を指定する条件

connect_by_conditions

階層の親行と子行の関係を指定する条件。少なくとも 1 つの条件は、親行を参照する　単項演算子で修飾される必要があります。

```
PRIOR column = expression
-- or
expression > PRIOR column
```

演算子

CONNECT BY クエリでは、次の演算子を使用できます。

LEVEL

階層内の現在の行レベルを返す疑似列。ルート行には 1、ルート行の子には 2、というように返します。

PRIOR

階層内の現在の行の親行の式を評価する単項演算子。

例

次の例は、John に直接的に、または間接的に報告する従業員の数を返す CONNECT BY クエリで、深さは 4 レベル未満です。

```
SELECT id, name, manager_id
FROM employee
WHERE LEVEL < 4
START WITH name = 'John'
CONNECT BY PRIOR id = manager_id;
```

以下は、クエリの結果です。

```
id      name      manager_id
------+----------+--------------
  101     John        100
  102     Jorge       101
  103     Kwaku       101
  110     Liu         101
  201     Sofía       102
  106     Mateo       102
  110     Nikki       103
  104     Paulo       103
  105     Richard     103
  120     Saanvi      104
  200     Shirley     104
  205     Zhang       104
```

例えば、テーブルは次のように定義されています。

```
CREATE TABLE employee (
   id INT,
```

```
    name VARCHAR(20),
    manager_id INT
    );
```

以下は、テーブルに挿入された行です。

```
INSERT INTO employee(id, name, manager_id)  VALUES
(100, 'Carlos', null),
(101, 'John', 100),
(102, 'Jorge', 101),
(103, 'Kwaku', 101),
(110, 'Liu', 101),
(106, 'Mateo', 102),
(110, 'Nikki', 103),
(104, 'Paulo', 103),
(105, 'Richard', 103),
(120, 'Saanvi', 104),
(200, 'Shirley', 104),
(201, 'Sofía', 102),
(205, 'Zhang', 104);
```

以下は、John が所属する部門の組織図です。

サブクエリの例

次の例は、サブクエリが SELECT クエリに適合するさまざまな方法を示しています。サブクエリの使用に関する別の例については、「JOIN 句の例」を参照してください。

SELECT リストのサブクエリ

次の例には、SELECT リストのサブクエリが含まれています。このサブクエリはスカラー値であり、1 つの列と 1 つの値のみを返します。外部クエリから返される行の結果ごとに、このサブクエリが繰り返されます。このクエリは、サブクエリが計算した Q1SALES 値を、外部クエリが定義する、2008 年の他の 2 つの四半期 (第 2 と第 3) のセールス値と比較します。

```
select qtr, sum(pricepaid) as qtrsales,
(select sum(pricepaid)
from sales join date on sales.dateid=date.dateid
where qtr='1' and year=2008) as q1sales
from sales join date on sales.dateid=date.dateid
```

```
where qtr in('2','3') and year=2008
group by qtr
order by qtr;

qtr  |  qtrsales   |   q1sales
-------+-------------+-------------
2     | 30560050.00 | 24742065.00
3     | 31170237.00 | 24742065.00
(2 rows)
```

WHERE 句のサブクエリ

次の例には、WHERE 句にテーブルサブクエリが含まれます。このサブクエリは複数の行を生成します。この場合、その行には列が 1 つだけ含まれていますが、テーブルサブクエリには他のテーブルと同様、複数の列と行が含まれていることがあります。

このクエリは、最大販売チケット数の観点でトップ 10 の販売会社を検索します。トップ 10 のリストは、チケットカウンターが存在する都市に住んでいるユーザーを削除するサブクエリによって制限されます。このクエリは、メインクエリ内の結合としてサブクエリを作成するなど、さまざまな方法で作成できます。

```
select firstname, lastname, city, max(qtysold) as maxsold
from users join sales on users.userid=sales.sellerid
where users.city not in(select venuecity from venue)
group by firstname, lastname, city
order by maxsold desc, city desc
limit 10;

firstname | lastname  |      city      | maxsold
-----------+-----------+----------------+---------
Noah       | Guerrero | Worcester      |       8
Isadora    | Moss     | Winooski       |       8
Kieran     | Harrison | Westminster    |       8
Heidi      | Davis    | Warwick        |       8
Sara       | Anthony  | Waco           |       8
Bree       | Buck     | Valdez         |       8
Evangeline | Sampson  | Trenton        |       8
Kendall    | Keith    | Stillwater     |       8
Bertha     | Bishop   | Stevens Point  |       8
Patricia   | Anderson | South Portland |       8
(10 rows)
```

WITH 句のサブクエリ

「WITH 句」を参照してください。

相関性のあるサブクエリ

次の例の WHERE 句には、相関性のあるサブクエリが含まれています。このタイプのサブクエリには、サブクエリの列と他のクエリが生成した列の間に相関性があります。この場合、相関は where s.listid=l.listid となります。外部クエリが生成する各行に対してサブクエリが実行され、行が適正か適正でないかが判断されます。

```
select salesid, listid, sum(pricepaid) from sales s
where qtysold=
(select max(numtickets) from listing l
where s.listid=l.listid)
group by 1,2
order by 1,2
limit 5;

salesid | listid |   sum
--------+--------+----------
 27     |     28 | 111.00
 81     |    103 | 181.00
 142    |    149 | 240.00
 146    |    152 | 231.00
 194    |    210 | 144.00
(5 rows)
```

サポートされていない相関サブクエリのパターン

クエリのプランナーは、MPP 環境で実行する相関サブクエリの複数のパターンを最適化するため、「サブクエリ相関解除」と呼ばれるクエリ再生成メソッドを使用します。相関サブクエリの中には、Amazon Redshift が相関を解除できずサポートすることができないパターンを採用しているものがいくつかあります。次の相関参照を含んでいるクエリがエラーを返します。

- クエリブロックをスキップする相関参照 (「スキップレベル相関参照」とも呼ばれています) 例えば、次のクエリでは、相関参照とスキップされるブロックを含むブロックは、NOT EXISTS 述語によって接続されます。

  ```
  select event.eventname from event
  where not exists
  ```

```
(select * from listing
where not exists
(select * from sales where event.eventid=sales.eventid));
```

このケースでスキップされたブロックは、LISTING テーブルに対するサブクエリです。相関参照は、EVENT テーブルと SALES テーブルを関係付けます。

- 外部クエリで ON 句の一部であるサブクエリからの相関参照:

```
select * from category
left join event
on category.catid=event.catid and eventid =
(select max(eventid) from sales where sales.eventid=event.eventid);
```

ON 句には、サブクエリの SALES から外部クエリの EVENT への相関参照が含まれています。

- Amazon Redshift システムテーブルに対する NULL センシティブな相関参照。次に例を示します。

```
select attrelid
from stv_locks sl, pg_attribute
where sl.table_id=pg_attribute.attrelid and 1 not in
(select 1 from pg_opclass where sl.lock_owner = opcowner);
```

- ウィンドウ関数を含んでいるサブクエリ内からの相関参照。

```
select listid, qtysold
from sales s
where qtysold not in
(select sum(numtickets) over() from listing l where s.listid=l.listid);
```

- 相関サブクエリの結果に対する、GROUP BY 列の参照。次に例を示します。

```
select listing.listid,
(select count (sales.listid) from sales where sales.listid=listing.listid) as list
from listing
group by list, listing.listid;
```

- 集計関数と GROUP BY 句のあり、IN 述語によって外部クエリに接続されているサブクエリからの相関参照。(この制限は、MIN と MAX 集計関数には適用されません。) 例:

```
select * from listing where listid in
(select sum(qtysold)
```

```
from sales
where numtickets>4
group by salesid);
```

SELECT INTO

任意のクエリによって定義された行を選択して、新しいテーブルに挿入します。一時テーブルと永続的テーブルのどちらを作成するかを指定できます。

構文

```
[ WITH with_subquery [, ...] ]
SELECT
[ TOP number ] [ ALL | DISTINCT ]
* | expression [ AS output_name ] [, ...]
INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table
[ FROM table_reference [, ...] ]
[ WHERE condition ]
[ GROUP BY expression [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | { EXCEPT | MINUS } } [ ALL ] query ]
[ ORDER BY expression
[ ASC | DESC ]
[ LIMIT { number | ALL } ]
[ OFFSET start ]
```

このコマンドのパラメータに関する詳細については、「SELECT」を参照してください。

例

EVENT テーブルからのすべての行を選択し、NEWEVENT テーブルを作成します。

```
select * into newevent from event;
```

集計クエリの結果を選択して、PROFITS という名前の一時テーブルに格納します。

```
select username, lastname, sum(pricepaid-commission) as profit
into temp table profits
from sales, users
where sales.sellerid=users.userid
group by 1, 2
```

```
order by 3 desc;
```

SET

サーバー設定パラメータの値を設定します。SET コマンドを使用して、現在のセッションまたはトランザクションのみの期間の設定をオーバーライドします。

RESET コマンドを使って、パラメータをデフォルト値に戻します。

サーバー構成パラメータは、いくつかの方法で変更できます。詳細については、「サーバー設定の変更」を参照してください。

構文

```
SET { [ SESSION | LOCAL ]
{ SEED | parameter_name } { TO | = }
{ value | 'value' | DEFAULT } |
SEED TO value }
```

次のステートメントは、セッションコンテキスト変数の値を設定します。

```
SET { [ SESSION | LOCAL ]
variable_name { TO | = }
{ value | 'value'  }
```

パラメータ

SESSION

現在のセッションに対して、設定が有効であることを指定します。デフォルト値

variable_name

セッションに設定されたコンテキスト変数の名前を指定します。

命名規則は、ドットで区切られた 2 つの部分で構成される名前です (例: identifier.identifier)。使用できるドット区切り文字は 1 つだけです。Amazon Redshift の標準識別子ルールに従った識別子を使用します。詳細については、「名前と識別子」を参照してください。区切り文字は許可されていません。

LOCAL

設定が現在の取引で有効なことを指定します。

SEED TO value

乱数生成用に RANDOM 関数が使用する内部シードを設定します。

SET SEED は、0 から 1 までの数値を取り、この数値に (2^{31}-1) を掛けて、RANDOM 関数関数で使用します。複数の RANDOM コールの前に SET SEED を使用すると、RANDOM は予想可能な順番で番号を生成します。

parameter_name

設定するパラメータの名前。パラメータの詳細については、「サーバー設定の変更」を参照してください。

value

新しいパラメータ値。一重引用符を使って、指定文字列に値を設定します。SET SEED を使用した場合、このパラメータには SEED 値が含まれます。

DEFAULT

パラメータをデフォルト値に設定します。

例

現在のセッションのパラメータの変更

次の例は、データスタイルを設定します。

```
set datestyle to 'SQL,DMY';
```

ワークロード管理用のクエリグループの設定

クラスターの WLM 設定の一部として、クエリグループをキュー定義で一覧表示した場合、一覧表示されたクエリクループ名に QUERY_GROUP パラメータを設定できます。以降のクエリは、関連するクエリキューに割り当てられます。QUERY_GROUP グループの設定は、セッションの有効期間中、または RESET QUERY_GROUP コマンドの遭遇するまで有効です。

この例では、クエリグループ「priority」の一部として 2 つのクエリを実行し、その後クエリグループをリセットします。

```
set query_group to 'priority';
select tbl, count(*)from stv_blocklist;
```

```
select query, elapsed, substring from svl_qlog order by query desc limit 5;
reset query_group;
```

詳細については、「ワークロード管理」を参照してください。

セッションのデフォルトの ID 名前空間を変更する

データベースユーザーは default_identity_namespace を設定できます。このサンプルでは、SET SESSION を使用して現在のセッション期間中に設定を上書きし、新しい ID プロバイダーの値を表示する方法を示します。これは、Redshift と IAM アイデンティティセンターで ID プロバイダーを使用している場合に最もよく使用されます。Redshift で ID プロバイダーを使用する方法の詳細については、「Redshift を IAM アイデンティティセンターに接続してユーザーにシングルサインオンエクスペリエンスを提供する」を参照してください。

```
SET SESSION default_identity_namespace = 'MYCO';

SHOW default_identity_namespace;
```

コマンドを実行したら、次のような GRANT ステートメントまたは CREATE ステートメントを実行できます。

```
GRANT SELECT ON TABLE mytable TO alice;

GRANT UPDATE ON TABLE mytable TO salesrole;

CREATE USER bob password 'md50c983d1a624280812631c5389e60d48c';
```

この場合、デフォルトの ID 名前空間を設定する効果は、各 ID に名前空間のプレフィックスを付けることと同じです。この例では、alice が MYCO:alice に置き換えられます。IAM アイデンティティセンターでの Redshift 構成に関連する設定の詳細については、「ALTER SYSTEM」および「他の ID プロバイダー」を参照してください。

クエリのグループのラベルの設定

QUERY_GROUP パラメータは、SET コマンド実行後の同じセッションで実行される 1 つまたは複数のクエリのためのラベルを定義します。また、実行されたクエリが STL_QUERY や STV_INFLIGHT システムテーブル、および SVL_QLOG ビューから返される結果を制約する場合に、このラベルがログ記録されます。

```
show query_group;
```

```
query_group
-------------
unset
(1 row)

set query_group to '6 p.m.';


show query_group;
query_group
-------------
6 p.m.
(1 row)

select * from sales where salesid=500;
salesid | listid | sellerid | buyerid | eventid | dateid | ...
---------+--------+----------+---------+---------+--------+-----
500 |    504 |     3858 |    2123 |    5871 |   2052 | ...
(1 row)

reset query_group;

select query, trim(label) querygroup, pid, trim(querytxt) sql
from stl_query
where label ='6 p.m.';
query | querygroup |  pid  |                  sql
-------+------------+-------+----------------------------------------
57 | 6 p.m.     | 30711 | select * from sales where salesid=500;
(1 row)
```

クエリグループのラベルは、スクリプトの一部として実行された個々のクエリやクエリグループを分離するための有益なメカニズムです。クエリの ID によってクエリの識別や追跡を行う必要はありません。ラベルによってクエリの追跡が可能です。

乱数生成用のシード値の設定

次の例では、SET コマンドで SEED オプションを使用し、RANDOM 関数により、予想可能な順番で数値を生成します。

まず、SEED 値を最初に設定せずに、3 つの整数の乱数を返します。

```
select cast (random() * 100 as int);
int4
```

```
------
6
(1 row)

select cast (random() * 100 as int);
int4
------
68
(1 row)

select cast (random() * 100 as int);
int4
------
56
(1 row)
```

次に、SEED 値を .25 に設定して、さらに 3 つの整数の乱数を返します。

```
set seed to .25;

select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
------
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

最後に、SEED 値を .25 にリセットして、RANDOM が前の 3 つの呼び出しと同じ結果を返すことを確認します。

```
set seed to .25;
```

```
select cast (random() * 100 as int);
int4
------
21
(1 row)

select cast (random() * 100 as int);
int4
------
79
(1 row)

select cast (random() * 100 as int);
int4
------
12
(1 row)
```

次の例では、カスタマイズされたコンテキスト変数を設定します。

```
SET app_context.user_id TO 123;
SET app_context.user_id TO 'sample_variable_value';
```

SET SESSION AUTHORIZATION

現在のセッションのユーザー名を設定します。

SET SESSION AUTHORIZATION コマンドは、権限のないユーザーとしてセッションやトランザクションを一時的に実行することで、データベースアクセスをテストする場合などに使用できます。このコマンドを実行するには、データベースのスーパーユーザー権限を持つ必要があります。

構文

```
SET [ LOCAL ] SESSION AUTHORIZATION { user_name | DEFAULT }
```

パラメータ

LOCAL

設定が現在の取引で有効なことを指定します。このパラメータを省略すると、現在のセッションに対する設定が有効であることを指定したことになります。

user_name

設定するユーザー名。ユーザー名は、識別子またはリテラル文字列として指定できます。

DEFAULT

セッションユーザー名をデフォルト値に設定します。

例

次の例では、現在のセッションのユーザー名を dwuser に設定します。

```
SET SESSION AUTHORIZATION 'dwuser';
```

次の例では、現在のトランザクションのユーザー名を dwuser に設定します。

```
SET LOCAL SESSION AUTHORIZATION 'dwuser';
```

この例では、現在のセッションのユーザー名をデフォルトのユーザー名に設定します。

```
SET SESSION AUTHORIZATION DEFAULT;
```

SET SESSION CHARACTERISTICS

このコマンドは廃止されました。

SHOW

サーバー 設定パラメータの現在の値を表示します。SET コマンドが有効な場合、この値は、現在のセッションに固有となることがあります。設定パラメータのリストについては、「設定リファレンス」を参照してください。

構文

```
SHOW { parameter_name | ALL }
```

次のステートメントは、セッションコンテキスト変数の現在の値を表示します。変数が存在しない場合、Amazon Redshift はエラーをスローします。

```
SHOW variable_name
```

パラメータ

parameter_name

指定したパラメータの現在の値を表示します。

ALL

すべてのパラメータの現在の値を表示します。

variable_name

指定した変数の現在の値を表示します。

例

次の例では、query_group パラメータの値を表示します。

```
show query_group;

query_group

unset
(1 row)
```

次の例では、すべてのパラメータとその値を一覧表示します。

```
show all;
name        |   setting
--------------------+--------------
datestyle          | ISO, MDY
extra_float_digits | 0
query_group        | unset
search_path        | $user,public
statement_timeout  | 0
```

次の例では、指定した変数の現在の値を表示します。

```
SHOW app_context.user_id;
```

SHOW COLUMNS

テーブル内の列のリストと、いくつかの列属性を表示します。

各出力行は、データベース名、スキーマ名、テーブル名、列名、序数位置、列のデフォルト、NULL 指定可、データ型、最大文字長、数値精度、および注釈をカンマで区切ったリストで構成されます。これらの属性の詳細については、「SVV_ALL_COLUMNS」を参照してください。

SHOW COLUMNS コマンドの結果が 10,000 列を超える場合は、エラーが返されます。

構文

```
SHOW COLUMNS FROM TABLE database_name.schema_name.table_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

パラメータ

database_name

一覧表示するテーブルが含まれているデータベースの名前。

AWS Glue Data Catalog でテーブルを表示するには、データベース名として (`awsdatacatalog`) を指定し、システム設定 `data_catalog_auto_mount` が `true` に設定されていることを確認します。詳細については、「ALTER SYSTEM」を参照してください。

schema_name

一覧表示するテーブルが含まれているスキーマの名前。

AWS Glue Data Catalog のテーブルを表示するには、AWS Glue データベース名をスキーマ名として指定します。

table_name

一覧表示する列が含まれているテーブルの名前。

filter_pattern

テーブル名とマッチングするパターンが含まれる有効な UTF-8 文字式。LIKE オプションでは、以下のパターンマッチングメタ文字をサポートする、大文字と小文字を区別したマッチングを行います。

メタ文字	説明
%	ゼロ個以上の任意の文字シーケンスをマッチングします。
_	任意の 1 文字をマッチングします。

filter_pattern にメタ文字が含まれていない場合、パターンは文字列そのものを表すだけです。この場合、LIKE は等号演算子と同じ働きをします。

row_limit

返される行の最大数。row_limit には、0～10,000 の値を指定できます。

例

次の例は、スキーマ `public` とテーブル `tb` にある、`dev` という Amazon Redshiftデータベース内の列を示しています。

```
SHOW COLUMNS FROM TABLE dev.public.tb;

 database_name | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
---------------+-------------+------------+-------------+------------------
+----------------+-------------+-----------+--------------------------
+-------------------+---------
 dev           | public      | tb         | col         |                1 |
     | YES          | integer   |                           |                 32 |
```

次の例は、スキーマ `batman` とテーブル `nation` にある、`awsdatacatalog` という AWS Glue Data Catalog データベース内のテーブルを示しています。出力は、2 行に制限されます。

```
SHOW COLUMNS FROM TABLE awsdatacatalog.batman.nation LIMIT 2;

 database_name  | schema_name | table_name | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | remarks
----------------+-------------+------------+-------------+------------------
+----------------+-------------+-----------+--------------------------
+-------------------+---------
```

```
awsdatacatalog | batman       | nation      | n_nationkey |                 1 |
      |               | integer    |                           |                    |
awsdatacatalog | batman       | nation      | n_name      |                 2 |
      |               | character |                           |                    |
```

SHOW EXTERNAL TABLE

テーブル属性と列属性を含む、外部テーブルの定義を表示します。SHOW EXTERNAL TABLE ステートメントの出力を使用すると、テーブルを再作成できます。

外部テーブル作成の詳細については、「CREATE EXTERNAL TABLE」を参照してください。

構文

```
SHOW EXTERNAL TABLE [external_database].external_schema.table_name [ PARTITION ]
```

パラメータ

external_database

関連付けられた外部データベースの名前。このパラメータはオプションです。

external_schema

関連付けられた外部スキーマの名前。

table_name

表示するテーブルの名前。

PARTITION

テーブル定義にパーティションを追加する ALTER TABLEE 句を表示します。

例

以下の例は、ここに定義された外部テーブルに基づいています。

```
CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
     csmallint smallint,
     cint int,
     cbigint bigint,
```

```
      cfloat float4,
      cdouble float8,
      cchar char(10),
      cvarchar varchar(255),
      cdecimal_small decimal(18,9),
      cdecimal_big decimal(30,15),
      ctimestamp TIMESTAMP,
      cboolean boolean,
      cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime TIMESTAMP)
STORED AS PARQUET
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';
```

以下は、`my_schema.alldatatypes_parquet_test_partitioned`のテーブルに関する SHOW EXTERNAL TABLE コマンドと、その出力の例です。

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';"
```

以下は、同じテーブル (ただし、データベースがパラメータでも指定されている) に関する SHOW EXTERNAL TABLE コマンドと、その出力の例です。

```
SHOW EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned;
```

```
"CREATE EXTERNAL TABLE my_database.my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
)
PARTITIONED BY (cdate date, ctime timestamp)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';"
```

PARTITION パラメータを使用する場合の、SHOW EXTERNAL TABLE コマンドと、その出力の例を以下に示します。この出力には、テーブル定義にパーティションを追加する ALTER TABLEE 句が含まれています。

```
SHOW EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned PARTITION;
```

```
"CREATE EXTERNAL TABLE my_schema.alldatatypes_parquet_test_partitioned (
    csmallint smallint,
    cint int,
    cbigint bigint,
    cfloat float4,
    cdouble float8,
    cchar char(10),
    cvarchar varchar(255),
    cdecimal_small decimal(18,9),
    cdecimal_big decimal(30,15),
    ctimestamp timestamp,
    cboolean boolean,
    cstring varchar(16383)
```

```
)
PARTITIONED BY (cdate date)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://amzn-s3-demo-bucket/alldatatypes_parquet_partitioned';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-01') LOCATION 's3://amzn-s3-demo-bucket/
alldatatypes_parquet_partitioned2/cdate=2021-01-01';
ALTER TABLE my_schema.alldatatypes_parquet_test_partitioned ADD IF NOT
 EXISTS PARTITION (cdate='2021-01-02') LOCATION 's3://amzn-s3-demo-bucket/
alldatatypes_parquet_partitioned2/cdate=2021-01-02';"
```

SHOW DATABASES

指定したアカウント ID のデータベースを表示します。

構文

```
SHOW DATABASES FROM
DATA CATALOG [ ACCOUNT  '<id1>', '<id2>', ... ]
[ LIKE '<expression>' ]
[ IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>' ]
```

パラメータ

ACCOUNT '<id1>', '<id2>', ...

データベースを一覧表示する AWS Glue Data Catalog アカウント。このパラメータを省略すると、Amazon Redshift はクラスターを所有するアカウントのデータベースを表示します。

LIKE '<expression>'

データベースのリストをフィルタリングして、指定した式と一致するものに絞り込みます。このパラメータは、ワイルドカード文字 % (パーセント) および _ (アンダースコア) を使用するパターンをサポートしています。

IAM_ROLE default | 'SESSION' | 'arn:aws:iam::<account-id>:role/<role-name>'

SHOW DATABASES コマンドの実行時にクラスターに関連付けられた IAM ロールを指定すると、Amazon Redshift はデータベースに対するクエリの実行時にロールの認証情報を使用します。

default キーワードを指定することは、デフォルトとして設定されてクラスターに関連付けられている IAM ロールを使用することを意味します。

フェデレーション ID を使用して Amazon Redshift クラスターに接続し、the section called “CREATE DATABASE” コマンドを使用して作成した外部データベースのテーブルにアクセスする場合は、'SESSION' を使用します。フェデレーション ID の使用例については、フェデレーション ID の設定方法を説明している「フェデレーション ID を使用して、ローカルリソースと Amazon Redshift Spectrum の外部テーブルへの Amazon Redshift アクセスを管理する」を参照してください。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。少なくとも、IAM ロールには、Amazon S3 バケットで LIST オペレーションを実行してアクセスを受ける許可と、バケットに含まれる Amazon S3 オブジェクトで GET オペレーションを実行する許可が必要です。データ共有のために AWS Glue Data Catalog で作成するデータベースと IAM_ROLE の使用に関する詳細については、「コンシューマーとしての Lake Formation 管理のデータ共有を使用する」を参照してください。

以下に ARN が 1 つの場合の IAM_ROLE パラメータ文字列の構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-name>'
```

ロールを連鎖することで、クラスターは別のアカウントに属している可能性がある別の IAM ロールを引き受けることができます。最大10 個までのロールを連鎖できます。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。

この IAM ロールに次のような IAM アクセス許可ポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
```

```
            "Resource": "arn:aws:secretsmanager:us-west-2:123456789012:secret:my-
rds-secret-VNenFy"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

フェデレーションクエリで使用する IAM ロールを作成するステップについては、「フェデレーテッドクエリを使用するためのシークレットと IAM ロールの作成」を参照してください。

> **Note**
>
> 連鎖したロールのリストには空白を含めないでください。

以下に連鎖された 3 つのロールの構文を示します。

```
IAM_ROLE 'arn:aws:iam::<aws-account-id>:role/<role-1-name>,arn:aws:iam::<aws-
account-id>:role/<role-2-name>,arn:aws:iam::<aws-account-id>:role/<role-3-name>'
```

例

次の例では、アカウント ID 123456789012 のすべてのデータカタログデータベースを表示します。

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012'

  catalog_id   | database_name |                          database_arn
 |      type      |                                                   target_database
                             | location | parameters
--------------+---------------+--------------------------------------------------------
+--------------
```

```
+----------------------------------------------------------------------------------------
+----------+------------
 123456789012 |   database1   | arn:aws:glue:us-east-1:123456789012:database/database1
 | Data Catalog |
                              |          |
 123456789012 |   database2   | arn:aws:glue:us-east-1:123456789012:database/database2
 | Data Catalog | arn:aws:redshift:us-
east-1:123456789012:datashare:035c45ea-61ce-86f0-8b75-19ac6102c3b7/database2 |
 |
```

IAM ロールの認証情報を使用して、アカウント ID 123456789012 のすべてのデータカタログデータベースを表示する方法を示す例は、以下のとおりです。

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE default;
```

```
SHOW DATABASES FROM DATA CATALOG ACCOUNT '123456789012' IAM_ROLE <iam-role-arn>;
```

SHOW MODEL

ステータス、モデルの作成に使用されたパラメータ、入力引数タイプを含む予測関数など、機械学習モデルに関する有用な情報を表示します。SHOW MODEL からの情報を使用して、モデルを再作成できます。ベーステーブルが変更されている場合、同じ SQL ステートメントで CREATE MODEL を実行すると、異なるモデルになります。SHOW MODEL によって返される情報は、モデルの所有者と EXECUTE 権限を持つユーザーによって異なります。SHOW MODEL は、モデルが Amazon Redshift からトレーニングされている場合、またはモデルが BYOM モデルである場合に、さまざまな出力を表示します。

構文

```
SHOW MODEL ( ALL | model_name )
```

パラメータ

ALL

ユーザーが使用できるすべてのモデルとそのスキーマを返します。

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

使用に関する注意事項

SHOW MODEL コマンドは、以下を返します。

- モデル名。
- モデルが作成されたスキーマ。
- モデルの所有者。
- モデルの作成時刻。
- モデルのステータス (READY、TRAINING、または FAILED など)。
- 失敗したモデルの失敗理由に関するメッセージ。
- モデルがトレーニングを終了した場合の検証エラー。
- BYOM アプローチ以外のモデルを導出するために必要な推定コスト。モデルの所有者のみがこの情報を表示できます。
- ユーザー指定のパラメータとその値のリスト。具体的には以下のとおりです。
  - 指定された TARGET 列。
  - モデルタイプ (AUTO または XGBoost)。
  - 問題の種類 (REGRESSION、BINARY_CLASSIFICATION、MULTICLASS_CLASSIFICATION など)。このパラメータは AUTO に固有です。
  - モデルを作成した Amazon SageMaker トレーニングジョブまたは Amazon SageMaker Autopilot ジョブの名前。このジョブ名を使用して、Amazon SageMaker のモデルに関する詳細情報を確認できます。
  - MSE、F1、精度などの目標。このパラメータは AUTO に固有です。
  - 作成された関数の名前。
  - 推論のタイプ (ローカルまたはリモート)。
  - 予測関数の入力引数。
  - 独自のモデル (BYOM) を持たないモデルの予測関数入力引数タイプ。
  - 予測関数の戻り値の型。このパラメータは BYOM に固有です。
  - リモート推論を持つ BYOM モデルの Amazon SageMaker エンドポイントの名前。
  - IAM ロール。モデルの所有者だけがこれを見ることができます。

- 使用される S3 バケット。モデルの所有者だけがこれを見ることができます。
- AWS KMS キー (提供されている場合)。モデルの所有者だけがこれを見ることができます。
- モデルを実行できる最大時間。
- モデルタイプが AUTO ではない場合、Amazon Redshift は提供されるハイパーパラメータとその値のリストも表示します。

SHOW MODEL によって提供される情報の一部は、pg_proc などの他のカタログテーブルで表示することもできます。Amazon Redshift は、pg_proc カタログテーブルに登録されている予測関数に関する情報を返します。この情報には、予測関数の入力引数名とその型が含まれます。Amazon Redshift は、SHOW MODEL コマンドで同じ情報を返します。

```
SELECT * FROM pg_proc WHERE proname ILIKE '%<function_name>%';
```

例

以下の例は、モデル表示の出力を示しています。

```
SHOW MODEL ALL;

Schema Name |  Model Name
------------+---------------
 public     | customer_churn
```

customer_churn の所有者は、次の出力を確認できます。EXECUTE 権限のみを持つユーザーは、IAM ロール、Amazon S3 バケット、およびモードの見積もりコストを確認することができません。

```
SHOW MODEL customer_churn;

       Key                 |           Value
---------------------------+-----------------------------------
 Model Name                | customer_churn
 Schema Name               | public
 Owner                     | 'owner'
 Creation Time             | Sat, 15.01.2000 14:45:20
 Model State               | READY
 validation:F1             | 0.855
 Estimated Cost            | 5.7
```

```
                            |
TRAINING DATA:              |
Table                       | customer_data
Target Column               | CHURN
                            |
PARAMETERS:                 |
Model Type                  | auto
Problem Type                | binary_classification
Objective                   | f1
Function Name               | predict_churn
Function Parameters         | age zip average_daily_spend average_daily_cases
Function Parameter Types    | int int float float
IAM Role                    | 'iam_role'
KMS Key                     | 'kms_key'
Max Runtime                 | 36000
```

SHOW DATASHARES

同じアカウントまたは複数のアカウント間で、クラスター内のインバウンドおよびアウトバウンドの共有を表示します。データ共有名を指定しない場合、Amazon Redshift はクラスター内のすべてのデータベースのすべてのデータ共有を表示します。ALTER および SHARE 権限を持つユーザーは、権限を持つ共有を表示できます。

構文

```
SHOW DATASHARES [ LIKE 'namepattern' ]
```

パラメータ

LIKE

指定された名前パターンをデータ共有の説明と比較するオプションの句。この句を使用すると、Amazon Redshift は指定された名前パターンに一致する名前のデータ共有のみを表示します。

namepattern

リクエストされたデータ共有の名前、またはワイルドカード文字を使用して照合する名前の一部。

例

次の例では、クラスター内のインバウンド共有とアウトバウンド共有を表示します。

```
SHOW DATASHARES;
SHOW DATASHARES LIKE 'sales%';

share_name   | share_owner | source_database | consumer_database | share_type |
 createdate          | is_publicaccessible | share_acl | producer_account |
 producer_namespace
-------------+-------------+-----------------+-------------------+------------
+---------------------+---------------------+-----------+------------------
+---------------------------------------
'salesshare' | 100         | dev             |                   | outbound
 | 2020-12-09 01:22:54.| False               |           |   123456789012   |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
```

SHOW PROCEDURE

特定のストアドプロシージャの定義を、その署名も含めて示します。SHOW PROCEDURE の出力を使用してストアドプロシージャを再作成できます。

構文

```
SHOW PROCEDURE sp_name [( [ [ argname ] [ argmode ] argtype [, ...] ] )]
```

パラメータ

sp_name

表示するプロシージャの名前。

[argname] [argmode] argtype

ストアドプロシージャを識別するための入力引数タイプ。必要に応じて、OUT 引数も含めて、完全な引数のデータタイプを指定できます。ストアドプロシージャの名前が一意である (重複していない) 場合、この部分は省略可能です。

例

次の例は、プロシージャ `test_spl2` の定義を示しています。

```
show procedure test_sp2(int, varchar);
                                        Stored Procedure Definition
------------------------------------------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE public.test_sp2(f1 integer, INOUT f2 character varying, OUT
 character varying)
LANGUAGE plpgsql
AS $_$
DECLARE
out_var alias for $3;
loop_var int;
BEGIN
IF f1 is null OR f2 is null THEN
RAISE EXCEPTION 'input cannot be null';
END IF;
CREATE TEMP TABLE etl(a int, b varchar);
FOR loop_var IN 1..f1 LOOP
insert into etl values (loop_var, f2);
f2 := f2 || '+' || f2;
END LOOP;
SELECT INTO out_var count(*) from etl;
END;
$_$

(1 row)
```

SHOW SCHEMAS

データベース内のスキーマのリストと、いくつかのスキーマ属性を表示します。

各出力行は、データベース名、スキーマ名、スキーマ所有者、スキーマタイプ、スキーマ ACL、ソースデータベース、およびスキーマオプションで構成されます。これらの属性の詳細については、「SVV_ALL_SCHEMAS」を参照してください。

SHOW SCHEMAS コマンドで 10,000 を超えるスキーマが生成される場合は、エラーが返されます。

構文

```
SHOW SCHEMAS FROM DATABASE database_name [LIKE 'filter_pattern'] [LIMIT row_limit ]
```

パラメータ

database_name

一覧表示するテーブルが含まれているデータベースの名前。

AWS Glue Data Catalog でテーブルを表示するには、データベース名として (`awsdatacatalog`) を指定し、システム設定 `data_catalog_auto_mount` が `true` に設定されていることを確認します。詳細については、「ALTER SYSTEM」を参照してください。

filter_pattern

スキーマ名とマッチングするパターンが含まれる有効な UTF-8 文字式。LIKE オプションでは、以下のパターンマッチングメタ文字をサポートする、大文字と小文字を区別したマッチングを行います。

メタ文字	説明
`%`	ゼロ個以上の任意の文字シーケンスをマッチングします。
`_`	任意の 1 文字をマッチングします。

filter_pattern にメタ文字が含まれていない場合、パターンは文字列そのものを表すだけです。この場合、LIKE は等号演算子と同じ働きをします。

row_limit

返される行の最大数。row_limit には、0～10,000 の値を指定できます。

例

次の例は、`dev` という Amazon Redshift データベース内のスキーマを表示します。

```
SHOW SCHEMAS FROM DATABASE dev;

 database_name |     schema_name      | schema_owner | schema_type |          schema_acl
          | source_database | schema_option
---------------+----------------------+--------------+-------------
+-----------------------------+-----------------+---------------
 dev           | pg_automv            |            1 | local       |
          |                 |
```

```
 dev           | pg_catalog            |             1 | local        | jpuser=UC/
jpuser~=U/jpuser  |                  |
 dev           | public                |             1 | local        | jpuser=UC/
jpuser~=UC/jpuser |                  |
 dev           | information_schema    |             1 | local        | jpuser=UC/
jpuser~=U/jpuser  |                  |
 dev           | schemad79cd6d93bf043 |             1 | local        |
          |                  |
```

次の例は、awsdatacatalog という AWS Glue Data Catalog データベース内のスキーマを表示します。出力行の最大数は 5 です。

```
SHOW SCHEMAS FROM DATABASE awsdatacatalog LIMIT 5;

 database_name  |     schema_name      | schema_owner | schema_type | schema_acl |
 source_database | schema_option
----------------+----------------------+--------------+-------------+------------
+-----------------+---------------
 awsdatacatalog | 000_too_many_glue_db |              | EXTERNAL    |            |
           |
 awsdatacatalog | 123_default          |              | EXTERNAL    |            |
           |
 awsdatacatalog | adhoc                |              | EXTERNAL    |            |
           |
 awsdatacatalog | all_shapes_10mb      |              | EXTERNAL    |            |
           |
 awsdatacatalog | all_shapes_1g        |              | EXTERNAL    |            |
           |
```

テーブルを表示する

テーブル属性、テーブル制約、列属性、列制約など、テーブルの定義を表示します。SHOW TABLE ステートメントの出力を使用して、テーブルを再作成できます。

テーブル作成の詳細については、「CREATE TABLE」を参照してください。

構文

```
SHOW TABLE [schema_name.]table_name
```

パラメータ

schema_name

(オプション) 関連するスキーマの名前。

table_name

表示するテーブルの名前。

例

以下は、テーブル `sales` の SHOW TABLE 出力の例です。

```
show table sales;
```

```
CREATE TABLE public.sales (
salesid integer NOT NULL ENCODE az64,
listid integer NOT NULL ENCODE az64 distkey,
sellerid integer NOT NULL ENCODE az64,
buyerid integer NOT NULL ENCODE az64,
eventid integer NOT NULL ENCODE az64,
dateid smallint NOT NULL,
qtysold smallint NOT NULL ENCODE az64,
pricepaid numeric(8,2) ENCODE az64,
commission numeric(8,2) ENCODE az64,
saletime timestamp without time zone ENCODE az64
)
DISTSTYLE KEY SORTKEY ( dateid );
```

以下は、スキーマ `public` のテーブル `category` の SHOW TABLE 出力の例です。

```
show table public.category;
```

```
CREATE TABLE public.category (
catid smallint NOT NULL distkey,
catgroup character varying(10) ENCODE lzo,
catname character varying(10) ENCODE lzo,
catdesc character varying(50) ENCODE lzo
) DISTSTYLE KEY SORTKEY ( catid );
```

次の例では、プライマリキーを使用しながらテーブル foo を作成します。

```
create table foo(a int PRIMARY KEY, b int);
```

SHOW TABLE の結果は、fooテーブルの作成ステートメントを、すべてのプロパティとともに表示します。

```
show table foo;
```

```
CREATE TABLE public.foo ( a integer NOT NULL ENCODE az64, b integer ENCODE az64,
 PRIMARY KEY (a) ) DISTSTYLE AUTO;
```

SHOW TABLES

スキーマ内のテーブルのリストと、いくつかのテーブル属性を表示します。

各出力行は、データベース名、スキーマ名、テーブル名、テーブルタイプ、テーブル ACL、および注釈で構成されます。これらの属性の詳細については、「SVV_ALL_TABLES」を参照してください。

SHOW TABLES コマンドの結果のテーブル数が 10,000 を超える場合は、エラーが返されます。

構文

```
SHOW TABLES FROM SCHEMA database_name.schema_name [LIKE 'filter_pattern']
 [LIMIT row_limit ]
```

パラメータ

database_name

一覧表示するテーブルが含まれているデータベースの名前。

AWS Glue Data Catalog でテーブルを表示するには、データベース名として (awsdatacatalog) を指定し、システム設定 data_catalog_auto_mount が true に設定されていることを確認します。詳細については、「ALTER SYSTEM」を参照してください。

schema_name

一覧表示するテーブルが含まれているスキーマの名前。

AWS Glue Data Catalog テーブルを表示するには、スキーマ名として AWS Glue データベース名を指定します。

filter_pattern

テーブル名とマッチングするパターンが含まれる有効な UTF-8 文字式。LIKE オプションでは、以下のパターンマッチングメタ文字をサポートする、大文字と小文字を区別したマッチングを行います。

メタ文字	説明
%	ゼロ個以上の任意の文字シーケンスをマッチングします。
_	任意の 1 文字をマッチングします。

filter_pattern にメタ文字が含まれていない場合、パターンは文字列そのものを表すだけです。この場合、LIKE は等号演算子と同じ働きをします。

row_limit

返される行の最大数。row_limit には、0～10,000 の値を指定できます。

例

次の例は、スキーマ public にある、dev という名前の Amazon Redshift データベース内のテーブルを示しています。

```
SHOW TABLES FROM SCHEMA dev.public;

 database_name | schema_name | table_name | table_type | table_acl | remarks
---------------+-------------+------------+------------+-----------+---------
 dev           | public      | tb         | TABLE      |           |
 dev           | public      | tb2        | TABLE      |           |
 dev           | public      | tb3        | TABLE      |           |
```

次の例は、スキーマ batman にある、awsdatacatalog という AWS Glue Data Catalog データベース内のテーブルを示しています。

```
SHOW TABLES FROM SCHEMA awsdatacatalog.batman;
```

```
 database_name  | schema_name |    table_name    | table_type | table_acl | remarks
----------------+-------------+------------------+------------+-----------+---------
 awsdatacatalog | batman      | nation           | EXTERNAL   |           |
 awsdatacatalog | batman      | part             | EXTERNAL   |           |
 awsdatacatalog | batman      | partsupp         | EXTERNAL   |           |
 awsdatacatalog | batman      | region           | EXTERNAL   |           |
 awsdatacatalog | batman      | supplier         | EXTERNAL   |           |
 awsdatacatalog | batman      | automount_nation | EXTERNAL   |           |
```

ビューを表示する

マテリアライズドビューと遅延バインドビューを含む、ビューの定義を表示します。SHOW VIEW ステートメントの出力を使用して、ビューを再作成できます。

構文

```
SHOW VIEW [schema_name.]view_name
```

パラメータ

schema_name

(オプション) 関連するスキーマの名前。

view_name

表示するビューの名前。

例

以下は、ビュー LA_Venues_v のビュー定義です。

```
create view LA_Venues_v as select * from venue where venuecity='Los Angeles';
```

次に、前に定義したビューの SHOW VIEW コマンドと出力の例を示します。

```
show view LA_Venues_v;
```

```
SELECT venue.venueid,
venue.venuename,
venue.venuecity,
```

```
venue.venuestate,
venue.venueseats
FROM venue WHERE ((venue.venuecity)::text = 'Los Angeles'::text);
```

以下は、スキーマ public のビュー public.Sports_v のビュー定義です。

```
create view public.Sports_v as select * from category where catgroup='Sports';
```

次に、前に定義したビューの SHOW VIEW コマンドと出力の例を示します。

```
show view public.Sports_v;
```

```
SELECT category.catid,
category.catgroup,
category.catname,
category.catdesc
FROM category WHERE ((category.catgroup)::text = 'Sports'::text);
```

START TRANSACTION

BEGIN 関数の同義語です。

「BEGIN」を参照してください。

TRUNCATE

テーブルをスキャンせずに、テーブルからすべての行を削除します。この操作は無条件の DELETE 操作に対する代替案ですが、より高速です。TRUNCATE コマンドを実行するには、TRUNCATE TABLE アクセス許可を付与されている、テーブルの所有者である、またはスーパーユーザーである必要があります。テーブルを切り捨てるアクセス許可を付与するには、GRANT コマンドを使用します。

TRUNCATE は DELETE よりもはるかに効率的であり、VACUUM および ANALYZE を必要としません。ただし、TRUNCATE では、その操作を実行するトランザクションがコミットされることに注意してください。

構文

```
TRUNCATE [ TABLE ] table_name
```

このコマンドはマテリアライズドビューでも機能します。

```
TRUNCATE materialized_view_name
```

パラメータ

TABLE

オプションキーワード

table_name

一時テーブルまたは永続的テーブル テーブルの所有者またはスーパーバイザだけがテーブルを切り取ることができます。

外部キー拘束で参照されるテーブルなど、任意のテーブルを切り取ることができます。

テーブルを切り取った後、テーブルに対して VACUUM を実行する必要はありません。

materialized_view_name

マテリアライズドビュー。

マテリアライズドビューへのストリーミング取り込み に使用されているマテリアライズドビューは切り捨てることができます。

使用に関する注意事項

TRUNCATE コマンドはコマンドが実行されたトランザクションをコミットします。そのため、TRUNCATE 操作をロールバックすることはできず、それ自体がコミットを行うと TRUNCATE コマンドが他の操作にコミットする場合があります。

例

TRUNCATE コマンドを使って、CATEGORY テーブルからすべての行を削除します。

```
truncate category;
```

TRUNCATE オペレーションのロールバック試行。

```
begin;
```

```
truncate date;

rollback;

select count(*) from date;
count
-------
0
(1 row)
```

TRUNCATE コマンドにより自動的に確定したため、ROLLBACK コマンドを実行しても DATE テーブルは空のままです。

次の例では、TRUNCATE コマンドを使用してマテリアライズドビューからすべての行を削除します。

```
truncate my_materialized_view;
```

マテリアライズドビューのすべてのレコードが削除され、マテリアライズドビューとそのスキーマはそのまま残ります。クエリでは、マテリアライズドビュー名はサンプルです。

UNLOAD

Amazon S3 のサーバー側の暗号化 (SSE-S3) を使用して、Amazon S3 上の 1 つ、または複数のテキスト、JSON、または Apache Parquet ファイルにクエリの結果をアンロードします。AWS Key Management Service キーを使用したサーバー側の暗号化 (SSE-KMS)、またはカスタマーマネージド型キーを使用したクライアント側の暗号化を指定することもできます。

デフォルトでは、アンロードされたファイルの形式はパイプ区切り (|) テキストです。

Amazon S3 のファイルは、サイズ、拡張子、ファイル数、MAXFILESIZE パラメータを設定することで管理できます。S3 IP 範囲が許可リストに追加されていることを確認します。必要な S3 IP 範囲の詳細については、「ネットワークの隔離」を参照してください。

分析用の効率的なオープン列型ストレージ形式である Apache Parquet では、Amazon Redshift クエリの結果を Amazon S3 データレイクにアンロードできます。Parquet 形式は、テキスト形式と比較して、アンロードが最大 2 倍速く、さらにストレージ使用量が Amazon S3 で最大 6 倍少なくすみます。これにより、Amazon S3 で行ったデータ変換とエンリッチメントをオープン形式で Amazon S3 データレイクに保存できます。次に、Redshift Spectrum や Amazon Athena、Amazon EMR、Amazon SageMaker などの他の AWS のサービスを使用してデータを分析できます。

UNLOAD コマンドの使用に関する詳細とシナリオ例については、「Amazon Redshift でのデータのアンロード」を参照してください。

必要な権限とアクセス許可

UNLOAD コマンドを正常に実行するには、少なくともデータベース内のデータに対する SELECT 権限と、Amazon S3 ロケーションへの書き込みアクセス許可が必要です。必要なアクセス許可は COPY コマンドと同様です。COPY コマンドのアクセス許可に関する情報は、「他の AWS リソースにアクセスするアクセス許可」を参照してください。

構文

```
UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
authorization
[ option, ...]

where authorization is
IAM_ROLE { default | 'arn:aws:iam::<AWS #####-id-1>:role/<role-name>[,arn:aws:iam::<AWS
 #####-id-2>:role/<role-name>][,...]' }

where option is
| [ FORMAT [ AS ] ] CSV | PARQUET | JSON
| PARTITION BY ( column_name [, ... ] ) [ INCLUDE ]
| MANIFEST [ VERBOSE ]
| HEADER
| DELIMITER [ AS ] 'delimiter-char'
| FIXEDWIDTH [ AS ] 'fixedwidth-spec'
| ENCRYPTED [ AUTO ]
| BZIP2
| GZIP
| ZSTD
| ADDQUOTES
| NULL [ AS ] 'null-string'
| ESCAPE
| ALLOWOVERWRITE
| CLEANPATH
| PARALLEL [ { ON | TRUE } | { OFF | FALSE } ]
| MAXFILESIZE [AS] max-size [ MB | GB ]
| ROWGROUPSIZE [AS] size [ MB | GB ]
| REGION [AS] 'aws-region' }
| EXTENSION 'extension-name'
```

パラメータ

('select-statement')

SELECT クエリ。クエリ結果をアンロードします。ほとんどの場合、クエリで ORDER BY 句を指定してソート順にデータをアンロードすることは有益です。この方法は、データの再ロード時にデータをソートするために必要な時間を節約します。

クエリは、次に示すように一重引用符で囲む必要があります:

```
('select * from venue order by venueid')
```

> **Note**
>
> クエリに引用符が含まれている場合 (リテラル値を囲む場合など)、リテラルを 2 つの単一引用符で囲みます。また、クエリを単一引用符で囲む必要があります。
>
> ```
> ('select * from venue where venuestate=''NV''')
> ```

TO 's3://object-path/name-prefix'

Amazon Redshift が出力ファイルオブジェクト (MANIFEST が指定されている場合はマニフェストファイルを含む) を書き込む Amazon S3 上の絶対パス (バケット名を含む)。オブジェクト名には name-prefix というプレフィックスが付きます。`PARTITION BY` を使用すると、必要に応じて name-prefix 値の末尾にスラッシュ (/) が自動的に追加されます。さらにセキュリティを強化するために、UNLOAD は HTTPS 接続を使用して Amazon S3 に接続します。デフォルトでは、UNLOAD はスライスごとに 1 つ以上のファイルを書き込みます。次に示すように、UNLOAD は指定された名前のプレフィックスにスライス番号とパート番号を追加します:

`<object-path>/<name-prefix><slice-number>_part_<part-number>`.

MANIFEST が指定された場合、マニフェストファイルは次のように書き込まれます:

`<object_path>/<name_prefix>manifest`.

PARALLEL が OFF に指定されている場合、データファイルは次のように書き込まれます。

<object_path>/*<name_prefix><part-number>*.

UNLOAD は、Amazon S3 サーバー側の暗号化 (SSE) を使用して自動的に暗号化ファイルを作成します。MANIFEST を使用すると、マニフェストファイルも暗号化されます。COPY コマンドは、ロード操作中に自動的にサーバー側で暗号化されたファイルを読み取ります。Amazon S3 コンソールまたは API を使用すると、サーバー側で暗号化されたファイルをバケットから透過的にダウンロードできます。詳細については、「[サーバー側の暗号化を使用したデータの保護](#)」を参照してください。

Amazon S3 クライアント側暗号化機能を使用するには、ENCRYPTED オプションを指定します。

> **Important**
>
> REGION は、Amazon S3 バケットが Amazon Redshift データベースと同じ AWS リージョン にない場合に必須です。

authorization

UNLOAD コマンドには Amazon S3 にデータを書き込むための認可が必要です。UNLOAD コマンドは、COPY コマンドが認可に使用するのと同じパラメータを使用します。詳細については、COPY コマンドの構文リファレンスの「[認可パラメータ](#)」を参照してください。

IAM_ROLE { default | 'arn:aws:iam::*<AWS #####-id-1>*:role/*<role-name>*'

デフォルトキーワードを使用して、UNLOAD コマンドの実行時にデフォルトとして設定されクラスターに関連付けられた IAM ロールの使用を、Amazon Redshift に指示します。

クラスターが認証と認可に使用する IAM ロールの Amazon リソースネーム (ARN) を使用します。IAM_ROLE を指定すると、ACCESS_KEY_ID および SECRET_ACCESS_KEY、SESSION_TOKEN、または CREDENTIALS は使用できません。IAM_ROLE は連鎖できます。詳細については、「Amazon Redshift 管理ガイド」の「[IAM ロールを連鎖する](#)」を参照してください。

[FORMAT [AS]] CSV | PARQUET | JSON

デフォルトの形式を上書きするアンロード形式を指定するキーワード。

CSV の場合、デフォルトの区切り文字としてカンマ (,) を使用して CSV 形式でテキストファイルをアンロードします。フィールドに区切り文字、二重引用符、改行文字、またはキャリッジリ

ターンが含まれている場合、アンロードしたファイルのフィールドは二重引用符で囲まれます。データフィールド内の二重引用符は、追加の二重引用符でエスケープされます。ゼロ行がアンロードされると、Amazon Redshift は空の Amazon S3 オブジェクトを書き込む可能性があります。

PARQUET の場合、Apache Parquet バージョン 1.0 形式のファイルにアンロードします。デフォルトでは、各行グループは SNAPPY 圧縮を使用して圧縮されます。Apache Parquet 形式の詳細については、「[Apache Parquet](https://parquet.apache.org/)」を参照してください。

JSON の場合、各行に JSON オブジェクトが含まれている JSON ファイルにアンロードして、クエリ結果の完全なレコードを表します。Amazon Redshift は、クエリ結果に SUPER 列が含まれている場合に、ネストされた JSON の書き込みをサポートします 有効な JSON オブジェクトを作成するには、クエリ内の各列の名前が一意である必要があります。JSON ファイルでは、ブール値が `t` または `f` としてアンロードされ、NULL 値が `null` としてアンロードされます。ゼロ行がアンロードされると、Amazon Redshift は Amazon S3 オブジェクトを書き込まなくなります。

キーワード FORMAT および AS はオプションです。CSV は FIXEDWIDTH または ADDQUOTES と併用できません。DELIMITER、FIXEDWIDTH、ADDQUOTES、ESCAPE、NULL AS、HEADER、GZIP、BZIP2、ZSTD では、PARQUET を使用できません。ENCRYPTED を使用した PARQUET は、AWS Key Management Service キー (SSE-KMS) を使用したサーバー側の暗号化でのみサポートされます。DELIMITER、HEADER、FIXEDWIDTH、ADDQUOTES、ESCAPE、または NULL AS では、JSON を使用できません。

PARTITION BY (column_name [, ...]) [INCLUDE]

アンロードオペレーションのパーティションキーを指定します。UNLOAD は、Apache Hive の規則に従って、パーティションキーの値に基づいて、出力ファイルをパーティションフォルダに自動的に分割します。たとえば、パーティション year 2019 および month September に属する Parquet ファイルには、次のプレフィックス `s3://amzn-s3-demo-bucket/my_prefix/year=2019/month=September/000.parquet` があります。

column_name の値は、アンロードされるクエリ結果の列である必要があります。

PARTITION BY で INCLUDE オプションを指定すると、アンロードされるファイルからパーティション列が削除されません。

Amazon Redshift は、PARTITION BY 句で文字列リテラルをサポートしていません。

MANIFEST [VERBOSE]

UNLOAD プロセスによって作成されたデータファイルの詳細を明示的に一覧表示するマニフェストファイルを作成します。マニフェストは、Amazon S3 に書き込まれた各ファイルの URL をリストする、JSON 形式のテキストファイルです。

MANIFEST が VERBOSE オプションとともに指定されている場合、マニフェストには以下の詳細が含まれます。

- 列名とデータ型、および CHAR、VARCHAR、または NUMERIC データ型の場合は各列のディメンション。CHAR および VARCHAR データ型の場合、ディメンションは長さです。DECIMAL または NUMERIC データ型の場合、ディメンションは精度とスケールです。
- 各ファイルにアンロードされた行数。HEADER オプションが指定されている場合、行数にはヘッダー行が含まれます。
- アンロードされたすべてのファイルの合計ファイルサイズ、およびすべてのファイルにアンロードされた合計行数。HEADER オプションが指定されている場合、行数にはヘッダー行が含まれます。
- 作成者。作成者は常に「Amazon Redshift」です。

VERBOSE は、MANIFEST の後にしか指定できません。

マニフェストファイルは、アンロードファイルと同じ Amazon S3 パスプレフィックスに `<object_path_prefix>manifest` 形式で書き込まれます。たとえば、UNLOAD で Amazon S3 パスプレフィックス '`s3://amzn-s3-demo-bucket/venue_`' が指定されている場合、マニフェストファイルの場所は '`s3://amzn-s3-demo-bucket/venue_manifest`' となります。

HEADER

各出力ファイルの先頭に列名を含むヘッダー行を追加します。CSV、DELIMITER、ADDQUOTES、ESCAPE などのテキスト変換オプションもヘッダー行に適用されます。HEADER は FIXEDWIDTH と併用できません。

DELIMITER AS 'delimiter_character'

パイプ文字 (|)、カンマ (,)、タブ (\t) など、出力ファイルのフィールドを区切るために使用する単一の ASCII 文字を指定します。テキストファイルのデフォルトの区切り文字はパイプ文字です。CSV ファイルのデフォルトの区切り文字はコンマ文字です。AS キーワードはオプションです。DELIMITER は FIXEDWIDTH と併用できません。データに区切り文字が含まれている場合、ESCAPE オプションを指定して、区切り文字をエスケープするか、ADDQUOTES を使って、データを二重引用符で囲む必要があります。代替案として、データ内に含まれていない区切り文字を指定します。

FIXEDWIDTH 'fixedwidth_spec'

区切り文字によって区切られているのではなく、各列の幅が固定長のファイルにデータをアンロードします。fixedwidth_spec は、列の数と列の幅を指定する文字列です。AS キーワードはオプションです。FIXEDWIDTH はデータを切り捨てないため、UNLOAD ステートメントの各列の指定は、少なくともその列の最長エントリの長さと同じになります。fixedwidth_spec の形式を次に示します。

```
'colID1:colWidth1,colID2:colWidth2, ...'
```

FIXEDWIDTH は DELIMITER または HEADER と併用できません。

ENCRYPTED [AUTO]

Amazon S3 上の出力ファイルは、Amazon S3 サーバー側の暗号化機能またはクライアント側の暗号化機能を使って暗号化されるよう指定します。MANIFEST を指定すると、マニフェストファイルも暗号化されます。詳細については、「暗号化されたデータファイルをアンロードする」を参照してください。ENCRYPTED パラメータを指定しない場合、UNLOAD は AWS 管理の暗号化キーによる Amazon S3 のサーバー側暗号化 (SSE-S3) 機能を使用して、暗号化されたファイルを自動的に作成します。

ENCRYPTED を指定すると、AWS KMSキーによるサーバー側の暗号化 (SSE-KMS) 機能を使用して Amazon S3 へのアンロードを実行することも可能です。その場合は、KMS_KEY_IDパラメータを使用してキー ID を指定します。CREDENTIALS パラメータを KMS_KEY_ID パラメータと併用することはできません。KMS_KEY_ID を使用してデータに対して UNLOAD コマンドを実行すると、キーを指定せずに同じデータに対して COPY オペレーションを実行できます。

お客様が提供した対称キーによるクライアント側の暗号化を使用して Amazon S3 にアンロードするには、以下の 2 つの方法のいずれかでキーを指定します。キーを提供するには、MASTER_SYMMETRIC_KEYパラメータまたは CREDENTIALS 認証情報文字列の `master_symmetric_key` 部分を使用します。ルート対称キーを使用してデータをアンロードする場合は、暗号化されたデータに対して COPY オペレーションを実行する際にも、必ず同じキーを指定します。

UNLOAD では、お客様が用意したキー (SSE-C) による Amazon S3 サーバー側の暗号化はサポートされません。

ENCRYPTED AUTO が使用されている場合、UNLOAD コマンドは、ターゲット Amazon S3 バケットのプロパティ上にあるデフォルトの AWS KMS 暗号化キーを取得し、Amazon S3 に書

き込まれたファイルに対し AWS KMS キーによる暗号化を行います。バケットにデフォルトの AWS KMS 暗号化キーがない場合、UNLOAD は Amazon Redshift サーバー側の暗号化に AWS 管理の暗号化キーを使用 (SSE-S3) して、暗号化されたファイルを自動的に作成します。このオプションは、master_symmetric_key を含む KMS_KEY_ID、MASTER_SYMMETRIC_KEY、または CREDENTIALS では使用できません。

KMS_KEY_ID 'key-id'

Amazon S3 上のデータファイルの暗号化に使用する AWS Key Management Service (AWS KMS) キー の ID を指定します。詳細については、「AWS Key Management Service とは」を参照してください。KMS_KEY_ID を指定する場合、ENCRYPTEDパラメータも指定する必要があります。KMS_KEY_ID を指定する場合、CREDENTIALS パラメータを使用して認証することはできません。代わりに IAM_ROLE または ACCESS_KEY_ID and SECRET_ACCESS_KEY のどちらかを使用します。

MASTER_SYMMETRIC_KEY 'root_key'

Amazon S3 上のデータファイルの暗号化に使用するルート対称キーを指定します。MASTER_SYMMETRIC_KEY を指定する場合、ENCRYPTEDパラメータも指定する必要があります。MASTER_SYMMETRIC_KEY は CREDENTIALS パラメータと併用できません。詳細については、「暗号化されたデータファイルを Amazon S3 からロードする」を参照してください。

BZIP2

スライスごとに、1 つまたは複数の bzip2 圧縮ファイルにデータをアンロードします。生成される各ファイルには、`.bz2`拡張子が付加されます。

GZIP

スライスごとに、1 つまたは複数の gzip 圧縮ファイルにデータをアンロードします。生成される各ファイルには、`.gz`拡張子が付加されます。

ZSTD

スライスごとに、1 つまたは複数の Zstandard 圧縮ファイルにデータをアンロードします。生成される各ファイルには、`.zst`拡張子が付加されます。

ADDQUOTES

アンロードされた各データフィールドは引用符で囲まれるため、Amazon Redshift は区切り文字自体を含んでいるデータ値をアンロードすることができます。例えば、区切りがカンマの場合、次のデータのアンロードと再ロードを完了することができます。

```
"1","Hello, World"
```

追加した引用符がない場合、文字列 `Hello, World` は 2 つの異なるフィールドとして解析されます。

一部の出力形式は ADDQUOTES をサポートしていません。

ADDQUOTES を使用した際にデータを再ロードするには、COPY で REMOVEQUOTES を指定する必要があります。

NULL AS 'null-string'

アンロードファイル内で NULL 値を表す文字列を指定します。このオプションを使用すると、選択したデータ内で検出された任意の NULL 値の代わりに、指定した文字列がすべての出力ファイルに挿入されます。このオプションを指定しない場合、NULL 値は次のようにアンロードされます。

- 区切り文字で区切られた出力に対するゼロ長文字列
- 固定幅出力に対するホワイトスペース文字列

固定幅のアンロードに対して NULL 文字列が指定され、出力列の幅が NULL 文字列の幅より小さい場合、次の動作が発生します。

- 空のフィールドは非文字列用の出力です。
- 文字列に対してはエラーがレポートされます。

ユーザー定義の文字列が null 値を表す他のデータ型とは異なり、Amazon Redshift は JSON 形式を使用して SUPER データ列をエクスポートし、JSON 形式で決定されるように null として表します。その結果、SUPER データ列では、UNLOAD コマンドで使用される NULL [AS] オプションを無視します。

ESCAPE

区切られたアンロードファイル内の CHAR と VARCHAR 列の場合、エスケープ文字 (\) が次の文字の前に必ず配置されます。

- ラインフィード: `\n`
- キャリッジリターン: `\r`
- アンロードデータ用に指定された区切り文字。
- エスケープ文字: `\`

• 引用符: " または ' (ESCAPE と ADDQUOTES の両方を UNLOAD コマンドで指定した場合)。

> ⚠ Important
>
> ESCAPE オプションを指定して COPY を使ってデータをロードした場合、UNLOAD コマンドでも ESCAPE オプションを指定して可逆出力ファイルを生成する必要があります。同様に、ESCAPE オプションを使って UNLOAD を実行すると、同じデータを COPY する場合に ESCAPE を使用する必要があります。

ALLOWOVERWRITE

デフォルトでは、UNLOAD によってファイルの上書きが発生する可能性がある場合、その UNLOAD 操作は失敗します。ALLOWOVERWRITE が指定された場合、UNLOAD によって、マニフェストファイルを含めた既存のファイルが上書きされます。

CLEANPATH

CLEANPATH オプションは、TO 句で指定された Amazon S3 パスにある既存のファイルを削除してから、指定した場所にファイルをアンロードします。

PARTITION BY 句を含めると、既存のファイルはパーティションフォルダからのみ削除され、UNLOAD オペレーションによって生成された新しいファイルを受信します。

Amazon S3 バケットでの `s3:DeleteObject` 許可が必要です。詳細については、Amazon Simple Storage Service ユーザーガイドの Amazon S3 でのポリシーとアクセス許可を参照してください。CLEANPATH オプションを使用して削除したファイルは、完全に削除され、復元することはできません。ターゲット Amazon S3 バケットでバージョニングが有効になっている場合、CLEANPATH オプションを使用した UNLOAD では、以前のバージョンのファイルは削除されません。

ALLOWOVERWRITE オプションを指定した場合、CLEANPATH オプションを指定することはできません。

PARALLEL

デフォルトでは、UNLOAD は、クラスター内のスライスの数に応じて、データを複数のファイルに同時に書き込みます。デフォルトのオプションは ON または TRUE です。PARALLEL が OFF または FALSE の場合、UNLOAD は、ORDER BY 句が使用される場合はそれに従って絶対的にソートされた 1 つ以上のデータファイルに逐次書き込みます。データファイルの最大サイズは

6.2 GB です。したがって、例えば、13.4 GB のデータをアンロードする場合、UNLOAD は以下の 3 つのファイルを作成します。

```
s3://amzn-s3-demo-bucket/key000    6.2 GB
s3://amzn-s3-demo-bucket/key001    6.2 GB
s3://amzn-s3-demo-bucket/key002    1.0 GB
```

> **Note**
>
> UNLOAD コマンドは、並列処理を使用するように設計されています。特別な理由がなく、特にファイルが COPY コマンドを使用してテーブルをロードするために使用される場合には、PARALLEL を有効にしたままにすることをお勧めします。

MAXFILESIZE [AS] 最大サイズ [MB | GB]

UNLOAD によって Amazon S3 で作成されるファイルの最大サイズを指定します。5 MB ~ 6.2 GB の十進値を指定します。AS キーワードはオプションです。デフォルト単位は MB です。MAXFILESIZE を指定しない場合、デフォルトの最大ファイルサイズは 6.2 GB です。マニフェストファイルが使用されている場合、このサイズは MAXFILESIZE に影響されません。

ROWGROUPSIZE [AS] サイズ [MB | GB]

行グループのサイズを指定します。大きいサイズを選択すると、行グループの数を減らし、ネットワーク通信量を減らすことができます。32 MBから 128 MB の間の整数値を指定します。AS キーワードはオプションです。デフォルト単位は MB です。

ROWGROUPSIZE が指定されなかった場合、デフォルトのサイズは 32 MB です。このパラメータを使用するには、ストレージ形式が Parquet で、ノードタイプが ra3.4xlarge、ra3.16xlarge、ds2.8xlarge のいずれかである必要があります。

REGION [AS] 'aws-region'

ターゲットの Amazon S3 バケットがある AWS リージョン を指定します。Amazon Redshift データベースと同じ AWS リージョン にない Amazon S3 バケットへの UNLOAD には、REGION が必要です。

aws_region の値は、「AWS 全般のリファレンス」の「Amazon Redshift のリージョンとエンドポイント」の表に記載されている AWS リージョンと一致している必要があります。

デフォルトでは、UNLOAD は、ターゲットの Amazon S3 バケットは、Amazon Redshift データベースと同じ AWS リージョン にあると見なします。

拡張子「extension-name」

アンロードされたファイルの名前に付加するファイル拡張子を指定します。Amazon Redshift は検証を実行しないため、指定したファイル拡張子が正しいことを確認する必要があります。GZIP などの圧縮方法を使用している場合でも、拡張子パラメータで `.gz` を指定する必要があります。拡張子を指定しない場合、Amazon Redshift はファイル名に何も追加しません。拡張子を指定せずに圧縮方法を指定した場合、Amazon Redshift は圧縮方法の拡張子のみをファイル名に追加します。

使用に関する注意事項

区切り文字が使われているすべての UNLOAD 操作に対する ESCAPE の使用

区切り文字を使用して UNLOAD する場合、データにはその区切り文字、または ESCAPE オプションの説明に一覧表示されている任意の文字を含めることができます。この場合は、UNLOAD ステートメントで ESCAPE オプションを使用する必要があります。UNLOAD コマンドで ESCAPE オプションを使用しない場合、その後のアンロードされたデータを使用する COPY 操作は失敗する可能性があります。

> **Important**
>
> 必ず UNLOAD ステートメントと COPY ステートメントの両方で ESCAPE オプションを使用することを強くお勧めします。ただし、データに区切り文字が含まれていないこと、またはエスケープが必要となる可能性のあるその他の文字が含まれていないことが確実である場合は除きます。

浮動小数点精度の損失

アンロードと再ロードを連続して実行した浮動小数点データは精度を失っていることがあります。

Limit 句

SELECT クエリは、外部の SELECT で LIMIT 句を使用することはできません。例えば、次の UNLOAD ステートメントは失敗します。

```
unload ('select * from venue limit 10')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

代わりに、次の例のようにネストした LIMIT 句を使用してください。

```
unload ('select * from venue where venueid in
(select venueid from venue order by venueid desc limit 10)')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole';
```

SELECT…INTO を使ってテーブルを入力するか、LIMIT 句を使って CREATE TABLE AS を実行し、そのテーブルからアンロードすることもできます。

GEOMETRY データ型の列のアップロード

GEOMETRY 列はテキストまたは CSV 形式でのみアップロードできます。FIXEDWIDTH オプションを使用して、GEOMETRY データをアップロードすることはできません。データは、16 進数の Extended Well-Known Binary (EWKB) 形式でアップロードされます。EWKB データのサイズが 4 MB 以上の場合、後でテーブルにデータをロードできなくなるため、警告が表示されます。

HLLSKETCH データ型のアンロード

HLLSKETCH 列はテキストまたは CSV 形式でのみアンロードできます。FIXEDWIDTH オプションを使用して、HLLSKETCH データをアップロードすることはできません。データは、デンスの HyperLogLog スケッチの場合は Base64 形式、スパースの HyperLogLog スケッチの場合は JSON 形式でアンロードされます。詳細については、「HyperLogLog 関数」を参照してください。

次の例では、HLLSKETCH 列を含むテーブルをファイルにエクスポートします。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

UNLOAD ('select * from hll_table') TO 's3://amzn-s3-demo-bucket/unload/'
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole' NULL AS 'null' ALLOWOVERWRITE
 CSV;
```

VARBYTE データ型の列のアンロード

VARBYTE列はテキストまたは CSV 形式でのみアンロードできます。データは 16 進数形式でアンロードされます。FIXEDWIDTH オプションを使用して、VARBYTE データをアンロードするこ

とはできません。CSV に UNLOAD するための `ADDQUOTES` オプションはサポートされていません。VARBYTE 列は、PARTITIONED BY 列にすることはできません。

FORMAT AS PARQUET 句

FORMAT AS PARQUET を使用する場合は、次の考慮事項に注意してください。

- [Unload to Parquet (Parquet にアンロード)] では、ファイルレベルの圧縮は使用されません。各行グループは SNAPPY で圧縮されます。
- MAXFILESIZE を指定しない場合、デフォルトの最大ファイルサイズは 6.2 GB です。MAXFILESIZE を使用して、5 MB～6.2 GB のファイルサイズを指定できます。実際のファイルサイズは、ファイルの書き込み時に概算されるため、指定した数と正確に等しくない場合があります。

  スキャンパフォーマンスを最大化するため、Amazon Redshift はまったく同じサイズの 32 MB の行グループを含む Parquet ファイルの作成を試みます。指定した MAXFILESIZE 値は、32 MB の最も近い倍数に自動的に切り捨てられます。例えば、MAXFILESIZE 200 MB を指定すると、アンロードされた各 Parquet ファイルは約 192 MB になります (32 MB の行グループ x 6 = 192 MB)。
- 列で TIMESTAMPTZ データ形式が使用されている場合、タイムスタンプ値のみがアンロードされます。タイムゾーン情報はアンロードされません。
- アンダースコア (_) 文字またはピリオド (.) 文字で始まるファイル名プレフィックスを指定しないでください。Redshift Spectrum は、これらの文字で始まるファイルを隠しファイルとして処理し、無視します。

PARTITION BY 句

PARTITION BY を使用する場合は、次の考慮事項に注意してください。

- パーティション列は出力ファイルに含まれていません。
- 必ず、UNLOAD ステートメントで使用される SELECT クエリにパーティション列を含めてください。UNLOAD コマンドでは、任意の数のパーティション列を指定できます。ただし、ファイルの一部となる非パーティション列が少なくとも 1 つ存在する必要があるという制限があります。
- パーティションキー値が null の場合、Amazon Redshift はそのデータをデフォルトパーティション (`partition_column=__HIVE_DEFAULT_PARTITION__`) に自動的にアンロードします。
- UNLOAD コマンドでは、外部カタログへの呼び出しは行われません。新しいパーティションを既存の外部テーブルの一部として登録するには、別個の ALTER TABLE ...。ADD PARTITION ... コマンドを使用します。または、CREATE EXTERNAL TABLE コマンドを実行して、アンロードされ

たデータを新しい外部テーブルとして登録することもできます。また、AWS Glueクローラを使用して、データカタログにデータを入力することもできます。詳細については、AWS Glueデベロッパーガイドの「クローラの定義」を参照してください。

- MANIFEST オプションを使用する場合、Amazon Redshift はルート Amazon S3 フォルダにマニフェストファイルを 1 つだけ生成します。
- パーティションキーとして使用できる列のデータ型は、SMALLINT、INTEGER、BIGINT、DECIMAL、REAL、BOOLEAN、CHAR、VARCHAR、DATE および TIMESTAMP です。

ASSUMEROLE 権限を使用して、UNLOAD オペレーションの IAM ロールへのアクセスを許可する

特定のユーザーおよびグループに UNLOAD オペレーション用の IAM ロールへのアクセスを提供するために、スーパーユーザーは IAM ロールに対する ASSUMEROLE 権限をユーザーおよびグループに付与できます。詳細については、GRANTを参照してください。

UNLOAD は、Amazon S3 アクセスポイントのエイリアスをサポートしていません

UNLOAD コマンドで、Amazon S3 アクセスポイントのエイリアスを使用することはできません。

例

UNLOAD コマンドの使用方法の例については、「UNLOAD の例」を参照してください。

UNLOAD の例

これらの例は、UNLOAD コマンドのさまざまなパラメータを示しています。多くの例では、TICKIT サンプルデータを使用しています。詳細については、「サンプルデータベース」を参照してください。

> **Note**
>
> 次の例では読みやすくするため、改行しています。credentials-args 文字列には改行やスペースを含めないでください。

パイプ区切りファイルへの VENUE のアンロード (デフォルト区切り文字)

次の例は、VENUE テーブルをアンロードし、データを `s3://amzn-s3-demo-bucket/unload/` に書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

デフォルトでは、UNLOAD はスライスごとに 1 つ以上のファイルを書き込みます。ノードごとに 2 つのスライスを装備した 2 ノードクラスターを想定すると、前の例では `amzn-s3-demo-bucket` に以下のファイルが作成されます。

```
unload/0000_part_00
unload/0001_part_00
unload/0002_part_00
unload/0003_part_00
```

出力ファイルの違いをわかりやすくするため、ロケーションにプレフィックスを含めることができます。次の例は、VENUE テーブルをアンロードし、データを `s3://amzn-s3-demo-bucket/unload/venue_pipe_` に書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

結果として、`unload`フォルダに以下の 4 つのファイルが生成されます。ここでも 4 つのスライスを想定しています。

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

パーティション化された Parquet ファイルへの LINEITEM テーブルのアンロード

次の例では、`l_shipdate`列によって分割された Parquet 形式で LINEITEM テーブルをアンロードします。

```
unload ('select * from lineitem')
to 's3://amzn-s3-demo-bucket/lineitem/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
PARQUET
PARTITION BY (l_shipdate);
```

4 つのスライスがある場合、生成される Parquet ファイルはさまざまなフォルダに動的に分割されます。

```
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-02/0000_part_00.parquet
                                                      0001_part_00.parquet
                                                      0002_part_00.parquet
                                                      0003_part_00.parquet
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-03/0000_part_00.parquet
                                                      0001_part_00.parquet
                                                      0002_part_00.parquet
                                                      0003_part_00.parquet
s3://amzn-s3-demo-bucket/lineitem/l_shipdate=1992-01-04/0000_part_00.parquet
                                                      0001_part_00.parquet
                                                      0002_part_00.parquet
                                                      0003_part_00.parquet
...
```

> **Note**
>
> 場合によっては、次の SQL ステートメントに示すように、UNLOAD コマンドで INCLUDE オプションを使用します。
>
> ```
> unload ('select * from lineitem')
> to 's3://amzn-s3-demo-bucket/lineitem/'
> iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
> PARQUET
> PARTITION BY (l_shipdate) INCLUDE;
> ```
>
> この場合、`l_shipdate`列は Parquet ファイルのデータにも含まれます。INCLUDE オプションを使用しない場合、`l_shipdate`列のデータは Parquet ファイルに含まれません。

VENUE テーブルを JSON ファイルにアンロードする

次の例は、VENUE テーブルをアンロードし、そのデータを JSON 形式で `s3://amzn-s3-demo-bucket/unload/` に書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
JSON;
```

VENUE テーブルのサンプル行を次に示します。

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

JSON へのアンロード後、ファイルの形式は次のようになります。

```
{"venueid":1,"venuename":"Pinewood
 Racetrack","venuecity":"Akron","venuestate":"OH","venueseats":0}
{"venueid":2,"venuename":"Columbus \"Crew\" Stadium
 ","venuecity":"Columbus","venuestate":"OH","venueseats":0}
{"venueid":4,"venuename":"Community, Ballpark, Arena","venuecity":"Kansas
 City","venuestate":"KS","venueseats":0}
```

CSV ファイルへの VENUE のアップロード

次の例は、VENUE テーブルをアンロードし、データを CSV 形式で s3://amzn-s3-demo-bucket/unload/ に書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV;
```

VENUE テーブルに次の行が含まれているとします。

```
venueid | venuename                  | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-----------
      1 | Pinewood Racetrack         | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

アンロードしたファイルは次のようになります。

```
1,Pinewood Racetrack,Akron,OH,0
2,"Columbus ""Crew"" Stadium",Columbus,OH,0
4,"Community, Ballpark, Arena",Kansas City,KS,0
```

区切り文字を使用した CSV ファイルへの VENUE のアンロード

次の例では、パイプ文字 (|) を区切り文字として使用して VENUE テーブルをアンロードし、データを CSV 形式で書き込みます。アンロードしたファイルは、`s3://amzn-s3-demo-bucket/unload/`に書き込まれます。この例の VENUE テーブルには、最初の行 (`Pinewood Race|track`) の値にパイプ文字が含まれています。これは、結果の値が二重引用符で囲まれることを示すためにそうされています。二重引用符がある場合は二重引用符でエスケープされ、フィールド全体が二重引用符で囲まれます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
CSV DELIMITER AS '|';
```

VENUE テーブルに次の行が含まれているとします。

```

venueid | venuename                   | venuecity       | venuestate | venueseats
--------+----------------------------+-----------------+------------+-------------
      1 | Pinewood Race|track        | Akron           | OH         | 0
      2 | Columbus "Crew" Stadium    | Columbus        | OH         | 0
      4 | Community, Ballpark, Arena | Kansas City     | KS         | 0
```

アンロードしたファイルは次のようになります。

```
1|"Pinewood Race|track"|Akron|OH|0
2|"Columbus ""Crew"" Stadium"|Columbus|OH|0
4|Community, Ballpark, Arena|Kansas City|KS|0
```

マニフェストファイルを使用した VENUE のアンロード

マニフェストファイルを作成するには、MANIFEST オプションを指定します。次の例では、VENUE テーブルをアンロードし、データファイルとともにマニフェストファイルを s3://amzn-s3-demo-bucket/venue_pipe_ に書き込みます。

> **Important**
>
> MANIFEST オプションを指定してファイルをアンロードする場合、そのファイルをロードするときに COPY コマンドで MANIFEST オプションを使用する必要があります。同じプレフィックスを使用してファイルをロードし、MANIFEST オプションを指定しない場合、COPY ではマニフェストファイルがデータファイルであると推測され、操作が失敗します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role 'arn:aws:iam::0123456789012:role/
MyRedshiftRole'
manifest;
```

結果は次の 5 ファイルです。

```
s3://amzn-s3-demo-bucket/venue_pipe_0000_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0001_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0002_part_00
s3://amzn-s3-demo-bucket/venue_pipe_0003_part_00
s3://amzn-s3-demo-bucket/venue_pipe_manifest
```

マニフェストファイルの内容を次に示します。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0000_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0001_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0002_part_00"},
    {"url":"s3://amzn-s3-demo-bucket/tickit/venue_0003_part_00"}
  ]
}
```

MANIFEST VERBOSE を使用した VENUE のアンロード

MANIFEST VERBOSE オプションを指定すると、マニフェストファイルに以下のセクションが含まれます。

- `entries` セクションには、各ファイルの Amazon S3 パス、ファイルサイズ、および行数が一覧表示されます。

- schema セクションには、列名、データ型、および各列のディメンションが一覧表示されます。
- meta セクションには、すべてのファイルの合計ファイルサイズと行数が表示されます。

次の例では、MANIFEST VERBOSE オプションを使用して VENUE テーブルをアンロードします。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload_venue_folder/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest verbose;
```

マニフェストファイルの内容を次に示します。


```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0000_part_00", "meta":
 { "content_length": 32295, "record_count": 10 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0001_part_00", "meta":
 { "content_length": 32771, "record_count": 20 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0002_part_00", "meta":
 { "content_length": 32302, "record_count": 10 }},
    {"url":"s3://amzn-s3-demo-bucket/venue_pipe_0003_part_00", "meta":
 { "content_length": 31810, "record_count": 15 }}
  ],
  "schema": {
    "elements": [
      {"name": "venueid", "type": { "base": "integer" }},
      {"name": "venuename", "type": { "base": "character varying", 25 }},
      {"name": "venuecity", "type": { "base": "character varying", 25 }},
      {"name": "venuestate", "type": { "base": "character varying", 25 }},
      {"name": "venueseats", "type": { "base": "character varying", 25 }}
    ]
  },
  "meta": {
    "content_length": 129178,
    "record_count": 55
  },
  "author": {
    "name": "Amazon Redshift",
    "version": "1.0.0"
  }
}
```

ヘッダーを使用した VENUE のアンロード

次の例では、ヘッダー行を指定して VENUE をアンロードします。

```
unload ('select * from venue where venueseats > 75000')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
header
parallel off;
```

以下に、出力ファイルの内容とヘッダー行を示します。

```
venueid|venuename|venuecity|venuestate|venueseats
6|New York Giants Stadium|East Rutherford|NJ|80242
78|INVESCO Field|Denver|CO|76125
83|FedExField|Landover|MD|91704
79|Arrowhead Stadium|Kansas City|MO|79451
```

より小さいファイルへの VENUE のアンロード

デフォルトでは、最大のファイルサイズは 6.2 GB です。アンロードするデータが 6.2 GB より大きい場合、UNLOAD は 6.2 GB のデータセグメントごとに新しいファイルを作成します。より小さなファイルを作成するには、MAXFILESIZE パラメータを含めます。前の例のデータのサイズを 20 GB とすると、次の UNLOAD コマンドは 各 1 GB サイズの 20 ファイルを作成します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
maxfilesize 1 gb;
```

VENUE の逐次アンロード

逐次アンロードを行うには、PARALLEL を OFF にします。これにより、UNLOAD が一度に書き込むファイルの数は 1 つになります (ファイルにつき最大で 6.2 GB まで)。

次の例は、VENUE テーブルをアンロードし、データを `s3://amzn-s3-demo-bucket/unload/` に逐次書き込みます。

```
unload ('select * from venue')
```

```
to 's3://amzn-s3-demo-bucket/unload/venue_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off;
```

結果として、venue_serial_000 という名前のファイルが 1 つ生成されます。

アンロードするデータが 6.2 GB より大きい場合、UNLOAD は 6.2 GB のデータセグメントごとに新しいファイルを作成します。次の例は、LINEORDER テーブルをアンロードし、データを `s3://amzn-s3-demo-bucket/unload/` に逐次書き込みます。

```
unload ('select * from lineorder')
to 's3://amzn-s3-demo-bucket/unload/lineorder_serial_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
parallel off gzip;
```

結果として、以下の一連のファイルが生成されます。

```
lineorder_serial_0000.gz
lineorder_serial_0001.gz
lineorder_serial_0002.gz
lineorder_serial_0003.gz
```

出力ファイルの違いをわかりやすくするため、ロケーションにプレフィックスを含めることができます。次の例は、VENUE テーブルをアンロードし、データを `s3://amzn-s3-demo-bucket/venue_pipe_` に書き込みます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/unload/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

結果として、`unload`フォルダに以下の 4 つのファイルが生成されます。ここでも 4 つのスライスを想定しています。

```
venue_pipe_0000_part_00
venue_pipe_0001_part_00
venue_pipe_0002_part_00
venue_pipe_0003_part_00
```

アンロードファイルからの VENUE のロード

一連のアンロードファイルからテーブルをロードするには、COPY コマンドを使用して単純にプロセスを逆順で実行します。次の例では、新しいテーブル LOADVENUE を作成し、前の例で作成したデータファイルからテーブルをロードします。

```
create table loadvenue (like venue);

copy loadvenue from 's3://amzn-s3-demo-bucket/venue_pipe_' iam_role
 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

アンロードファイルとともにマニフェストファイルを作成するときに MANIFEST オプションを使用した場合は、同じマニフェストファイルを使用してデータをロードできます。これには、COPY コマンドで MANIFEST オプションを指定します。次の例では、マニフェストファイルを使用してデータをロードします。

```
copy loadvenue
from 's3://amzn-s3-demo-bucket/venue_pipe_manifest' iam_role
 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

暗号化ファイルへの VENUE のアンロード

次の例では、AWS KMSキーを使用して、VENUE テーブルが一連の暗号化ファイルにアンロードされます。ENCRYPTED オプションでマニフェストファイルを指定すると、マニフェストファイルも暗号化されます。詳細については、「暗号化されたデータファイルをアンロードする」を参照してください。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_encrypt_kms'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
kms_key_id '1234abcd-12ab-34cd-56ef-1234567890ab'
manifest
encrypted;
```

次の例では、ルート対称キーを使用して VENUE テーブルを一連の暗号化ファイルにアンロードします。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_encrypt_cmk'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
encrypted;
```

暗号化ファイルからの VENUE のロード

UNLOAD を ENCRYPT オプションとともに使用して作成された一連のファイルからテーブルをロードするには、COPY コマンドを使用してプロセスを逆順で実行します。同時にこのコマンドでは、ENCRYPTED オプションを使用ながら、UNLOAD コマンドに使用されたものと同じルート対称キーを指定します。次の例では、前の例で作成した暗号化されたデータファイルから LOADVENUE テーブルをロードします。

```
create table loadvenue (like venue);

copy loadvenue
from 's3://amzn-s3-demo-bucket/venue_encrypt_manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
master_symmetric_key 'EXAMPLEMASTERKEYtkbjk/OpCwtYSx/M4/t7DMCDIK722'
manifest
encrypted;
```

タブ区切りファイルへの VENUE データのアンロード

```
unload ('select venueid, venuename, venueseats from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t';
```

出力データファイルは次のようになります。

```
1 Toyota Park Bridgeview IL 0
2 Columbus Crew Stadium Columbus OH 0
3 RFK Stadium Washington DC 0
4 CommunityAmerica Ballpark Kansas City KS 0
5 Gillette Stadium Foxborough MA 68756
...
```

固定幅のデータファイルへの VENUE のアンロード

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_fw_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
```

```
fixedwidth as 'venueid:3,venuename:39,venuecity:16,venuestate:2,venueseats:6';
```

出力データファイルは次のようになります。

```
1  Toyota Park              Bridgeview  IL0
2  Columbus Crew Stadium    Columbus    OH0
3  RFK Stadium              Washington  DC0
4  CommunityAmerica BallparkKansas City KS0
5  Gillette Stadium         Foxborough  MA68756
...
```

VENUE を一連のタブ区切り GZIP 圧縮ファイルにアンロードします。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter as '\t'
gzip;
```

VENUE を GZIP 圧縮テキストファイルにアンロードする

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_tab_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
extension 'txt.gz'
gzip;
```

区切り文字を含むデータのアンロード

この例では、ADDQUOTES オプションを使って、実際の一部のデータフィールドにカンマが含まれているカンマ区切りデータをアンロードします。

最初に引用符を含むテーブルを作成します。

```
create table location (id int, location char(64));

insert into location values (1,'Phoenix, AZ'),(2,'San Diego, CA'),(3,'Chicago, IL');
```

次に ADDQUOTES オプションを使って、データをアンロードします。

```
unload ('select id, location from location')
```

```
to 's3://amzn-s3-demo-bucket/location_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
delimiter ',' addquotes;
```

アンロードされたデータファイルは次のようになります。

```
1,"Phoenix, AZ"
2,"San Diego, CA"
3,"Chicago, IL"
...
```

結合クエリの結果のアンロード

次の例では、ウィンドウ関数を含む結合クエリの結果をアンロードします。

```
unload ('select venuecity, venuestate, caldate, pricepaid,
sum(pricepaid) over(partition by venuecity, venuestate
order by caldate rows between 3 preceding and 3 following) as winsum
from sales join date on sales.dateid=date.dateid
join event on event.eventid=sales.eventid
join venue on event.venueid=venue.venueid
order by 1,2')
to 's3://amzn-s3-demo-bucket/tickit/winsum'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

出力ファイルは次のようになります。

```
Atlanta|GA|2008-01-04|363.00|1362.00
Atlanta|GA|2008-01-05|233.00|2030.00
Atlanta|GA|2008-01-06|310.00|3135.00
Atlanta|GA|2008-01-08|166.00|8338.00
Atlanta|GA|2008-01-11|268.00|7630.00
...
```

NULL AS を使用したアンロード

UNLOAD では、デフォルトで空の文字列として Null 値を出力します。以下の例では、NULL AS を使用して Null のテキスト文字列を置換する方法を示します。

これらの例では、VENUE テーブルにいくつかの Null 値を追加します。

```
update venue set venuestate = NULL
```

```
where venuecity = 'Cleveland';
```

VENUESTATE が Null である VENUE から選択して、列に Null が格納されていることを確認します。

```
select * from venue where venuestate is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
```

次に、VENUE テーブルに対して NULL AS オプションを指定して UNLOAD を実行し、Null 値をキャラクタ文字列 'fred' で置き換えます。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

次のアンロードファイルの例は、Null 値が fred で置き換えられたことを示しています。この場合、VENUESEATS の値の一部も Null であったため、fredで置き換えられたことが分かります。VENUESEATS のデータ型が整数であっても、UNLOAD によってアンロードファイルの値がテキストに変換され、さらに COPY によって再び整数に戻されます。固定幅ファイルにアンロードする場合、NULL AS の文字列をフィールド幅以下にする必要があります。

```
248|Charles Playhouse|Boston|MA|0
251|Paris Hotel|Las Vegas|NV|fred
258|Tropicana Hotel|Las Vegas|NV|fred
300|Kennedy Center Opera House|Washington|DC|0
306|Lyric Opera House|Baltimore|MD|0
308|Metropolitan Opera|New York City|NY|0
  5|Gillette Stadium|Foxborough|MA|5
 22|Quicken Loans Arena|Cleveland|fred|0
101|Progressive Field|Cleveland|fred|43345
...
```

アンロードファイルからテーブルをロードするには、COPY コマンドで同じ NULL AS オプションを指定します。

> **Note**
>
> NOT NULL として定義された列に Null のロードを試みた場合、COPY コマンドは失敗します。

```
create table loadvenuenulls (like venue);

copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
null as 'fred';
```

列に単なる空の文字列ではなく Null が含まれていることを確認するには、LOADVENUENULLS から選択して Null でフィルタリングします。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename          | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |

...
```

Null を含むテーブルに対してデフォルトの NULL AS の動作を使用して UNLOAD を実行し、次にデフォルトの NULL AS の動作を使用して COPY を実行してデータをテーブルに戻すことができます。ただし、ターゲットテーブルの数値以外のフィールドには、Null ではなく空の文字列が格納されます。デフォルトでは、UNLOAD によって Null が空の文字列 (空白またはゼロ長) に変換されます。COPY によって、数値列では空の文字列が Null に変換されますが、数値以外の列には空の文字列が挿入されます。次の例では、デフォルトの NULL AS 動作を使用して UNLOAD の後で COPY を実行する方法を示します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;
```

```
truncate loadvenuenulls;
copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

この場合、Null でフィルタリングすると、VENUESEATS に Null が含まれていた行のみが表示されます。テーブル (VENUE) で VENUESTATE に Null が含まれていた場合、ターゲットテーブル (LOADVENUENULLS) の VENUESTATE には空の文字列が格納されます。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
     251 | Paris Hotel              | Las Vegas | NV         |
...
```

数値以外の列に空の文字列を Null としてロードするには、EMPTYASNULL オプションまたは BLANKSASNULL オプションを含めます。両方を組み合わせて使用しても問題ありません。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' allowoverwrite;

truncate loadvenuenulls;
copy loadvenuenulls from 's3://amzn-s3-demo-bucket/nulls/'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole' EMPTYASNULL;
```

列に単なる空白または空の文字列ではなく Null が含まれていることを確認するには、LOADVENUENULLS でその列を選択して、Null によるフィルタリングを行います。

```
select * from loadvenuenulls where venuestate is null or venueseats is null;

 venueid |        venuename         | venuecity | venuestate | venueseats
---------+--------------------------+-----------+------------+------------
      72 | Cleveland Browns Stadium | Cleveland |            |      73200
     253 | Mirage Hotel             | Las Vegas | NV         |
     255 | Venetian Hotel           | Las Vegas | NV         |
      22 | Quicken Loans Arena      | Cleveland |            |          0
     101 | Progressive Field        | Cleveland |            |      43345
     251 | Paris Hotel              | Las Vegas | NV         |
```

```
    ...
```

ALLOWOVERWRITE パラメータを使用してアンロード

デフォルトでは、UNLOAD は送り先バケットの既存のファイルを上書きしません。例えば、送り先バケット内のファイルを変更せずに、同じ UNLOAD ステートメントを 2 回実行すると、2 回目の UNLOAD は失敗します。マニフェストファイルを含めて既存のファイルを上書きするには、ALLOWOVERWRITE オプションを指定します。

```
unload ('select * from venue')
to 's3://amzn-s3-demo-bucket/venue_pipe_'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest allowoverwrite;
```

PARALLEL パラメータと MANIFEST パラメータを使用して EVENT テーブルをアンロード

テーブルを並行して UNLOAD し、マニフェストファイルを生成できます。Amazon S3 データファイルはすべて同じレベルで作成され、名前の末尾にパターン `0000_part_00` が付きます。マニフェストファイルはデータファイルと同じフォルダレベルにあり、末尾にテキスト `manifest` が付きます。次の SQL は EVENT テーブルをアンロードし、ベース名 `parallel` でファイルを作成します。

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/parallel'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel on
manifest;
```

Amazon S3 ファイルのリストは次のようになります。

```
 Name                       Last modified                        Size
 parallel0000_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0001_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0002_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 52.1 KB
 parallel0003_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 51.1 KB
 parallel0004_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.6 KB
 parallel0005_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 53.4 KB
 parallel0006_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 54.1 KB
 parallel0007_part_00 -   August 2, 2023, 14:54:39 (UTC-07:00) 55.9 KB
 parallelmanifest       -   August 2, 2023, 14:54:39 (UTC-07:00) 886.0 B
```

parallelmanifest ファイルの内容は次のようになります。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/parallel0000_part_00", "meta": { "content_length":
 53316 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0001_part_00", "meta": { "content_length":
 54704 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0002_part_00", "meta": { "content_length":
 53326 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0003_part_00", "meta": { "content_length":
 52356 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0004_part_00", "meta": { "content_length":
 55933 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0005_part_00", "meta": { "content_length":
 54648 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0006_part_00", "meta": { "content_length":
 55436 }},
    {"url":"s3://amzn-s3-demo-bucket/parallel0007_part_00", "meta": { "content_length":
 57272 }}
  ]
}
```

PARALLEL OFF パラメータと MANIFEST パラメータを使用して EVENT テーブルをアンロード

テーブルを順番にアンロード (PARALLEL OFF) して、マニフェストファイルを生成できます。Amazon S3 データファイルはすべて同じレベルで作成され、名前の末尾にパターン 0000 が付きます。マニフェストファイルはデータファイルと同じフォルダレベルにあり、末尾にテキスト manifest が付きます。

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/serial'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
parallel off
manifest;
```

Amazon S3 ファイルのリストは次のようになります。

```
 Name                      -    Last modified                         Size
 serial0000                -    August 2, 2023, 15:54:39 (UTC-07:00) 426.7 KB
```

```
serialmanifest         -   August 2, 2023, 15:54:39 (UTC-07:00) 120.0 B
```

serialmanifest ファイルの内容は、次のようになります。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/serial000", "meta": { "content_length": 436991 }}
  ]
}
```

PARTITION BY パラメータと MANIFEST パラメータを使用して EVENT テーブルをアンロード

テーブルをパーティションごとにアンロードし、マニフェストファイルを生成できます。Amazon S3 に、子パーティションフォルダを含む新しいフォルダが作成され、子フォルダ内のデータファイルは 0000_par_00 のような名前パターンで作成されます。マニフェストファイルは、manifest という名前の付いた子フォルダと同じフォルダレベルにあります。

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/partition'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
partition by (eventname)
manifest;
```

Amazon S3 ファイルのリストは次のようになります。

```
Name                   Type     Last modified                        Size

partition            Folder
```

partition フォルダ内には、パーティション名が付いた子フォルダとマニフェストファイルがあります。次に示すのは、partition フォルダ内のフォルダリストの最下部であり、次のようになります。

```
Name                   Type      Last modified                        Size

...
```

```
eventname=Zucchero/    Folder
eventname=Zumanity/    Folder
eventname=ZZ Top/      Folder
manifest               -     August 2, 2023, 15:54:39 (UTC-07:00) 467.6 KB
```

eventname=Zucchero/ フォルダには、次のようなデータファイルがあります。

```
Name               Last modified                          Size
0000_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 70.0 B
0001_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 106.0 B
0002_part_00 -   August 2, 2023, 15:59:15 (UTC-07:00) 70.0 B
0004_part_00 -   August 2, 2023, 15:59:17 (UTC-07:00) 141.0 B
0006_part_00 -   August 2, 2023, 15:59:16 (UTC-07:00) 35.0 B
0007_part_00 -   August 2, 2023, 15:59:19 (UTC-07:00) 108.0 B
```

manifest ファイルの内容の最下部は次のようになります。

```
{
  "entries": [
    ...
    {"url":"s3://amzn-s3-demo-bucket/partition/eventname=Zucchero/007_part_00", "meta":
 { "content_length": 108 }},
    {"url":"s3://amzn-s3-demo-bucket/partition/eventname=Zumanity/007_part_00", "meta":
 { "content_length": 72 }}
  ]
}
```

MAXFILESIZE、ROWGROUPSIZE、MANIFEST の各パラメータを使用して EVENT テーブルをアンロード

テーブルを並行して UNLOAD し、マニフェストファイルを生成できます。Amazon S3 データファイルはすべて同じレベルで作成され、名前の末尾にパターン 0000_part_00 が付きます。生成される Parquet データファイルは 256 MB、行グループのサイズは 128 MB に制限されています。マニフェストファイルはデータファイルと同じフォルダレベルにあり、末尾に manifest が付きます。

```
unload ('select * from mytickit1.event')
to 's3://amzn-s3-demo-bucket/eventsize'
iam_role 'arn:aws:iam::123456789012:role/MyRedshiftRole'
```

```
maxfilesize 256 MB
rowgroupsize 128 MB
parallel on
parquet
manifest;
```

Amazon S3 ファイルのリストは次のようになります。

```
Name                            Type      Last modified                           Size
eventsize0000_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.5 KB
eventsize0001_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
eventsize0002_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.4 KB
eventsize0003_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.0 KB
eventsize0004_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.3 KB
eventsize0005_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 24.8 KB
eventsize0006_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.0 KB
eventsize0007_part_00.parquet parquet August 2, 2023, 17:35:21 (UTC-07:00) 25.6 KB
eventsizemanifest                 -       August 2, 2023, 17:35:21 (UTC-07:00) 958.0 B
```

eventsizemanifest ファイルの内容は次のようになります。

```
{
  "entries": [
    {"url":"s3://amzn-s3-demo-bucket/eventsize0000_part_00.parquet", "meta":
 { "content_length": 25130 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0001_part_00.parquet", "meta":
 { "content_length": 25428 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0002_part_00.parquet", "meta":
 { "content_length": 25025 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0003_part_00.parquet", "meta":
 { "content_length": 24554 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0004_part_00.parquet", "meta":
 { "content_length": 25918 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0005_part_00.parquet", "meta":
 { "content_length": 25362 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0006_part_00.parquet", "meta":
 { "content_length": 25647 }},
    {"url":"s3://amzn-s3-demo-bucket/eventsize0007_part_00.parquet", "meta":
 { "content_length": 26256 }}
  ]
```

```
}
```

UPDATE

トピック

- 構文
- パラメータ
- 使用に関する注意事項
- UPDATE ステートメントの例

条件が満たされた場合、1 つまたは複数のテーブル列の値を更新します。

> **Note**
>
> 単一 SQL ステートメントの最大サイズは 16 MB です。

構文

```
[ WITH [RECURSIVE] common_table_expression [, common_table_expression , ...] ]
            UPDATE table_name [ [ AS ] alias ] SET column = { expression | DEFAULT }
 [,...]

[ FROM fromlist ]
[ WHERE condition ]
```

パラメータ

WITH 句

1 つ以上の common-table-expressions を指定する任意の句。「WITH 句」を参照してください。

table_name

一時テーブルまたは永続的テーブル テーブルの所有者またはテーブルに関する UPDATE 権限を持っているユーザーだけが行を更新できます。FROM 句を使用したり、式または条件でテーブルを選択する場合、そのテーブルに関する SELECT 権限を所有している必要があります。ここではテーブルにエイリアスを指定することはできません。ただし、FROM 句内でエイリアスを指定することはできます。

> **Note**
>
> Amazon Redshift Spectrum の外部テーブルは読み込み専用です。外部テーブルを UPDATE することはできません。

alias

ターゲットテーブルの一時的な代替名。エイリアスはオプションです。AS キーワードは常にオプションです。

SET column =

修正する 1 つまたは複数の列。一覧表示されていない列は現在の値を保持します。ターゲット列の仕様にテーブル名を含めないでください。たとえば、`UPDATE tab SET tab.col = 1`は無効です。

expression

指定された列の新しい値を定義する式。

DEFAULT

CREATE TABLE ステートメントで列に割り当てられたデフォルト値を使って、列を更新します。

FROM tablelist

他のテーブルの情報を参照することで、テーブルを更新できます。FROM 句の他のテーブルを一覧表示するか、WHERE 条件の一部としてサブクエリを使用します。FROM 句で一覧表示されているテーブルには、エイリアスを設定することができます。リスト内に UPDATE ステートメントのターゲットテーブルを含める必要がある場合は、エイリアスを使用します。

WHERE condition

条件と一致する行への更新を制限するオプション句。条件が `true` を返した場合、指定された SET 列が更新されます。条件は列に関するシンプルな述語の場合もあれば、サブクエリの結果に基づく条件の場合もあります。

UPDATE のターゲットテーブルなど、サブクエリ内の任意のテーブルを指定できます。

使用に関する注意事項

テーブル内の多数の行を更新した後:

- ストレージ容量を再利用し、行を再ソートするため、テーブルにバキューム処理を実行します。
- テーブルを分析して、クエリプランナーの統計情報を更新します。

Left、right、および full 外部結合は、UPDATE ステートメントの FROM 句ではサポートされず、以下のエラーを返します。

```
ERROR: Target table must be part of an equijoin predicate
```

外部結合を指定する必要がある場合は、UPDATE ステートメントの WHERE 句でサブクエリを使用します。

UPDATE ステートメントでターゲットテーブルへの自己結合が必要な場合、更新操作の対象となる行を限定する WHERE 句の基準だけでなく、結合条件も指定する必要があります。一般的に、ターゲットテーブルを自分または他のテーブルに結合する場合のベストプラクティスは、アップデータ対象の行を限定する基準と、結合条件を明確に分離するサブクエリを使用することです。

行ごとに複数の一致がある UPDATE クエリは、構成パラメータ `error_on_nondeterministic_update` が true に設定されている場合にエラーをスローします。詳細については、「error_on_nondeterministic_update」を参照してください。

GENERATED BY DEFAULT AS IDENTITY 列を更新できます。GENERATED BY DEFAULT AS IDENTITY として定義された列は、指定した値で更新できます。詳細については、「GENERATED BY DEFAULT AS IDENTITY」を参照してください。

UPDATE ステートメントの例

次の例で使用されるテーブルの詳細については、「サンプルデータベース」を参照してください。

TICKIT データベースの CATEGORY テーブルには、次の行が含まれています。

```
+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 5     | Sports   | MLS       | Major League Soccer                        |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 1     | Sports   | MLB       | Major League Baseball                      |
| 6     | Shows    | Musicals  | Musical theatre                            |
| 3     | Sports   | NFL       | National Football League                   |
| 8     | Shows    | Opera     | All opera and light opera                  |
```

```
| 2     | Sports   | NHL       | National Hockey League                      |
| 9     | Concerts | Pop       | All rock and pop music concerts             |
| 4     | Sports   | NBA       | National Basketball Association             |
| 7     | Shows    | Plays     | All non-musical theatre                     |
| 10    | Concerts | Jazz      | All jazz singers and bands                  |
+-------+----------+-----------+---------------------------------------------+
```

値の範囲に基づくテーブルの更新

CATID 列の値の範囲に基づいて、CATGROUP 列を更新します。

```
UPDATE category
SET catgroup='Theatre'
WHERE catid BETWEEN 6 AND 8;

SELECT * FROM category
WHERE catid BETWEEN 6 AND 8;

+-------+----------+----------+---------------------------+
| catid | catgroup | catname  |          catdesc          |
+-------+----------+----------+---------------------------+
| 6     | Theatre  | Musicals | Musical theatre           |
| 7     | Theatre  | Plays    | All non-musical theatre   |
| 8     | Theatre  | Opera    | All opera and light opera |
+-------+----------+----------+---------------------------+
```

現在の値に基づくテーブルの更新

CATNAME 列および CATDESC 列を現在の CATGROUP 値に基づいて更新します。

```
UPDATE category
SET catdesc=default, catname='Shows'
WHERE catgroup='Theatre';

SELECT * FROM category
WHERE catname='Shows';

+-------+----------+---------+---------+
| catid | catgroup | catname | catdesc |
+-------+----------+---------+---------+
| 6     | Theatre  | Shows   | NULL    |
| 7     | Theatre  | Shows   | NULL    |
| 8     | Theatre  | Shows   | NULL    |
```

```
+-------+----------+---------+---------+)
```

この場合、テーブルが作成されたときにデフォルト値が定義されなかったため、CATDESC 列は null に設定されています。

次のコマンドを実行して、CATEGORY テーブルのデータを元の値に設定し直します。

```
TRUNCATE category;

COPY category
FROM 's3://redshift-downloads/tickit/category_pipe.txt'
DELIMITER '|'
IGNOREHEADER 1
REGION 'us-east-1'
IAM_ROLE default;
```

WHERE 句のサブクエリの結果に基づく、テーブルの更新

WHERE 句のサブクエリの結果に基づいて、CATEGORY テーブルを更新します。

```
UPDATE category
SET catdesc='Broadway Musical'
WHERE category.catid IN
(SELECT category.catid FROM category
JOIN event ON category.catid = event.catid
JOIN venue ON venue.venueid = event.venueid
JOIN sales ON sales.eventid = event.eventid
WHERE venuecity='New York City' AND catname='Musicals');
```

更新したテーブルを表示します。

```
SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 6     | Shows    | Musicals  | Broadway Musical                           |
| 7     | Shows    | Plays     | All non-musical theatre                    |
```

```
| 8     | Shows    | Opera     | All opera and light opera                  |
| 9     | Concerts | Pop       | All rock and pop music concerts            |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
+-------+----------+-----------+--------------------------------------------+
```

WITH 句のサブクエリの結果に基づくテーブルの更新

WITH 句を使用するサブクエリの結果に基づいて CATEGORY テーブルを更新するには、次の例を使用します。

```
WITH u1 as (SELECT catid FROM event ORDER BY catid DESC LIMIT 1)
UPDATE category SET catid='200' FROM u1 WHERE u1.catid=category.catid;

SELECT * FROM category ORDER BY catid DESC LIMIT 1;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
| 200   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+---------------------------------+
```

結合条件の結果に基づく、テーブルの更新

EVENT テーブルで一致する CATID 行 に基づいて、CATEGORY テーブルの元の 11 行を更新します。

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid;

SELECT * FROM category ORDER BY catid;

+-------+----------+-----------+--------------------------------------------+
| catid | catgroup |  catname  |                  catdesc                   |
+-------+----------+-----------+--------------------------------------------+
| 2     | Sports   | NHL       | National Hockey League                     |
| 3     | Sports   | NFL       | National Football League                   |
| 4     | Sports   | NBA       | National Basketball Association            |
| 5     | Sports   | MLS       | Major League Soccer                        |
| 10    | Concerts | Jazz      | All jazz singers and bands                 |
| 11    | Concerts | Classical | All symphony, concerto, and choir concerts |
| 100   | Concerts | Pop       | All rock and pop music concerts            |
```

```
| 100   | Shows    | Plays     | All non-musical theatre                      |
| 100   | Shows    | Opera     | All opera and light opera                    |
| 100   | Shows    | Musicals  | Broadway Musical                             |
+-------+----------+-----------+----------------------------------------------+
```

EVENT テーブルは FROM 句にリストされ、ターゲットテーブルへの結合条件は WHERE 句で定義されていることに注意してください。更新用に限定される行は 4 行だけです。これらの 4 行は、CATID 値が元々 6、7、8、および 9 だった行です。この 4 つのカテゴリだけが、EVENT テーブル内で表現されます。

```
SELECT DISTINCT catid FROM event;

+-------+
| catid |
+-------+
| 6     |
| 7     |
| 8     |
| 9     |
+-------+
```

前の例を拡張して、WHERE 句に別の条件を追加することで、CATEGORY テーブル内の元の 11 行を更新します。CATGROUP 列に関する制限により、更新用に限定される行は 1 行だけです (ただし、結合用には 4 行は有資格となります)。

```
UPDATE category SET catid=100
FROM event
WHERE event.catid=category.catid
AND catgroup='Concerts';

SELECT * FROM category WHERE catid=100;

+-------+----------+---------+---------------------------------+
| catid | catgroup | catname |             catdesc             |
+-------+----------+---------+---------------------------------+
| 100   | Concerts | Pop     | All rock and pop music concerts |
+-------+----------+---------+---------------------------------+
```

この例を作成するためのもう 1 つの方法を次に示します。

```
UPDATE category SET catid=100
```

```
FROM event JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
```

この方法のメリットは、結合基準が、更新対象の行を限定する他の規格と明確に分離されることです。FROM 句の CATEGORY テーブルに対するエイリアス CAT の使用に注意してください。

FROM 句内の外部結合を使った更新

前の例では、UPDATE ステートメントの FROM 句で指定した内部結合を示しました。次の例では、FROM 句がターゲットテーブルに対する外部結合をサポートしていないため、エラーが返されます。

```
UPDATE category SET catid=100
FROM event LEFT JOIN category cat ON event.catid=cat.catid
WHERE cat.catgroup='Concerts';
ERROR:  Target table must be part of an equijoin predicate
```

UPDATE ステートメントに対して外部結合が必要な場合、外部結合構文をサブクエリに移動することができます。

```
UPDATE category SET catid=100
FROM
(SELECT event.catid FROM event LEFT JOIN category cat ON event.catid=cat.catid)
 eventcat
WHERE category.catid=eventcat.catid
AND catgroup='Concerts';
```

SET 句内の別のテーブルの列を使用した更新

TICKIT サンプルデータベース内の listing テーブルを sales テーブルの値を使用して更新するには、次の例を使用します。

```
SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;

+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 4          |
| 108334 | 24         |
| 117150 | 4          |
| 135915 | 20         |
```

```
| 205927 | 6          |
+--------+------------+

UPDATE listing
SET numtickets = sales.sellerid
FROM sales
WHERE sales.sellerid = 1 AND listing.sellerid = sales.sellerid;

SELECT listid, numtickets FROM listing WHERE sellerid = 1 ORDER BY 1 ASC LIMIT 5;

+--------+------------+
| listid | numtickets |
+--------+------------+
| 100423 | 1          |
| 108334 | 1          |
| 117150 | 1          |
| 135915 | 1          |
| 205927 | 1          |
+--------+------------+
```

VACUUM

指定されたテーブルまたは現在のデータベース内のすべてのテーブルで、行を再ソートしてスペースを再利用します。

> **Note**
>
> 必要なテーブルのアクセス許可を持つユーザーのみが、テーブルにバキューム処理を効果的に行うことができます。必要なテーブルアクセス許可なしで VACUUM が実行された場合、オペレーションは完了しますが、効果はありません。VACUUM を効果的に実行するのに有効なテーブルアクセス許可のリストについては、「必要な権限」セクションを参照してください。

Amazon Redshift は、背景で自動的にデータをソートし、VACUUM DELETE を実行します。これにより、VACUUM コマンドを実行する必要が少なくなります。詳細については、「テーブルのバキューム処理」を参照してください。

デフォルトではVACUUM コマンドで、テーブルの行の 95 パーセント以上がすでにソートされているテーブルのソートフェーズをスキップします。ソートフェーズをスキップすることにより、VACUUM のパフォーマンスが大幅に向上します。1 つのテーブルのデフォルトのソートある

いは削除しきい値を変更するには、VACUUM を実行するときに、テーブル名および TO threshold PERCENT パラメータを含めます。

ユーザーは、バキューム処理中のテーブルにアクセスできます。バキューム処理中のテーブルにクエリおよび書き込み操作を実行できますが、データ操作言語 (DML) コマンドおよびバキュームを同時に実行すると両方の処理時間が長くなる可能性があります。バキューム処理中に UPDATE および DELETE ステートメントを実行する場合は、システムのパフォーマンスが低減する場合があります。VACUUM DELETE は、更新操作と削除操作を一時的にブロックします。

Amazon Redshift は、バックグラウンドで自動的に DELETE ONLY vacuum を実行します。ユーザーが ALTER TABLE などのデータ定義言語 (DDL) 操作を実行すると、自動バキューム操作は一時停止します。

> **Note**
>
> Amazon Redshift の VACUUM コマンドの構文と動作は、PostgreSQL の VACUUM 操作とは大幅に異なります。例えば、Amazon Redshift でのデフォルトの VACUUM 操作は VACUUM FULL です。これは、ディスク領域を再利用し、すべての行を再ソートします。これに対して、PostgreSQL のデフォルトの VACUUM 操作は、単純に領域を再利用し、再び使用できるようにするだけです。

詳細については、「テーブルのバキューム処理」を参照してください。

必要な権限

以下に、VACUUM に必要な権限を示します。

- スーパーユーザー
- VACUUM の権限を持つユーザー
- テーブルの所有者
- テーブルの共有先であるデータベース所有者

構文

```
VACUUM [ FULL | SORT ONLY | DELETE ONLY | REINDEX | RECLUSTER ]
[ [ table_name ] [ TO threshold PERCENT ] [ BOOST ] ]
```

パラメータ

FULL

指定されたテーブル (または現在のデータベースのすべてのテーブル) をソートし、直前の UPDATE 操作および DELETE 操作で削除対象のマークが付けられた行によって占有されているディスク領域を再利用します。VACUUM FULL がデフォルトです。

完全バキュームは、インターリーブテーブルのインデックスを再作成しません。インターリーブテーブルのインデックスを再作成するには、VACUUM REINDEXオプションを使用します。

デフォルトで、VACUUM FULL は、少なくとも 95 パーセントがソート済みであるテーブルのソートフェーズをすべてスキップします。VACUUM がソートフェーズをスキップできれば、DELETE ONLY を実行し、削除フェーズで残りの行の少なくとも 95 パーセントが削除対象としてマークされていない領域を再利用します。

ソートしきい値に達しておらず (例えば 90 パーセントの行がソートされていて) VACUUM が完全ソートを実行する場合は、完全な削除オペレーションも実行され、削除された行のスペースが 100 パーセント回復されます。

一つのテーブルに対してのみデフォルトの VACUUM しきい値を変更できます。1 つのテーブルのデフォルトの VACUUM しきい値を変更するには、テーブル名および TO threshold PERCENT パラメータを含めます。

SORT ONLY

削除した行によって解放されたスペースを再利用せずに、指定されたテーブル (または現在のデータベース内のすべてのテーブル) をソートします。このオプションは、ディスク容量の再利用が重要でなく、新しい行の再ソートが重要な場合に役に立ちます。ソートされていない領域が削除済みの行が大量に含んでおらず、ソート済み領域全体にまたがっていない場合に、SORT ONLY のバキューム操作を実行すると、バキューム操作にかかる時間が短縮されます。ディスク容量による拘束がなく、テーブルの行を常にソート状態に維持するクエリの最適化に依存するアプリケーションは、このようなバキューム操作からメリットを得ることができます。

デフォルトで、VACUUM SORT ONLY は少なくとも 95 パーセントがソート済みであるテーブルをスキップします。1 つのテーブルのデフォルトのソートしきい値を変更するには、VACUUM を実行するときに、テーブル名および TO threshold PERCENT パラメータを含めます。

DELETE ONLY

Amazon Redshift はバックグラウンドで自動的に DELETE ONLY vacuum を実行するため、DELETE ONLY vacuum を実行する必要は、ほとんどの場合ありません。

VACUUM DELETE は、直前の UPDATE 操作と DELETE 操作で削除対象のマークが付けられた行によって占有されているディスク容量を回収し、テーブルを圧縮して、消費されている領域を解放します。DELETE ONLY バキューム操作を実行しても、テーブルのデータはソートされません。

このオプションは、ディスク容量が重要で、新しい行の再ソートが重要でない場合に、バキューム操作にかかる時間を短縮します。このオプションは、クエリのパフォーマンスが既に最適で、行の再ソートのためにクエリのパフォーマンスを最適化する必要がない場合にも役立ちます。

デフォルトでは、VACUUM DELETE ONLY は、残りの行の少なくとも 95 パーセントが削除対象としてマークされていない領域を再利用します。1 つのテーブルのデフォルトの削除しきい値を変更するには、VACUUM を実行するときに、テーブル名および TO threshold PERCENT パラメータを含めます。

一部のオペレーション (`ALTER TABLE APPEND` など) により、テーブルは断片化される場合があります。`DELETE ONLY`句を使用すると、バキュームオペレーションにより、断片化されたテーブルから領域が解放されます。しきい値として同じ 95 パーセントが最適化オペレーションに適用されます。

REINDEX tablename

インターリーブソートキー列の値の分散を分析した後、完全バキューム処理を実行します。REINDEX を使用する場合は、テーブル名が必要です。

VACUUM REINDEX は、インターリーブソートキーを分析する目的で追加パスを作成するため、VACUUM FULL よりも大幅に実行時間が長くなります。インターリーブテーブルでは、ソート操作およびマージ操作の時間が長くなる場合があります。これは、インターリーブソートでは、複合ソートよりも多くの行の再調整が必要になる可能性があるためです。

VACUUM REINDEX 操作がその完了前に終了した場合、次の VACUUM は完全バキューム操作の実行前に REINDEX 操作を再開します。

VACUUM REINDEX は TO threshold PERCENT ではサポートされいません。

table_name

バキューム操作を実行するテーブルの名前。テーブル名を指定しない場合、バキューム操作は現在のデータベースのすべてのテーブルに適用されます。ユーザーが作成した常設テーブルまたは一時テーブルを指定できます。このコマンドは、ビューやシステムテーブルなど、他のオブジェクトに対しては意味を持ちません。

TO threshold PERCENT パラメータを含める場合は、テーブル名が必要です。

RECLUSTER tablename

テーブルのソートされていない部分をソートします。自動テーブルソートによってすでにソートされているテーブルの一部はそのまま残ります。このコマンドは、新しくソートされたデータをソート済みの領域とマージしません。また、削除対象としてマークされたすべての領域が、再利用されるわけではありません。このコマンドの完了後、テーブルが完全にソートされない状態で、SVV_TABLE_INFO の `unsorted` フィールドに表示されることがあります。

大規模なテーブルにおいて、取り込み頻度が高かったり最新のデータのみにアクセスするクエリを実行する場合には、VACUUM RECLUSTER を使用することをお勧めします。

VACUUM RECLUSTER は TO threshold PERCENT ではサポートされていません。RECLUSTER を使用する場合は、テーブル名が必要です。

VACUUM RECLUSTER は、インターリーブソートキーを使用する、分散スタイルが ALL に指定されたテーブルではサポートされません。

table_name

バキューム操作を実行するテーブルの名前。ユーザーが作成した常設テーブルまたは一時テーブルを指定できます。このコマンドは、ビューやシステムテーブルなど、他のオブジェクトに対しては意味を持ちません。

TO threshold PERCENT

VACUUM が削除フェーズをスキップする最低のしきい値、および削除フェーズでスペースを再利用する対象となるしきい値を指定する句です。ソートしきい値は、バキューム処理の前の指定されたテーブルにおけるソート済みの行の合計の割合です。 削除しきい値は、バキューム処理後に削除対象としてマークされていない行の合計の最低割合です。

VACUUM は、テーブルのソート済みの行の割合がソートしきい値以下の場合のみ行を再ソートするため、Amazon Redshift は、バキューム処理の時間を大幅に削減できます。同様に、VACUUM は削除対象としてマークされた行のスペースを 100 パーセント再利用するよう制約を受けない場合、削除対象としてマークされた行を数行のみ含むブロックの書き換えをスキップできます。

たとえば、しきい値に 75 を指定すると、テーブルの行の 75 パーセント以上がすでにソート済みの場合、VACUUM はソートフェーズをスキップします。削除フェーズでは、バキューム処理後に削除対象としてマークされていない行がテーブルに 少なくとも 75 パーセントあるようなテーブルを VACUUM はディスク領域の再利用の対象として設定します。しきい値の値は、0 から 100 の間の整数でなければなりません。デフォルトは 95 です。100 という値を指定すると、VACUUM はすでに完全にソートされていない限り常にテーブルをソートし、削除対象とし

てマークされたすべての行のスペースを再利用します。0 という値を指定すると、VACUUM は一切テーブルをソートせず、一切スペースを再利用しません。

TO threshold PERCENT パラメータを含める場合は、テーブルの名前も指定する必要があります。テーブル名を省略すると、VACUUM は失敗します。

TO threshold PERCENT パラメータは REINDEX と併用できません。

BOOST

使用可能なメモリやディスク容量などの追加のリソースを使用して VACUUM コマンドを実行します。BOOST オプションを使用して、VACUUM は 1 つのウィンドウで動作し、VACUUM 操作の期間の同時削除や更新をブロックします。BOOST オプションを使用して実行すると、システムリソースのために競合するので、クエリパフォーマンスに影響する場合があります。VACUUM BOOST は、メンテナンスオペレーションの間など、システムのロードが少ない場合に実行します。

BOOST オプションを使用する際は、以下を考慮します。

- BOOST を指定した場合は、table_name 値が必要です。
- BOOST は REINDEX でサポートされていません。
- BOOST は DELETE ONLY で無視されます。

使用に関する注意事項

ほとんどの Amazon Redshift アプリケーションでは、完全バキュームをお勧めします。詳細については、「テーブルのバキューム処理」を参照してください。

バキューム操作を実行する前に、以下の動作に注意してください。

- トランザクションブロック (BEGIN ... END) 内で VACUUM を実行することはできません。トランザクションの詳細については、「直列化可能分離」を参照してください。
- 一度にクラスターで実行できる VACUUM コマンドは 1 つだけです。複数のバキューム動作を同時に実行しようとすると、Amazon Redshift はエラーを返します。
- テーブルのバキュームを実行すると、ある程度のテーブル容量の増加が発生することがあります。再利用の対象となる削除済みの行が存在しない場合や、テーブルの新しいソート順によりデータ圧縮率が低下する場合、これは想定される動作です。
- バキューム操作の実行中、クエリのパフォーマンスがある程度低下することが予想されます。バキューム操作が終了すると直ちに通常のパフォーマンスに戻ります。

- バキュームオペレーション中にも、同時実行書き込みオペレーションは進行しますが、バキューム中の書き込みオペレーションの実行は推奨されていません。バキューム操作を実行する前に、書き込み操作を終了する方がより効率的です。また、バキューム操作開始後に書き込まれたすべてのデータは、その操作でバキュームすることができません。その場合は 2 回目のバキューム操作が必要です。
- ロード操作または挿入操作が既に進行中の場合、バキューム操作を開始できないことがあります。バキューム操作を開始するには、テーブルへの一時的な排他アクセスが必要になります。この排他アクセスは短時間しか必要でないため、バキューム操作により同時ロードと同時挿入が長時間ブロックされることはありません。
- 特定のテーブルに対して実行する作業がない場合、バキューム操作はスキップされます。ただし、操作がスキップ可能かどうかの検出に関連して、ある程度のオーバーヘッドが発生します。テーブルがあまり使われていないことが判明している場合、または、バキュームのしきい値を満たさないことが判明している場合は、これに対してバキューム操作を実行しないでください。
- 小さなテーブルに対して DELETE ONLY バキューム操作を実行した場合、データの保存に使われるブロック数が減少しないことがあります。特にテーブルに多数の列が存在している場合、またはクラスターがノードごとに多数のスライスを使用している場合、この傾向が顕著になります。このようなバキューム操作はテーブルへの同時挿入を実現するため、1 列ごとに 1 ブロックを各スライスに追加します。また、このようなオーバーヘッドが発生した場合、ブロック数の削減の方が再利用されるディスク容量を上回る可能性があります。例えばバキューム操作前に、8 ノードクラスター上の 10 列のテーブルが 1000 ブロックを占有している場合、削除された行が原因で 80 ブロックを超えるディスク容量が再利用されない限り、バキュームを実行しても実際のブロック数は減少しません。(各データブロックは 1 MB 使用します。)

次のいずれかの条件が満たされると、自動バキューム操作は一時停止します。

- ユーザーが ALTER TABLE などのデータ定義言語 (DDL) 操作を実行します。この操作では、現在自動バキューム処理が行われている表に対して排他ロックが必要です。
- ユーザーはクラスター内の任意のテーブルで VACUUM をトリガーします (一度に実行できる VACUUM は 1 つのみ)。
- クラスター負荷が高い期間。

例

すべてのテーブルで、デフォルトのバキュームしきい値 95 パーセントに基づいて、容量とデータベースの再利用と行の再ソートを実行します。

```
vacuum;
```

SALES テーブルで、デフォルトのしきい値 95 パーセントに基づいて、容量の再利用と行の再ソートを実行します。

```
vacuum sales;
```

SALES テーブルで、常に容量の再利用と行の再ソートを実行します。

```
vacuum sales to 100 percent;
```

SALES テーブルで、すでにソートされている行が 75 パーセント未満の場合のみ、行の再ソートを実行します。

```
 vacuum sort only sales to 75 percent;
```

バキューム処理後に削除対象としてマークされていない行が残りの行の 少なくとも 75 パーセント以上である SALES テーブルで容量の再利用を実行します。

```
vacuum delete only sales to 75 percent;
```

LISTING テーブルのインデックスを再作成した後、バキューム処理を実行します。

```
vacuum reindex listing;
```

次のコマンドはエラーを返します。

```
vacuum reindex listing to 75 percent;
```

LISTING テーブルのインデックスを再クラスターした後、バキューム処理を実行します。

```
vacuum recluster listing;
```

LISTING テーブルを再クラスターしてから、BOOST オプションを使用し、バキューム処理を実行します。

```
vacuum recluster listing boost;
```

SQL 関数リファレンス

トピック

Amazon Redshift は SQL 標準を拡張する多くの関数をサポートします。また、標準の集計関数、スカラー関数、およびウィンドウ関数もサポートします。

> **Note**
>
> Amazon Redshift は PostgreSQL に基づいています。Amazon Redshift と PostgreSQL の間には非常に重要な相違点がいくつかあり、データウェアハウスアプリケーションを設計して

開発するときはそれを考慮する必要があります。Amazon Redshift SQL と PostgreSQL の違いについては、Amazon Redshift および PostgreSQLを参照してください。

リーダーノード専用関数

一部のAmazon Redshift クエリは配信され、コンピューティングノードで実行されます。ほかのクエリはリーダーノードのみで実行されます。

クエリがユーザーが作成したテーブルまたはシステムテーブル (STL または STV を持つテーブル、および SVL または SVV プレフィックスを持つシステムビュー) を参照する場合、リーダーノードは SQL をコンピューティングノードに配布します。カタログテーブルのみを参照するクエリ (PG_TABLE_DEF など、PG プレフィックスを持つテーブル) またはテーブルを参照しないクエリは、リーダーノードのみで実行されます。

Amazon Redshift SQL 関数の中にはリーダーノードのみでサポートされ、コンピューティングノードではサポートされないものがあります。リーダーノード専用関数を使用するクエリは、コンピューティングノードではなくリーダーノードのみで実行される必要があり、そうでなければエラーが返されます。

各リーダーノード専用関数のドキュメントには注意点として、その関数がユーザー定義のテーブルまたは Amazon Redshift システムテーブルを参照する場合にエラーを返すことが示されています。

詳細については、「リーダーノードでサポートされる SQL 関数」を参照してください。

次の SQL 関数はリーダーノード専用関数であり、コンピューティングノードではサポートされません。

システム情報関数

- CURRENT_SCHEMA
- CURRENT_SCHEMAS
- HAS_DATABASE_PRIVILEGE
- HAS_SCHEMA_PRIVILEGE
- HAS_TABLE_PRIVILEGE

文字列関数

- SUBSTR

数学関数

- FACTORIAL()

以下のリーダーノード専用関数は廃止され、サポートは終了しました。

日付関数

- AGE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- LOCALTIME
- ISFINITE
- NOW

文字列関数

- GETBIT
- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

集計関数

トピック

- MIN 関数
- PERCENTILE_CONT 関数
- STDDEV_SAMP および STDDEV_POP 関数
- SUM 関数
- VAR_SAMP および VAR_POP 関数

集計関数は入力値のセットから 1 つの結果の値を計算します。

集計関数を使用する SELECT ステートメントには、2 つのオプション句 (GROUP BY および HAVING) を含めることができます。これらの句の構文は次のとおりです (例として COUNT 関数を使用)。

```
SELECT count (*) expression FROM table_reference
WHERE condition [GROUP BY expression ] [ HAVING condition]
```

GROUP BY 句は、指定した列 (単数または複数) の一意の値によって結果を集計およびグループ化します。HAVING 句は、特定の集計条件が真の場合 (例: count (*) > 1) に行に返す結果を制限します。HAVING 句は行の値に基づく列を制限するために、WHERE と同様に使用されます。追加されたこれらの句の例については、COUNTを参照してください。

集計関数は、入れ子にした集計関数またはウィンドウ関数を引数として使用できません。

ANY_VALUE 関数

ANY_VALUE 関数は、入力式の値から任意の値を非決定的に返します。この関数は、入力式で行が返されない場合に NULL を返します。この関数は、入力式に NULL 値がある場合にも NULL を返します。

構文

```
ANY_VALUE( [ DISTINCT | ALL ] expression )
```

引数

DISTINCT | ALL

DISTINCT または ALL のいずれかを指定すると、入力式の値から任意の値が返されます。DISTINCT 引数は効果がなく、無視されます。

expression

関数が動作するターゲット列または式。式は、以下に示すデータ型の 1 つを取ります。

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- BOOLEAN
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- VARBYTE
- SUPER
- HLLSKETCH
- GEOMETRY
- GEOGRAPHY

戻り値

同じデータ型を expression として返します。

使用に関する注意事項

列の ANY_VALUE 関数を指定するステートメントに 2 番目の列参照も含まれている場合、2 番目の列は GROUP BY 句に含めるか、集計関数に含める必要があります。

例

この例では、Amazon Redshift 入門ガイドの「ステップ 4: Amazon S3 のサンプルデータをロードする」で作成したイベントテーブルを使用します。次の例では、イベント名が Eagles である任意の dateid のインスタンスを返します。

```
select any_value(dateid) as dateid, eventname from event where eventname ='Eagles'
 group by eventname;
```

結果は、以下の通りです。

```
dateid | eventname
-------+---------------
 1878  | Eagles
```

次の例では、イベント名が Eagles または Cold War Kids である任意の dateid のインスタンスを返します。

```
select any_value(dateid) as dateid, eventname from event where eventname in('Eagles',
 'Cold War Kids') group by eventname;
```

結果は、以下のとおりです。

```
dateid | eventname
-------+---------------
 1922  | Cold War Kids
 1878  | Eagles
```

APPROXIMATE PERCENTILE_DISC 関数

APPROXIMATE PERCENTILE_DISC は、離散型分散モデルを前提とする逆分散関数です。これは、パーセンタイル値とソート仕様を取得し、特定のセットからエレメントを返します。概算により、関数の実行がはるかに高速になり、相対誤差は約 0.5% と低くなります。

特定の percentile 値に対して、APPROXIMATE PERCENTILE_DISC は分位数要約アルゴリズムを使用して ORDER BY 句の式の離散パーセンタイルを概算します。APPROXIMATE PERCENTILE_DISC は、percentile と同じであるかそれより大きい最少累積分散値 (同じソート仕様を基準とする) を持つ値を返します。

構文

```
APPROXIMATE  PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
```

引数

percentile

0 と 1 の間の数値定数。Null は計算では無視されます。

WITHIN GROUP (ORDER BY expr)

パーセンタイルをソートして計算するための数値または日付/時間値を指定する句。

戻り値

WITHIN GROUP 句の ORDER BY 式と同じデータ型。

使用に関する注意事項

APPROXIMATE PERCENTILE_DISC ステートメントに GROUP BY 句が含まれている場合、結果セットは制限されます。制限は、ノードタイプとノード数によって異なります。制限を超えると、関数は失敗して以下のエラーを返します。

```
GROUP BY limit for approximate percentile_disc exceeded.
```

評価するグループの数が制限の許可数を超える場合は、PERCENTILE_CONT 関数の使用を検討してください。

例

次の例では、上位 10 日の販売数、販売総額、50 番目のパーセンタイル値を返します。

```
select top 10 date.caldate,
count(totalprice), sum(totalprice),
approximate percentile_disc(0.5)
within group (order by totalprice)
from listing
join date on listing.dateid = date.dateid
```

```
group by date.caldate
order by 3 desc;

caldate    | count | sum        | percentile_disc
-----------+-------+------------+----------------
2008-01-07 |   658 | 2081400.00 |         2020.00
2008-01-02 |   614 | 2064840.00 |         2178.00
2008-07-22 |   593 | 1994256.00 |         2214.00
2008-01-26 |   595 | 1993188.00 |         2272.00
2008-02-24 |   655 | 1975345.00 |         2070.00
2008-02-04 |   616 | 1972491.00 |         1995.00
2008-02-14 |   628 | 1971759.00 |         2184.00
2008-09-01 |   600 | 1944976.00 |         2100.00
2008-07-29 |   597 | 1944488.00 |         2106.00
2008-07-23 |   592 | 1943265.00 |         1974.00
```

AVG 関数

AVG 関数は、入力式の値の平均 (算術平均) を返します。AVG 関数は数値に対してはたらき、NULL 値は無視します。

構文

```
AVG ( [ DISTINCT | ALL ] expression )
```

引数

expression

関数の対象となる列または式。式は、以下に示すデータ型の 1 つを取ります。

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISION
- SUPER

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は平均を計算する前に重複した値をすべて指定された式から削除します。引数 ALL を指定すると、この関数は平均を計算する式の重複した値をすべて保持します。ALL がデフォルトです。

データ型

AVG 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、DOUBLE PRECISION および SUPER です。

AVG 関数でサポートされる戻り値の型は次のとおりです。

- 整数型の引数の場合は BIGINT
- 浮動小数点の引数の場合は DOUBLE PRECISION
- 他の引数型については、expression と同じデータ型を返します。

NUMERIC または DECIMAL 引数を使用した AVG 関数の結果のデフォルト精度は 38 です。結果のスケールは、引数のスケールと同じです。例えば、DEC(5,2) 列の AVG は DEC(38,2) のデータ型を返します。

例

SALES テーブルから取引ごとの平均販売量を検索します。

```
select avg(qtysold)from sales;

avg
-----
2
(1 row)
```

すべてのリストに一覧表示された合計額の平均を検索します。

```
select avg(numtickets*priceperticket) as avg_total_price from listing;

avg_total_price
```

```
-----------------
3034.41
(1 row)
```

支払い額の平均を月ごとに降順で検索します。

```
select avg(pricepaid) as avg_price, month
from sales, date
where sales.dateid = date.dateid
group by month
order by avg_price desc;

avg_price | month
-----------+-------
659.34 | MAR
655.06 | APR
645.82 | JAN
643.10 | MAY
642.72 | JUN
642.37 | SEP
640.72 | OCT
640.57 | DEC
635.34 | JUL
635.24 | FEB
634.24 | NOV
632.78 | AUG
(12 rows)
```

COUNT 関数

COUNT 関数は式で定義された行をカウントします。

COUNT 関数には 3 つのバリエーションがあります。

- COUNT(*) は null を含むかどうかにかかわらず、ターゲットテーブルのすべての行をカウントします。
- COUNT (expression) は、特定の列または式にある Null 以外の値を持つ行数を計算します。
- COUNT (DISTINCT expression) は、列または式にある Null 以外の一意な値の数を計算します。
- APPROXIMATE COUNT DISTINCT は、列または式にある Null 以外の個別の値の数を概算します。

構文

```
COUNT( * | expression )
```

```
COUNT ( [ DISTINCT | ALL ] expression )
```

```
APPROXIMATE COUNT ( DISTINCT expression )
```

引数

expression

関数の対象となる列または式。COUNT 関数は引数のデータ型をすべてサポートしています。

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数はカウントを行う前に指定された式から重複した値をすべて削除します。引数 ALL を指定すると、この関数はカウントに使用する式から重複する値をすべて保持します。ALL がデフォルトです。

APPROXIMATE

COUNT DISTINCT 関数を APPROXIMATE とともに使用すると、HyperLogLog アルゴリズムを使用して、列または式にある Null 以外の個別の値の数を概算します。APPROXIMATE キーワードを使用するクエリは、はるかに高速に実行され、相対誤差は約 2% と低くなります。クエリあたり、または GROUP BY 句がある場合にはグループあたりで、数百万個以上の多数の個別の値を返すクエリについては、概算を使用するのが妥当です。個別の値が数千個のように比較的少ない場合は、概算は正確なカウントよりも低速になる可能性があります。APPROXIMATE は、COUNT DISTINCT でのみ使用できます。

戻り型

COUNT 関数は BIGINT を返します。

例

フロリダ州のユーザーをすべてカウントします。

```
select count(*) from users where state='FL';
```

```
count
-------
510
```

EVENT テーブルからすべてのイベント名をカウントします。

```
select count(eventname) from event;

count
-------
8798
```

EVENT テーブルからすべてのイベント名をカウントします。

```
select count(all eventname) from event;

count
-------
8798
```

EVENT テーブルから一意の会場 ID をすべてカウントします。

```
select count(distinct venueid) as venues from event;

venues
---------
204
```

4 枚より多いチケットをまとめて販売した販売者ごとの回数をカウントします。販売者 ID で結果をグループ化します。

```
select count(*), sellerid from listing
where numtickets > 4
group by sellerid
order by 1 desc, 2;

count | sellerid
------+----------
12    |    6386
11    |    17304
11    |    20123
```

```
11    |    25428
...
```

次の例では、COUNT および APPROXIMATE COUNT の戻り値と実行時間を比較します。

```
select  count(distinct pricepaid) from sales;

count
-------
  4528


Time: 48.048 ms


select approximate count(distinct pricepaid) from sales;

count
-------
  4553


Time: 21.728 ms
```

LISTAGG 関数

クエリの各グループについて、LISTAGG 集計関数は、ORDER BY 式に従ってそのグループの行をソートしてから、それらの値を 1 つの文字列に連結します。

構文

```
LISTAGG( [DISTINCT] aggregate_expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
```

引数

DISTINCT

連結する前に、指定した式から重複した値を削除する句。末尾のスペースは無視されます。例えば、文字列 'a' および 'a ' は重複として扱われます。LISTAGG は、発生した最初の値を使用します。詳細については、「末尾の空白の重要性」を参照してください。

aggregate_expression

集計する値を返す任意の有効な式 (列名など)。NULL 値および空の文字列は無視されます。

delimiter

連結された値を区切る文字列定数。デフォルトは NULL です。

WITHIN GROUP (ORDER BY order_list)

集計値のソート順を指定する句。

戻り値

VARCHAR(MAX)。結果セットが VARCHAR の最大サイズより大きい場合、LISTAGG は次のエラーを返します。

```
Invalid operation: Result size exceeds LISTAGG limit
```

使用に関する注意事項

- WITHIN GROUP 句を使用する複数の LISTAGG 関数が 1 つのステートメントに含まれる場合、各 WITHIN GROUP 句で ORDER BY 値を使用する必要があります。

  例えば、次のステートメントはエラーを返します。

  ```
  SELECT LISTAGG(sellerid)
  WITHIN GROUP (ORDER BY dateid) AS sellers,
  LISTAGG(dateid)
  WITHIN GROUP (ORDER BY sellerid) AS dates
  FROM sales;
  ```

  以下のステートメントは正常に実行されます。

  ```
  SELECT LISTAGG(sellerid)
  WITHIN GROUP (ORDER BY dateid) AS sellers,
  LISTAGG(dateid)
  WITHIN GROUP (ORDER BY dateid) AS dates
  FROM sales;

  SELECT LISTAGG(sellerid)
  WITHIN GROUP (ORDER BY dateid) AS sellers,
  LISTAGG(dateid) AS dates
  ```

```
FROM sales;
```

例

以下の例は、販売者 ID を集計し、販売者 ID 順で返します。

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
FROM sales
WHERE eventid = 4337;

listagg

------------------------------------------------------------------------------------------------
380, 380, 1178, 1178, 1178, 2731, 8117, 12905, 32043, 32043, 32043, 32432, 32432,
 38669, 38750, 41498, 45676, 46324, 47188, 47188, 48294
```

次の例では、DISTINCT を使用して一意の販売者 ID のリストを返します。

```
SELECT LISTAGG(DISTINCT sellerid, ', ')
WITHIN GROUP (ORDER BY sellerid)
FROM sales
WHERE eventid = 4337;

listagg

-------------------------------------------------------------------------------------------
380, 1178, 2731, 8117, 12905, 32043, 32432, 38669, 38750, 41498, 45676, 46324, 47188,
 48294
```

以下の例は、販売者 ID を集計し、日付順で返します。

```
SELECT LISTAGG(sellerid, ', ')
WITHIN GROUP (ORDER BY dateid)
FROM sales
WHERE eventid = 4337;

   listagg
------------------------------------------------------------------------------------------------
 41498, 47188, 47188, 1178, 1178, 1178, 380, 45676, 46324, 48294, 32043, 32043, 32432,
 12905, 8117, 38750, 2731, 32432, 32043, 380, 38669
```

次の例は、ID が 660 の購入者についてパイプで区切った販売日のリストを返します。

```
SELECT LISTAGG(
    (SELECT caldate FROM date WHERE date.dateid=sales.dateid), ' | '
)
WITHIN GROUP (ORDER BY sellerid DESC, salesid ASC)
FROM sales
WHERE buyerid = 660;

             listagg
-------------------------------------------------
2008-07-16 | 2008-07-09 | 2008-01-01 | 2008-10-26
```

次の例は、ID が 660、661、662 の購入者についてカンマで区切った販売 ID のリストを返します。

```
SELECT buyerid,
LISTAGG(salesid,', ')
WITHIN GROUP (ORDER BY salesid) AS sales_id
FROM sales
WHERE buyerid BETWEEN 660 AND 662
GROUP BY buyerid
ORDER BY buyerid;

buyerid |                sales_id
--------+------------------------------------------------------
660     | 32872, 33095, 33514, 34548
661     | 19951, 20517, 21695, 21931
662     | 3318, 3823, 4215, 51980, 53202, 55908, 57832, 171603
```

MAX 関数

MAX 関数は行のセットの最大値を返します。DISTINCT または ALL が使用される可能性がありますが、結果には影響しません。

構文

```
MAX ( [ DISTINCT | ALL ] expression )
```

引数

expression

関数の対象となる列または式。式は、以下に示すデータ型の 1 つを取ります。

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は最大値を計算する前に指定された式から重複した値をすべて削除します。引数 ALL を指定すると、この関数は最大値を計算する式から重複する値をすべて保持します。ALL がデフォルトです。

データ型

同じデータ型を expression として返します。MIN 関数のブールバージョンは BOOL_AND 関数 であり、MAX 関数のブールバージョンは BOOL_OR 関数 です。

例

すべての販売から最高支払価格を検索します。

```
select max(pricepaid) from sales;

max
----------
12624.00
(1 row)
```

すべての販売からチケットごとの最高支払価格を検索します。

```
select max(pricepaid/qtysold) as max_ticket_price
from sales;

max_ticket_price
-----------------
2500.00000000
(1 row)
```

MEDIAN 関数

値の範囲の中央値を計算します。範囲内の NULL 値は無視されます。

MEDIAN は、連続型分散モデルを前提とする逆分散関数です。

MEDIAN は PERCENTILE_CONT の特殊なケースです。

構文

```
MEDIAN(median_expression)
```

引数

median_expression

関数の対象となる列または式。

データ型

戻り値の型は、データ型 median_expression によって決まります。次の表は、各 median_expression 式のデータ型に対応する戻り型を示しています。

Input type	戻り型
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用に関する注意事項

median_expression 引数が DECIMAL データ型であり、その最大精度が 38 桁である場合、MEDIAN が不正確な結果またはエラーを返す可能性があります。MEDIAN 関数の戻り値が 38 桁を超える場合、結果は 38 桁までとなり、39 桁以降は切り捨てられるため、精度が失われます。補間中に中間結果が最大精度を超えた場合には、数値オーバーフローが発生し、この関数はエラーを返します。このような状態を回避するため、精度が低いデータ型を使用するか、median_expression 引数を低い精度にキャストすることをお勧めします。

ステートメントにソートベースの集計関数 (LISTAGG、PERCENTILE_CONT、または MEDIAN) に対する複数の呼び出しが含まれる場合、すべて同じ ORDER BY 値を使用する必要があります。MEDIAN では、式の値による暗黙的な順序が適用されることに注意してください。

例えば、次のステートメントはエラーを返します。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;
```

```
ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

次のステートメントは正常に実行されます。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
GROUP BY salesid, pricepaid;
```

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

以下は、MEDIAN が PERCENTILE_CONT(0.5) と同じ結果を生成する例です。

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
|        2 |       2 |               2 |      2 |
|       26 |       1 |               1 |      1 |
|       33 |       1 |               1 |      1 |
|       38 |       1 |               1 |      1 |
|       43 |       1 |               1 |      1 |
|       48 |       2 |               2 |      2 |
|       48 |       3 |               3 |      3 |
|       77 |       4 |               4 |      4 |
|       85 |       4 |               4 |      4 |
|       95 |       2 |               2 |      2 |
+----------+---------+-----------------+--------+
```

次の例では、各 sellerid の平均販売数量を求めます。

```
SELECT sellerid,
```

```
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+
| sellerid | median |
+----------+--------+
|        1 |    1.5 |
|        2 |      2 |
|        3 |      2 |
|        4 |      2 |
|        5 |      1 |
|        6 |      1 |
|        7 |    1.5 |
|        8 |      1 |
|        9 |      4 |
|       12 |      2 |
+----------+--------+
```

最初の sellerid に対する前回のクエリの結果を検証するには、次の例を使用します。

```
SELECT qtysold
FROM sales
WHERE sellerid=1;

+---------+
| qtysold |
+---------+
|       2 |
|       1 |
+---------+
```

MIN 関数

MIN 関数は行のセットの最小値を返します。DISTINCT または ALL が使用される可能性がありますが、結果には影響しません。

構文

```
MIN ( [ DISTINCT | ALL ] expression )
```

引数

expression

関数の対象となる列または式。式は、以下に示すデータ型の 1 つを取ります。

- SMALLINT
- INTEGER
- BIGINT
- DECIMAL
- REAL
- DOUBLE PRECISION
- CHAR
- VARCHAR
- DATE
- TIMESTAMP
- TIMESTAMPTZ
- TIME
- TIMETZ
- VARBYTE
- SUPER

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は最小値を計算する前に指定された式から重複した値をすべて削除します。引数 ALL を指定すると、この関数は最小値を計算する式から重複する値をすべて保持します。ALL がデフォルトです。

データ型

同じデータ型を expression として返します。MIN 関数のブールバージョンは BOOL_AND 関数 であり、MAX 関数のブールバージョンは BOOL_OR 関数 です。

例

すべての販売から最低支払価格を検索します。

```
select min(pricepaid) from sales;
```

```
min
-------
20.00
(1 row)
```

すべての販売からチケットごとの最低支払価格を検索します。

```
select min(pricepaid/qtysold)as min_ticket_price
from sales;

min_ticket_price
------------------
20.00000000
(1 row)
```

PERCENTILE_CONT 関数

PERCENTILE_CONT は、連続型分散モデルを前提とする逆分散関数です。これは、パーセンタイル値とソート仕様を取得し、ソート仕様を基準として、そのパーセンタイル値に該当する補間値を返します。

PERCENTILE_CONT は、値の順序付けを行った後に値の間の線形補間を計算します。この関数は、パーセンタイル値 (`P`) と集計グループの Null ではない行の数 (`N`) を使用して、ソート使用に従って行の順序付けを行った後に行番号を計算します。この行番号 (`RN`) は、計算式 `RN = (1+ (P*(N-1))` に従って計算されます。集計関数の最終結果は、行番号 `CRN = CEILING(RN)` および `FRN = FLOOR(RN)` にある行の値の間の線形補間に基づいて計算されます。

最終結果は次のとおりです。

`(CRN = FRN = RN)` である場合、結果は `(value of expression from row at RN)`

そうでない場合、結果は

`(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN)`.

構文

```
PERCENTILE_CONT(percentile)
WITHIN GROUP(ORDER BY expr)
```

引数

percentile

0 と 1 の間の数値定数。NULL 値は計算で無視されます。

expr

パーセンタイルをソートして計算するための数値または日付/時間値を指定します。

戻り値

戻り型は、WITHIN GROUP 句の ORDER BY 式のデータ型に基づいて決定されます。次の表は、各 ORDER BY 式のデータ型に対応する戻り型を示しています。

Input type	戻り型
INT2, INT4, INT8, NUMERIC, DECIMAL	DECIMAL
FLOAT, DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP
TIMESTAMPTZ	TIMESTAMPTZ

使用に関する注意事項

ORDER BY 式が DECIMAL データ型であり、その最大精度が 38 桁である場合、PERCENTILE_CONT が不正確な結果またはエラーを返す可能性があります。PERCENTILE_CONT 関数の戻り値が 38 桁を超える場合、結果は 38 桁までとなり、39 桁以降は切り捨てられるため、精度が失われます。補間中に中間結果が最大精度を超えた場合には、数値オーバーフローが発生し、この関数はエラーを返します。このような状態を回避するため、精度が低いデータ型を使用するか、ORDER BY 式を低い精度にキャストすることをお勧めします。

ステートメントにソートベースの集計関数 (LISTAGG、PERCENTILE_CONT、または MEDIAN) に対する複数の呼び出しが含まれる場合、すべて同じ ORDER BY 値を使用する必要があります。MEDIAN では、式の値による暗黙的な順序が適用されることに注意してください。

例えば、次のステートメントはエラーを返します。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

An error occurred when executing the SQL command:
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(pricepaid)
FROM sales
GROUP BY salesid, pricepaid;

ERROR: within group ORDER BY clauses for aggregate functions must be the same
```

次のステートメントは正常に実行されます。

```
SELECT TOP 10 salesid, SUM(pricepaid),
PERCENTILE_CONT(0.6) WITHIN GROUP(ORDER BY salesid),
MEDIAN(salesid)
FROM sales
GROUP BY salesid, pricepaid;
```

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

以下は、PERCENTILE_CONT(0.5) が MEDIAN と同じ結果を生成する例です。

```
SELECT TOP 10 DISTINCT sellerid, qtysold,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold),
MEDIAN(qtysold)
FROM sales
GROUP BY sellerid, qtysold;

+----------+---------+-----------------+--------+
| sellerid | qtysold | percentile_cont | median |
+----------+---------+-----------------+--------+
|        2 |       2 |               2 |      2 |
|       26 |       1 |               1 |      1 |
|       33 |       1 |               1 |      1 |
```

```
|        38 |       1 |                 1 |        1 |
|        43 |       1 |                 1 |        1 |
|        48 |       2 |                 2 |        2 |
|        48 |       3 |                 3 |        3 |
|        77 |       4 |                 4 |        4 |
|        85 |       4 |                 4 |        4 |
|        95 |       2 |                 2 |        2 |
+----------+---------+-----------------+--------+
```

次の例では、SALES テーブルの各 selleridD の販売数量の PERCENTILE_CONT(0.5) と PERCENTILE_CONT(0.75) を求めます。

```
SELECT sellerid,
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY qtysold) as pct_05,
PERCENTILE_CONT(0.75) WITHIN GROUP(ORDER BY qtysold) as pct_075
FROM sales
GROUP BY sellerid
ORDER BY sellerid
LIMIT 10;

+----------+--------+---------+
| sellerid | pct_05 | pct_075 |
+----------+--------+---------+
|        1 |    1.5 |    1.75 |
|        2 |      2 |    2.25 |
|        3 |      2 |       3 |
|        4 |      2 |       2 |
|        5 |      1 |     1.5 |
|        6 |      1 |       1 |
|        7 |    1.5 |    1.75 |
|        8 |      1 |       1 |
|        9 |      4 |       4 |
|       12 |      2 |    3.25 |
+----------+--------+---------+
```

最初の sellerid に対する前回のクエリの結果を検証するには、次の例を使用します。

```
SELECT qtysold
FROM sales
WHERE sellerid=1;

+---------+
| qtysold |
```

```
+---------+
|       2 |
|       1 |
+---------+
```

STDDEV_SAMP および STDDEV_POP 関数

STDDEV_SAMP および STDDEV_POP 関数は、数値のセットの標本標準偏差と母集団標準偏差 (整数、10 進数、または浮動小数点) を返します。STDDEV_SAMP 関数の結果は、値の同じセットの標本分散の平方根に相当します。

STDDEV_SAMP および STDDEV は、同じ関数のシノニムです。

構文

```
STDDEV_SAMP | STDDEV ( [ DISTINCT | ALL ] expression)
STDDEV_POP ( [ DISTINCT | ALL ] expression)
```

この式は整数、10 進数、または浮動小数点数データ型である必要があります。式のデータ型にかかわらず、この関数の戻り値の型は倍精度の数値です。

> **Note**
> 標準偏差は浮動小数点演算を使用して計算されます。これはわずかに不正確である可能性があります。

使用に関する注意事項

標本標準偏差 (STDDEV または STDDEV_SAMP) が 1 つの値で構成される式に対して計算される場合、関数の結果は 0 ではなく、NULL になります。

例

次のクエリは VENUE テーブルの VENUESEATS 列の値の平均、続いて値の同じセットの標本標準偏差と母集団標準偏差を返します。VENUESEATS は INTEGER 列です。結果のスケールは 2 桁に減らされます。

```
select avg(venueseats),
```

```
cast(stddev_samp(venueseats) as dec(14,2)) stddevsamp,
cast(stddev_pop(venueseats) as dec(14,2)) stddevpop
from venue;

avg  | stddevsamp | stddevpop
-------+------------+-----------
17503 |   27847.76 |  27773.20
(1 row)
```

次のクエリは SALES テーブルの COMMISSION 列に標本標準偏差を返します。COMMISSION は DECIMAL 列です。結果のスケールは 10 桁に減らされます。

```
select cast(stddev(commission) as dec(18,10))
from sales;

stddev
----------------
130.3912659086
(1 row)
```

次のクエリは整数として COMMISSION 列の標本標準偏差をキャストします。

```
select cast(stddev(commission) as integer)
from sales;

stddev
--------
130
(1 row)
```

次のクエリは COMMISSION 列に標本標準偏差と標本分散の平方根の両方を返します。これらの計算の結果は同じです。

```
select
cast(stddev_samp(commission) as dec(18,10)) stddevsamp,
cast(sqrt(var_samp(commission)) as dec(18,10)) sqrtvarsamp
from sales;

stddevsamp    |  sqrtvarsamp
----------------+----------------
130.3912659086 | 130.3912659086
```

```
(1 row)
```

SUM 関数

SUM 関数は入力列または式の値の合計を返します。SUM 関数は数値に対してはたらき、NULL 値を無視します。

構文

```
SUM ( [ DISTINCT | ALL ] expression )
```

引数

expression

関数の対象となる列または式。式は、以下に示すデータ型の 1 つを取ります。

- SMALLINT
- INTEGER
- BIGINT
- NUMERIC
- DECIMAL
- REAL
- DOUBLE PRECISION
- SUPER

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は合計を計算する前に指定された式から重複した値をすべて削除します。引数 ALL を指定すると、この関数は合計を計算する式から重複する値をすべて保持します。ALL がデフォルトです。

データ型

SUM 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、DOUBLE PRECISION および SUPER です。

SUM 関数でサポートされる戻り値の型は次のとおりです。

- BIGINT、SMALLINT、および INTEGER 引数の場合は BIGINT
- NUMERIC 引数の場合は NUMERIC
- 浮動小数点の引数の場合は DOUBLE PRECISION
- 他の引数型については、expression と同じデータ型を返します。

NUMERIC または DECIMAL 引数を持つ SUM 関数結果のデフォルト精度は 38 です。結果のスケールは、引数のスケールと同じです。例えば、DEC(5,2) 列の SUM は DEC(38,2) のデータ型を返します。

例

SALES テーブルから支払われたすべての手数料の合計を検索します。

```
select sum(commission) from sales;

sum
-------------
16614814.65
(1 row)
```

フロリダ州の全会場の座席数を検索します。

```
select sum(venueseats) from venue
where venuestate = 'FL';

sum
--------
250411
(1 row)
```

5 月に販売された座席数を検索します。

```
select sum(qtysold) from sales, date
where sales.dateid = date.dateid and date.month = 'MAY';

sum
-------
```

```
32291
(1 row)
```

VAR_SAMP および VAR_POP 関数

VAR_SAMP および VAR_POP 関数は、数値のセットの標本分散と母集団分散 (整数、10 進数、または浮動小数点) を返します。VAR_SAMP 関数の結果は、値の同じセットの 2 乗の標本標準偏差に相当します。

VAR_SAMP および VARIANCE は同じ関数のシノニムです。

構文

```
VAR_SAMP | VARIANCE ( [ DISTINCT | ALL ] expression)
VAR_POP ( [ DISTINCT | ALL ] expression)
```

この式は整数、10 進数、または浮動小数点数データ型である必要があります。式のデータ型にかかわらず、この関数の戻り値の型は倍精度の数値です。

> **Note**
> これらの関数の結果は、それぞれのクラスターの設定に応じて、データウェアハウスクラスターによって変わる場合があります。

使用に関する注意事項

標本分散 (VARIANCE または VAR_SAMP) が 1 つの値で構成される式に対して計算される場合、関数の結果は 0 ではなく、NULL になります。

例

次のクエリは LISTING テーブルの NUMTICKETS 列の四捨五入した標本分散と母集団分散を返します。

```
select avg(numtickets),
round(var_samp(numtickets)) varsamp,
round(var_pop(numtickets)) varpop
from listing;
```

```
avg | varsamp | varpop
-----+---------+--------
10 |      54 |     54
(1 row)
```

次のクエリは同じ計算を実行しますが、10 進値の結果をキャストします。

```
select avg(numtickets),
cast(var_samp(numtickets) as dec(10,4)) varsamp,
cast(var_pop(numtickets) as dec(10,4)) varpop
from listing;

avg | varsamp | varpop
-----+---------+---------
10 | 53.6291 | 53.6288
(1 row)
```

配列関数

以下に、Amazon Redshift が配列にアクセスして操作するためにサポートする SQL 配列関数の説明を示します。

トピック

array 関数

SUPER データ型の配列を作成します。

構文

```
ARRAY( [ expr1 ] [ , expr2 [ , ... ] ] )
```

引数

expr1、expr2

Amazon Redshift では、日付と時刻の型は SUPER データ型にキャストされないため、任意の Amazon Redshift データ型の式では日付と時刻の型は除外されます。引数は同じデータ型である必要はありません。

戻り型

配列関数は、SUPER データ型を返します。

例

次の例は、数値の配列と、さまざまなデータ型の配列を示しています。

```
--an array of numeric values
select array(1,50,null,100);
      array
------------------
 [1,50,null,100]
(1 row)

--an array of different data types
select array(1,'abc',true,3.14);
        array
-----------------------
 [1,"abc",true,3.14]
(1 row)
```

array_concat 関数

array_concat 関数は、2 つの配列を連結して、最初の配列のすべての要素と、それに続く 2 番目の配列のすべての要素を含む配列を作成します。2 つの引数は有効な配列でなければなりません。

構文

```
array_concat( super_expr1,  super_expr2 )
```

引数

super_expr1

連結する 2 つの配列のうち、最初の配列を指定する値。

super_expr2

連結する 2 つの配列のうち、2 番目の配列を指定する値。

戻り型

array_concat 関数は、SUPER データ値を返します。

例

次の例は、同じ型の 2 つの配列の連結と、異なる型の 2 つの配列の連結を示しています。

```
-- concatenating two arrays
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY(10003,10004));
              array_concat
------------------------------------
 [10001,10002,10003,10004]
(1 row)

-- concatenating two arrays of different types
SELECT ARRAY_CONCAT(ARRAY(10001,10002),ARRAY('ab','cd'));
          array_concat
------------------------------
 [10001,10002,"ab","cd"]
(1 row)
```

array_flatten 関数

複数の配列を SUPER 型の単一の配列にマージします。

構文

```
array_flatten( super_expr1,super_expr2,.. )
```

引数

super_expr1、super_expr2

配列形式の有効な SUPER 表現。

戻り型

array_flatten 関数は、SUPER データ値を返します。

例

次の例は、array_flatten 関数を示しています。

```
SELECT ARRAY_FLATTEN(ARRAY(ARRAY(1,2,3,4),ARRAY(5,6,7,8),ARRAY(9,10)));
     array_flatten
------------------------
 [1,2,3,4,5,6,7,8,9,10]
(1 row)
```

get_array_length 関数

指定された配列の長さを返します。GET_ARRAY_LENGTH 関数は、オブジェクトまたは配列パスが与えられた SUPER 配列の長さを返します。

構文

```
get_array_length( super_expr )
```

引数

super_expr

配列形式の有効な SUPER 表現。

戻り型

get_array_length 関数は、BIGINT を返します。

例

次の例は、get_array_length 関数を示しています。

```
SELECT GET_ARRAY_LENGTH(ARRAY(1,2,3,4,5,6,7,8,9,10));
 get_array_length
----------------------
            10
(1 row)
```

split_to_array 関数

区切り文字を任意のパラメータとして使用します。区切り文字がない場合、デフォルトはコンマです。

構文

```
split_to_array( string,delimiter )
```

引数

string

分割する入力文字列。

delimiter

入力文字列が分割されるオプションの値。デフォルトではカンマを使用します。

戻り型

split_to_array 関数は、SUPER データ値を返します。

例

次の例は、split_to_array 関数を示しています。

```
SELECT SPLIT_TO_ARRAY('12|345|6789', '|');
     split_to_array
-------------------------
 ["12","345","6789"]
(1 row)
```

subarray 関数

入力配列のサブセットを返すように配列を操作します。

構文

```
SUBARRAY( super_expr, start_position, length )
```

引数

super_expr

配列形式で有効な SUPER 表現です。

start_position

インデックス位置 0 から始めて、抽出を開始する配列内の位置。負の位置は、配列の最後から逆方向にカウントされます。

length

抽出する要素の数 (サブ文字列の長さ)。

戻り型

subarray 関数は、SUPER データ値を返します。

例

サブアレイ関数の例を次に示します。

```
 SELECT SUBARRAY(ARRAY('a', 'b', 'c', 'd', 'e', 'f'), 2, 3);
   subarray
---------------
 ["c","d","e"]
(1 row)
```

ビット単位の集計関数

ビット単位の集計関数は、ビット演算を計算し、整数値に変換または丸めることができる整数の列と列の集計を実行します。

トピック

- ビット単位の集計 NULL を使用する
- ビット単位の集計の DISTINCT サポート

- ビット単位関数の概要の例
- BIT_AND 関数
- BIT_OR 関数
- BOOL_AND 関数
- BOOL_OR 関数

ビット単位の集計 NULL を使用する

ビット単位関数が Null 使用可能な列に適用されている場合、NULL 値は関数の結果が計算される前に削除されます。集計を満たす行がない場合、ビット単位関数は NULL を返します。同じ動作が標準の集計関数に適用されます。次に例を示します。

```
select sum(venueseats), bit_and(venueseats) from venue
where venueseats is null;

sum  | bit_and
------+---------
null |    null
(1 row)
```

ビット単位の集計の DISTINCT サポート

他の集計関数のように、ビット単位関数は DISTINCT キーワードをサポートします。

ただし、これらの関数で DISTINCT を使用しても結果に影響はありません。値の最初のインスタンスは、ビット単位の AND または OR 演算を満たすのに十分です。評価される式に重複する値が存在する場合でも、違いはありません。

DISTINCT プロセスはクエリ実行オーバーヘッドを伴う可能性があるため、ビット単位の関数で DISTINCT を使用しないことをお勧めします。

ビット単位関数の概要の例

以下に、ビット単位の関数を操作する方法を示す概要例をいくつか示します。また、各関数の説明とともに特定のコード例を見つけることができます。

ビット単位の関数の例は、TICKIT サンプルデータベースに基づいています。TICKIT サンプルデータベースの USERS テーブルには複数のブール列が含まれ、ユーザーごとに異なるイベントのタイプ (スポーツ、演劇、オペラなど) を好きかどうかを示しています。以下に例を示します。

```
select userid, username, lastname, city, state,
likesports, liketheatre
from users limit 10;

userid | username | lastname  |     city     | state | likesports | liketheatre
-------+----------+-----------+--------------+-------+------------+-------------
1 | JSG99FHE | Taylor    | Kent         | WA    | t          | t
9 | MSD36KVR | Watkins   | Port Orford  | MD    | t          | f
```

USERS テーブルの新しいバージョンが別の方法で構築されていると想定します。この新しいバージョンでは、各ユーザーが好きまたは嫌いな 8 種類のイベントを (バイナリ形式で) 定義する 1 つの整数列です。この設計では、各ビット位置はイベントの種類を表します。8 種類すべてのタイプが好きなユーザーは、(以下のテーブルの 1 行目のように) 8 ビットすべてが 1 に設定されています。これらのどのイベントも好きではないユーザーは、8 ビットすべてに 0 が設定されます (2 行目を参照)。スポーツとジャズのみが好きなユーザーを次の 3 行目に示しています。

USER	SPORTS	THEATRE	JAZZ	OPERA	ROCK	VEGAS	BROADW	CLASSICAL
ユーザー 1	1	1	1	1	1	1	1	1
ユーザー 2	0	0	0	0	0	0	0	0
ユーザー 3	1	0	1	0	0	0	0	0

データベーステーブルでこれらのバイナリ値は、次のように整数として 1 つの LIKES 列に保存される場合があります。

ユーザー	バイナリ値	保存値 (整数)
ユーザー 1	11111111	255
ユーザー 2	00000000	0
ユーザー 3	10100000	160

BIT_AND 関数

BIT_AND 関数は、単一の整数列または式内のすべての値に対してビット単位の AND 演算を実行します。この関数は、式の整数値ごとに対応する各バイナリ値の各ビットを集計します。

値のすべてにおいてどのビットにも 1 が設定されていない場合、BIT_AND 関数は `0` の結果を返します。値のすべてにおいて 1 つ以上のビットに 1 が設定されている場合、関数は整数値を返します。この整数はこれらのビットのバイナリ値に対応する数値です。

例えば、テーブルは列に 4 つの整数値を含みます (3、7、10、および 22)。これらの整数は、次のようにバイナリで表されます。

整数	バイナリ値
3	11
7	111
10	1010
22	10110

このデータセットの BIT_AND 演算は、すべてのビットで最後から 2 番目の位置にのみ 1 が設定されていることが分かります。結果は、整数値 `00000010` を表す 2 のバイナリ値です。したがって、BIT_AND 関数は 2 を返します。

構文

```
BIT_AND ( [DISTINCT | ALL] expression )
```

引数

expression

関数の対象となる列または式。この式には、INT、INT2、または INT8 のデータ型がある必要があります。関数は同等の INT、INT2、または INT8 のデータ型を返します。

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は結果を計算する前に指定された式から重複した値をすべて削除します。引数 ALL 指定すると、この関数は重複する値をすべて保持します。ALL がデ

フォルトです。詳細については、「ビット単位の集計の DISTINCT サポート」を参照してください。

例

有効な企業情報が整数列に保存されるとすると、ビット単位関数を使用してこの情報を抽出および集計できます。次のクエリは USERLIKES と呼ばれるテーブルの LIKES 列に BIT_AND 関数を適用し、CITY 列による結果をグループ化します。

```
select city, bit_and(likes) from userlikes group by city
order by city;
city          | bit_and
--------------+---------
Los Angeles   |       0
Sacramento    |       0
San Francisco |       0
San Jose      |      64
Santa Barbara |     192
(5 rows)
```

これらの結果は次のように解釈できます。

- サンタバーバラの整数値 192 は、バイナリ値 11000000 に変換されます。つまり、この都市のすべてのユーザーがスポーツと演劇を好きですが、すべてのユーザーがその他のタイプのイベントを好きというわけではありません。
- 整数 64 は 01000000 に変換されます。したがって、サンノゼのユーザーの場合、全員が好きなイベントタイプは演劇のみです。
- 他の 3 都市の 0 の値は、「好き」と回答したユーザーがこれらの都市で 1 人もいないことを示します。

BIT_OR 関数

BIT_OR 関数は、単一の整数列または式内のすべての値に対してビット単位の OR 演算を実行します。この関数は、式の整数値ごとに対応する各バイナリ値の各ビットを集計します。

例えば、テーブルが列に 4 つの整数値を含んでいるとします (3、7、10、および 22)。これらの整数は、次のようにバイナリで表されます。

整数	バイナリ値
3	11
7	111
10	1010
22	10110

BIT_OR 関数を整数値のセットに適用すると、オペレーションは各位置で 1 が見つかった値を探します。この場合、1が少なくとも 1 つの値の最後の 5 つの位置に存在し、00011111のバイナリ結果を生成します。そのため、関数は 31 (または 16 + 8 + 4 + 2 + 1) を返します。

構文

```
BIT_OR ( [DISTINCT | ALL] expression )
```

引数

expression

関数の対象となる列または式。この式には、INT、INT2、または INT8 のデータ型がある必要があります。関数は同等の INT、INT2、または INT8 のデータ型を返します。

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は結果を計算する前に指定された式から重複した値をすべて削除します。引数 ALL 指定すると、この関数は重複する値をすべて保持します。ALL がデフォルトです。詳細については、「ビット単位の集計の DISTINCT サポート」を参照してください。

例

次のクエリは USERLIKES と呼ばれるテーブルの LIKES 列に BIT_OR 関数を適用し、CITY 列による結果をグループ化します。

```
select city, bit_or(likes) from userlikes group by city
```

```
order by city;
city          | bit_or
--------------+--------
Los Angeles   |    127
Sacramento    |    255
San Francisco |    255
San Jose      |    255
Santa Barbara |    255
(5 rows)
```

表示された 4 都市では、すべてのイベントタイプで少なくとも 1 人のユーザーが好きと答えています (255=11111111)。ロサンゼルスでは、スポーツを除くすべてのイベントタイプで「好き」と答えたユーザーが少なくとも 1 人います (127=01111111)。

BOOL_AND 関数

BOOL_AND 関数は、単一のブール値、整数値、または式に対して動作します。この関数は、BIT_AND 関数と BIT_OR 関数に同様のロジックを適用します。この関数の戻り値の型はブール値 (`true` または `false`) です。

セットの値がすべて true の場合、BOOL_AND 関数は `true` (`t`) を返します。いずれかの値が false の場合、関数は `false` (`f`) を返します。

構文

```
BOOL_AND ( [DISTINCT | ALL] expression )
```

引数

expression

関数の対象となる列または式。この式には BOOLEAN または整数のデータ型がある必要があります。関数の戻り値の型は BOOLEAN です。

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は結果を計算する前に指定された式から重複した値をすべて削除します。引数 ALL 指定すると、この関数は重複する値をすべて保持します。ALL がデフォルトです。詳細については、「ビット単位の集計の DISTINCT サポート」を参照してください。

例

ブール関数をブール式または整数式のどちらかに対して使用できます。例えば、次のクエリは複数のブール列のある TICKIT データベースの標準 USERS テーブルから結果を返します。

BOOL_AND 関数は 5 行すべてに `false` を返します。これら各州のすべてのユーザーがスポーツを好きというわけではありません。

```
select state, bool_and(likesports) from users
group by state order by state limit 5;

state | bool_and
------+---------
AB    | f
AK    | f
AL    | f
AZ    | f
BC    | f
(5 rows)
```

BOOL_OR 関数

BOOL_OR 関数は、単一のブール値、整数値、または式に対して動作します。この関数は、BIT_AND 関数と BIT_OR 関数に同様のロジックを適用します。この関数の戻り値の型はブール値 (`true`、`false`、または NULL) です。

セットの 1 つ以上の値が `true` の場合、BOOL_OR 関数は `true` (t) を返します。セットの値がすべて `false` の場合、関数は `false` (f) を返します。値が不明な場合は NULL を返すことができます。

構文

```
BOOL_OR ( [DISTINCT | ALL] expression )
```

引数

expression

　関数の対象となる列または式。この式には BOOLEAN または整数のデータ型がある必要があります。関数の戻り値の型は BOOLEAN です。

DISTINCT | ALL

引数 DISTINCT を指定すると、この関数は結果を計算する前に指定された式から重複した値をすべて削除します。引数 ALL 指定すると、この関数は重複する値をすべて保持します。ALL がデフォルトです。「ビット単位の集計の DISTINCT サポート」を参照してください。

例

ブール関数はブール式または整数式のどちらかで使用できます。例えば、次のクエリは複数のブール列のある TICKIT データベースの標準 USERS テーブルから結果を返します。

BOOL_OR 関数は 5 行すべてに `true` を返します。これらの州のそれぞれにおいて少なくとも 1 人のユーザーがスポーツを好きです。

```
select state, bool_or(likesports) from users
group by state order by state limit 5;

state | bool_or
------+--------
AB    | t
AK    | t
AL    | t
AZ    | t
BC    | t
(5 rows)
```

次の例は、NULL を返します。

```
SELECT BOOL_OR(NULL = '123')
               bool_or
------
NULL
```

条件式

トピック

- CASE 条件式
- DECODE 関数
- GREATEST および LEAST 関数

- NVL および COALESCE 関数
- NVL2 関数
- NULLIF 関数

Amazon Redshift は、SQL 標準の拡張であるいくつかの条件式をサポートしています。

CASE 条件式

CASE 式は条件式であり、他の言語で使われる if/then/else ステートメントと似ています。CASE は、複数の条件がある場合に結果を指定するために使用されます。SELECT コマンドでなど、SQL 式が有効な場合に CASE を使用してください。

2 種類の CASE 式 (簡易および検索) があります。

- 簡易 CASE 式では、式は値と比較されます。一致が検出された場合、THEN 句で指定されたアクションが適用されます。一致が検出されない場合、ELSE 句のアクションが適用されます。
- 検索 CASE 式では、CASE ごとにブール式に基づいて検証され、CASE ステートメントは最初を一致する CASE を返します。WHEN 句で検出されない場合、ELSE 句のアクションが返されます。

構文

条件を満たすために使用される簡易 CASE ステートメント

```
CASE expression
  WHEN value THEN result
  [WHEN...]
  [ELSE result]
END
```

各条件を検証するために使用する検索 CASE ステートメント

```
CASE
  WHEN condition THEN result
  [WHEN ...]
  [ELSE result]
END
```

引数

expression

列名または有効な式。

value

数値定数または文字列などの式を比較する値。

result

式またはブール条件が検証されるときに返されるターゲット値または式。すべての結果式のデータタイプは、単一の出力タイプに変換できる必要があります。

condition

true または false に評価される Boolean 式。条件が true の場合、CASE 式の値は条件に続く結果であり、残りの CASE 式は処理されません。条件が false の場合、後続の WHEN 句はすべて評価されます。WHEN 条件の結果がどれも true ではない場合、CASE 式の値が ELSE 句の結果になります。ELSE 句が省略され、どの条件も true ではない場合、結果は Null となります。

例

次の例では、サンプル TICKIT データの VENUE テーブルと SALES テーブルを使用しています。詳細については、「サンプルデータベース」を参照してください。

簡易 CASE 式を使用し、VENUE テーブルに対するクエリで `New York City` を `Big Apple` に置換します。その他すべての都市名を `other` に置換します。

```
select venuecity,
  case venuecity
    when 'New York City'
    then 'Big Apple' else 'other'
  end
from venue
order by venueid desc;

venuecity        |   case
-----------------+-----------
Los Angeles      | other
New York City    | Big Apple
San Francisco    | other
```

```
Baltimore          | other
...
```

検索 CASE 式を使用し、それぞれのチケット販売の PRICEPAID 値に基づいてグループ番号を割り当てます。

```
select pricepaid,
  case when pricepaid <10000 then 'group 1'
    when pricepaid >10000 then 'group 2'
    else 'group 3'
  end
from sales
order by 1 desc;

pricepaid |  case
----------+---------
12624     | group 2
10000     | group 3
10000     | group 3
9996      | group 1
9988      | group 1
...
```

DECODE 関数

DECODE 式は、等価条件の結果に応じて、特定の値を別の特定の値またはデフォルト値で置換します。この演算は簡易 CASE 式または IF-THEN-ELSE ステートメントの演算と同じです。

構文

```
DECODE ( expression, search, result [, search, result ]... [ ,default ] )
```

このタイプの式は、テーブルに格納されている略語またはコードを、レポートに必要な意味のあるビジネス値に置き換える場合に便利です。

パラメータ

expression

テーブル内の列など、比較する値のソース。

search (検索)

ソース式と比較されるターゲット値 (数値や文字列など)。検索式の結果は単一の固定値である必要があります。値の範囲 (`age between 20 and 29` など) を検証する式を指定することはできません。置換する値ごとに個別の検索/結果のペアを指定する必要があります。

検索式のすべてのインスタンスのデータ型は、同じまたは互換性がある必要があります。expression および search パラメータも互換性のある必要があります。

result

式が検索値を満たす場合、クエリが返す置換値。DECODE 式に少なくとも 1 の検索/結果のペアを含める必要があります。

結果式のすべてのインスタンスのデータ型は同じまたは互換性がある必要があります。result および default パラメータも互換性のある必要があります。

デフォルト

検索条件が失敗した場合に使用されるオプションのデフォルト値。デフォルト値を指定しない場合、DECODE 式は NULL を返します。

使用に関する注意事項

expression 値と search 値が両方とも NULL の場合、DECODE 結果は対応する result の値になります。関数のこの使用方法の説明については、「例」のセクションを参照してください。

このように使用する場合、DECODE は「NVL2 関数」に似ていますが、いくつかの違いがあります。その違いについては、NVL2 の使用に関する注意事項を参照してください。

例

値 `2008-06-01` が datetable の caldate 列に存在する場合、次の例ではこれが `June 1st, 2008` に置換されます。この例では、他のすべての caldate 値が NULL に置き換えられます。

```
select decode(caldate, '2008-06-01', 'June 1st, 2008')
from datetable where month='JUN' order by caldate;

case
----------------
June 1st, 2008

...
```

```
(30 rows)
```

次の例では、DECODE 式を使用して、CATEGORY テーブルの 5 つの省略された CATNAME 列を完全名に変換し、列の他の値を Unknown に変換します。

```
select catid, decode(catname,
'NHL', 'National Hockey League',
'MLB', 'Major League Baseball',
'MLS', 'Major League Soccer',
'NFL', 'National Football League',
'NBA', 'National Basketball Association',
'Unknown')
from category
order by catid;

catid  | case
-------+---------------------------------
1      | Major League Baseball
2      | National Hockey League
3      | National Football League
4      | National Basketball Association
5      | Major League Soccer
6      | Unknown
7      | Unknown
8      | Unknown
9      | Unknown
10     | Unknown
11     | Unknown
(11 rows)
```

DECODE 式を使用して、VENUESEATS 列が NULL のコロラドとネバダの会場を検索して、NULL をゼロに変換します。VENUESEATS 列が NULL でない場合は、結果として 1 を返します。

```
select venuename, venuestate, decode(venueseats,null,0,1)
from venue
where venuestate in('NV','CO')
order by 2,3,1;

venuename                | venuestate     | case
------------------------------+----------------+-----------
Coors Field                  | CO          |   1
Dick's Sporting Goods Park   | CO          |   1
Ellie Caulkins Opera House   | CO          |   1
```

```
INVESCO Field        | CO         |   1
Pepsi Center        | CO         |   1
Ballys Hotel        | NV         |   0
Bellagio Hotel                    | NV         |   0
Caesars Palace                    | NV         |   0
Harrahs Hotel                     | NV         |   0
Hilton Hotel                      | NV         |   0
...
(20 rows)
```

GREATEST および LEAST 関数

任意の数の式のリストから最大値または最小値を返します。

構文

```
GREATEST (value [, ...])
LEAST (value [, ...])
```

パラメータ

expression_list

列名などの式のカンマ区切りリスト。式はすべて一般的なデータ型に変換可能である必要があります。リスト内の NULL 値は無視されます。すべての式が NULL と評価された場合、結果は NULL になります。

戻り値

指定された式のリストから最大 (GREATEST の場合) または最小 (LEAST の場合) の値を返します。

例

次の例は、アルファベット順で最も高い `firstname` または `lastname` の値を返します。

```
select firstname, lastname, greatest(firstname,lastname) from users
where userid < 10
order by 3;

 firstname | lastname  | greatest
-----------+-----------+-----------
 Lars      | Ratliff   | Ratliff
```

```
 Reagan    | Hodge     | Reagan
 Colton    | Roy       | Roy
 Barry     | Roy       | Roy
 Tamekah   | Juarez    | Tamekah
 Rafael    | Taylor    | Taylor
 Victor    | Hernandez | Victor
 Vladimir  | Humphrey  | Vladimir
 Mufutau   | Watkins   | Watkins
(9 rows)
```

NVL および COALESCE 関数

一連の式の中で、Null 以外の最初の式の値を返します。Null 以外の値が見つかると、リスト内の残りの式は評価されません。

NVL は COALESCE と同じです。これらはシノニムです。このトピックでは、両方の構文について説明し、例を示します。

構文

```
NVL( expression, expression, ... )
```

COALESCE の構文は同じです。

```
COALESCE( expression, expression, ... )
```

すべての式が null の場合、結果は null になります。

これらの関数は、プライマリ値がないか Null の場合にセカンダリ値を返すときに役に立ちます。例えば、クエリを実行すると、使用可能な 3 つの電話番号 (携帯、自宅、職場) のうち最初の電話番号が返されることがあります。関数内の式の順序によって、評価の順序が決まります。

引数

expression

Null ステータスが評価される列名などの式。

戻り型

Amazon Redshift は、入力式に基づいて戻り値のデータ型を決定します。入力式のデータ型に共通の型がない場合は、エラーが返されます。

例

リストに整数式が含まれている場合、関数は整数を返します。

```
SELECT COALESCE(NULL, 12, NULL);

coalesce
--------------
12
```

この例は前の例と同じですが、NVL を使用して、同じ結果が返される点が異なります。

```
SELECT NVL(NULL, 12, NULL);

coalesce
--------------
12
```

次の例は、文字列型を返します。

```
SELECT COALESCE(NULL, 'Amazon Redshift', NULL);

coalesce
--------------
Amazon Redshift
```

次の例では、式リストのデータ型が異なるため、エラーになります。この場合、リストには文字列型と数値型の両方があります。

```
SELECT COALESCE(NULL, 'Amazon Redshift', 12);
ERROR: invalid input syntax for integer: "Amazon Redshift"
```

この例では、START_DATE および END_DATE 列を持つテーブルを作成し、Null 値を含む行を挿入して、NVL 式をその 2 列に適用します。

```
create table datetable (start_date date, end_date date);
insert into datetable values ('2008-06-01','2008-12-31');
insert into datetable values (null,'2008-12-31');
insert into datetable values ('2008-12-31',null);
```

```
select nvl(start_date, end_date)
```

```
from datetable
order by 1;

coalesce
------------
2008-06-01
2008-12-31
2008-12-31
```

NVL 式のデフォルトの列名は COALESCE です。次のクエリは同じ結果を返します。

```
select coalesce(start_date, end_date)
from datetable
order by 1;
```

次のクエリ例では、ホテルの予約情報のサンプルを含むテーブルを作成し、複数の行を挿入します。一部のレコードは NULL 値を含んでいます。

```
create table booking_info (booking_id int, booking_code character(8), check_in date,
 check_out date, funds_collected numeric(12,2));
```

次のサンプルデータを挿入します。一部のレコードには `check_out` 日付または `funds_collected` 数がありません。

```
insert into booking_info values (1, 'OCEAN_WV', '2023-02-01','2023-02-03',100.00);
insert into booking_info values (2, 'OCEAN_WV', '2023-04-22','2023-04-26',120.00);
insert into booking_info values (3, 'DSRT_SUN', '2023-03-13','2023-03-16',125.00);
insert into booking_info values (4, 'DSRT_SUN', '2023-06-01','2023-06-03',140.00);
insert into booking_info values (5, 'DSRT_SUN', '2023-07-10',null,null);
insert into booking_info values (6, 'OCEAN_WV', '2023-08-15',null,null);
```

次のクエリは、日付のリストを返します。`check_out` 日付が使用できない場合は、`check_in` 日付が表示されます。

```
select coalesce(check_out, check_in)
from booking_info
order by booking_id;
```

結果は、以下のとおりです。最後の 2 つのレコードには `check_in` 日付が表示されていることに注意してください。

```
coalesce
------------
2023-02-03
2023-04-26
2023-03-16
2023-06-03
2023-07-10
2023-08-15
```

クエリが特定の関数または列に対して Null 値を返すことが予想される場合は、NVL 式を使用して Null 値を他の値に置換できます。例えば、SUM などの集計関数は検証する行がない場合に、ゼロの代わりに Null 値を返します。NVL 式を使用して、これらの Null 値を `700.0` に置き換えることができます。`485` の代わりに、`funds_collected` を合計した結果が `1885` になります。これは、Null を含む 2 つの行が `700` に置き換えられるためです。

```
select sum(nvl(funds_collected, 700.0)) as sumresult from booking_info;

sumresult
------
 1885
```

NVL2 関数

指定された式の結果が NULL か NOT NULL かに基づいて、2 つの値のいずれかを返します。

構文

```
NVL2 ( expression, not_null_return_value, null_return_value )
```

引数

expression

Null ステータスが評価される列名などの式。

not_null_return_value

expression が NOT NULL に評価された場合に返される値。not_null_return_value 値は、expression と同じデータ型を持つか、そのデータ型に暗黙的に変換可能である必要があります。

null_return_value

expression が NULL に評価される場合に値が返されます。null_return_value 値は、expression と同じデータ型を持つか、そのデータ型に暗黙的に変換可能である必要があります。

戻り型

NVL2 の戻り値の型は、次のように決まります。

- not_null_return_value または null_return_value が null の場合、not-null 式のデータ型が返されます。

not_null_return_value と null_return_value がどちらも null である場合

- not_null_return_value と null_return_value のデータ型が同じ場合、そのデータ型が返されます。
- not_null_return_value と null_return_value の数値データ型が異なる場合、互換性を持つ最小の数値データ型が返されます。
- not_null_return_value と null_return_value の日時データ型が異なる場合、タイムスタンプデータ型が返されます。
- not_null_return_value と null_return_value の文字データ型が異なる場合、not_null_return_value のデータ型が返されます。
- not_null_return_value と null_return_value で数値データ型と数値以外のデータ型が混合している場合、not_null_return_value のデータ型が返されます。

> **Important**
>
> not_null_return_value のデータ型が返される最後の 2 つのケースでは、null_return_value がそのデータ型に暗黙的にキャストされます。データ型に互換性がない場合、関数は失敗します。

使用に関する注意事項

DECODE 関数 は、式と検索パラメータの両方が null の場合、NVL2 と同様の方法で使用できます。異なるのは、DECODE の場合、result パラメータの値とデータ型の両方が戻り値に含められるという点です。一方、NVL2 の場合、not_null_return_value または null_return_value パラメータ

のうち、どちらか関数により選択された方の値が戻り値に含められますが、データ型については not_null_return_value のデータ型が含められます。

例えば、column1 が NULL の場合、次のクエリは同じ値を返します。ただし、DECODE の戻り値のデータ型は INTEGER となり、NVL2 の戻り値のデータ型は VARCHAR になります。

```
select decode(column1, null, 1234, '2345');
select nvl2(column1, '2345', 1234);
```

例

次の例では、一部のサンプルデータを変更し、2 つのフィールドを評価してユーザーに適切な連絡先情報を提供します。

```
update users set email = null where firstname = 'Aphrodite' and lastname = 'Acevedo';

select (firstname + ' ' + lastname) as name,
nvl2(email, email, phone) AS contact_info
from users
where state = 'WA'
and lastname  like 'A%'
order by lastname, firstname;

name        contact_info
--------------------+-------------------------------------------
Aphrodite Acevedo (906) 632-4407
Caldwell Acevedo  Nunc.sollicitudin@Duisac.ca
Quinn Adams    vel@adipiscingligulaAenean.com
Kamal Aguilar  quis@vulputaterisusa.com
Samson Alexander  hendrerit.neque@indolorFusce.ca
Hall Alford    ac.mattis@vitaediamProin.edu
Lane Allen     et.netus@risusDonec.org
Xander Allison    ac.facilisis.facilisis@Infaucibus.com
Amaya Alvarado    dui.nec.tempus@eudui.edu
Vera Alvarez   at.arcu.Vestibulum@pellentesque.edu
Yetta Anthony  enim.sit@risus.org
Violet Arnold  ad.litora@at.com
August Ashley  consectetuer.euismod@Phasellus.com
Karyn Austin   ipsum.primis.in@Maurisblanditenim.org
Lucas Ayers    at@elitpretiumet.com
```

NULLIF 関数

構文

NULLIF 式は 2 つの引数を比較し、引数が等しい場合に Null を返します。引数が等しくない場合、最初の引数が返されます。この式は NVL または COALESCE 式の逆です。

```
NULLIF ( expression1, expression2 )
```

引数

expression1, expression2

比較対象の列または式。戻り値の型は、最初の式の型と同じです。NULLIF の結果のデフォルトの列名は、最初の式の列名です。

例

次の例では、引数が等しくないため、クエリは `first` 文字列を返します。

```
SELECT NULLIF('first', 'second');

case
-------
first
```

次の例では、文字列リテラル引数が等しいため、クエリは NULL を返します。

```
SELECT NULLIF('first', 'first');

case
-------
NULL
```

次の例では、整数の引数が等しくないため、クエリは 1 を返します。

```
SELECT NULLIF(1, 2);

case
-------
1
```

次の例では、整数の引数が等しいため、クエリは NULL を返します。

```
SELECT NULLIF(1, 1);

case
-------
NULL
```

次の例に示すクエリは、LISTID 値と SALESID 値が一致する場合に Null を返します。

```
select nullif(listid,salesid), salesid
from sales where salesid<10 order by 1, 2 desc;

listid  | salesid
--------+---------
     4  |       2
     5  |       4
     5  |       3
     6  |       5
     10 |       9
     10 |       8
     10 |       7
     10 |       6
        |       1
(9 rows)
```

NULLIF を使用し、空の文字列が常に Null として返されるようにします。以下の例では、NULLIF 式は Null 値または少なくとも 1 文字を含む文字列のどちらかを返します。

```
insert into category
values(0,'','Special','Special');

select nullif(catgroup,'') from category
where catdesc='Special';

catgroup
----------
null
(1 row)
```

NULLIF は末尾の空白を無視します。文字列が空ではないが空鶴が含まれる場合も、NULLIF は Null を返します。

```
create table nulliftest(c1 char(2), c2 char(2));

insert into nulliftest values ('a','a ');

insert into nulliftest values ('b','b');


select nullif(c1,c2) from nulliftest;
c1
------
null
null
(2 rows)
```

データ型フォーマット関数

トピック

データ型フォーマット関数を使用すると、簡単な方法で値のデータ型を変換できます。これらの各関数では必ず、最初の引数にはフォーマットする値を指定し、2 番目の引数には新しい形式のテンプレートを指定します。Amazon Redshift は、いくつかのデータ型のフォーマット関数をサポートしています。

CAST 関数

CAST 関数は、1 つのデータ型を互換性のある別のデータ型に変換します。例えば、文字列を日付に変換したり、数値型を文字列に変換したりできます。CAST はランタイム変換を実行します。つま

り、変換によってソーステーブル内の値のデータ型は変更されません。クエリのコンテキストでのみ変更されます。

CAST 関数は、あるデータ型から別のデータ型に変換するという点では the section called “CONVERT” とよく似ていますが、呼び出し方が異なります。

特定のデータ型は、CAST 関数または CONVERT 関数を使用して、他のデータ型に明示的に変換する必要があります。その他のデータ型は、CAST や CONVERT を使用せずに、別のコマンドの一部として暗黙的に変換できます。「型の互換性と変換」を参照してください。

構文

式のデータ型を別のデータ型に変換するには、次の 2 つの同等な構文フォームのいずれかを使用します。

```
CAST ( expression AS type )
expression :: type
```

引数

expression

1 つ以上の値 (列名、値など) に評価される式。null 値を変換すると、null が返されます。式に、空白または空の文字列を含めることはできません。

type

サポートされる データ型 の 1 つ。

戻り型

CAST は、type 引数で指定されたデータ型を返します。

> **Note**
>
> 次の正確性が失われる DECIMAL 変換のように、問題のある変換を実行しようとすると、Amazon Redshift はエラーを返します。
>
> ```
> select 123.456::decimal(2,1);
> ```
>
> また、次の INTEGER 変換では、オーバーフローが生じます。

```
select 12345678::smallint;
```

例

一部の例では、サンプルの TICKIT データベースを使用しています。サンプルデータの設定の詳細については、「データをロードする」を参照してください。

次の 2 つのクエリは同等です。どちらも 10 進値を整数に変換します。

```
select cast(pricepaid as integer)
from sales where salesid=100;

pricepaid
-----------
162
(1 row)
```

```
select pricepaid::integer
from sales where salesid=100;

pricepaid
-----------
162
(1 row)
```

次の場合も同様の結果が得られます。サンプルデータの実行は必要ありません。

```
select cast(162.00 as integer) as pricepaid;

pricepaid
-----------
162
(1 row)
```

次の例では、タイムスタンプ列の値を日付に変換して、各結果から時刻を削除します。

```
select cast(saletime as date), salesid
```

```
from sales order by salesid limit 10;

 saletime   | salesid
-----------+---------
2008-02-18 |       1
2008-06-06 |       2
2008-06-06 |       3
2008-06-09 |       4
2008-08-31 |       5
2008-07-16 |       6
2008-06-26 |       7
2008-07-10 |       8
2008-07-22 |       9
2008-08-06 |      10
(10 rows)
```

前のサンプルで示した CAST を使用しなかった場合、結果には 2008-02-18 02:36:48 という時刻が含まれます。

次のクエリは、可変文字データを日付に変換します。実行にサンプルデータは必要ありません。

```
select cast('2008-02-18 02:36:48' as date) as mysaletime;

mysaletime
--------------------
2008-02-18
(1 row)
```

次の例では、日付列の値をタイムスタンプに変換します。

```
select cast(caldate as timestamp), dateid
from date order by dateid limit 10;

      caldate       | dateid
--------------------+--------
2008-01-01 00:00:00 |   1827
2008-01-02 00:00:00 |   1828
2008-01-03 00:00:00 |   1829
2008-01-04 00:00:00 |   1830
2008-01-05 00:00:00 |   1831
2008-01-06 00:00:00 |   1832
2008-01-07 00:00:00 |   1833
```

```
2008-01-08 00:00:00 |    1834
2008-01-09 00:00:00 |    1835
2008-01-10 00:00:00 |    1836
(10 rows)
```

前のサンプルのような場合は、TO_CHAR を使用して出力フォーマットをさらに制御できます。

次の例では、整数を文字列に変換します。

```
select cast(2008 as char(4));

bpchar
--------
2008
```

次の例では、DECIMAL(6,3) 値を DECIMAL(4,1) 値に変換します。

```
select cast(109.652 as decimal(4,1));

numeric
---------
109.7
```

この例は、より複雑な式を示しています。SALES テーブル内の PRICEPAID 列 (DECIMAL(8,2) 列) を DECIMAL(38,2) 列に変換し、結果の値を 100000000000000000000 で乗算します。

```
select salesid, pricepaid::decimal(38,2)*100000000000000000000
as value from sales where salesid<10 order by salesid;


 salesid |           value
---------+----------------------------
       1 | 72800000000000000000000.00
       2 |  7600000000000000000000.00
       3 | 35000000000000000000000.00
       4 | 17500000000000000000000.00
       5 | 15400000000000000000000.00
       6 | 39400000000000000000000.00
       7 | 78800000000000000000000.00
       8 | 19700000000000000000000.00
       9 | 59100000000000000000000.00
```

```
(9 rows)
```

> **Note**
>
> GEOMETRY データ型に対して CAST や CONVERT オペレーションを実行して別のデータ型に変更することはできません。ただし、GEOMETRY 引数を受け入れる関数への入力として、Extended Well-Known Binary (EWKB) 形式の 16 進数の文字列リテラルを指定できます。例えば、後続の ST_AsText 関数は GEOMETRY データ型を想定します。
>
> ```
> SELECT ST_AsText('01010000000000000000001C400000000000002040');
> ```
>
> ```
> st_astext
> ------------
>  POINT(7 8)
> ```
>
> また、明示的に GEOMETRY データ型を指定することもできます。
>
> ```
> SELECT ST_AsText('010100000000000000000014400000000000001840'::geometry);
> ```
>
> ```
> st_astext
> ------------
>  POINT(5 6)
> ```

CONVERT 関数

CAST 関数と同様に、CONVERT 関数はあるデータ型を互換性のある別のデータ型に変換します。例えば、文字列を日付に変換したり、数値型を文字列に変換したりできます。CONVERT は、ランタイム変換を実行します。つまり、変換によってソーステーブルの値のデータ型は変更されません。クエリのコンテキストでのみ変更されます。

特定のデータ型は、CONVERT 関数を使用して、他のデータ型に明示的に変換する必要があります。その他のデータ型は、CAST や CONVERT を使用せずに、別のコマンドの一部として暗黙的に変換できます。「型の互換性と変換」を参照してください。

構文

```
CONVERT ( type, expression )
```

引数

type

サポートされる データ型 の 1 つ。

expression

1 つ以上の値 (列名、値など) に評価される式。null 値を変換すると、null が返されます。式に、空白または空の文字列を含めることはできません。

戻り型

CONVERT は、type 引数で指定されたデータ型を返します。

> **Note**
>
> 次の正確性が失われる DECIMAL 変換のように、問題のある変換を実行しようとすると、Amazon Redshift はエラーを返します。
>
> ```
> SELECT CONVERT(decimal(2,1), 123.456);
> ```
>
> また、次の INTEGER 変換では、オーバーフローが生じます。
>
> ```
> SELECT CONVERT(smallint, 12345678);
> ```

例

一部の例では、サンプルの TICKIT データベースを使用しています。サンプルデータの設定の詳細については、「データをロードする」を参照してください。

次のクエリは、CONVERT 関数を使用して小数の列を整数に変換します。

```
SELECT CONVERT(integer, pricepaid)
FROM sales WHERE salesid=100;
```

この例では、整数を文字列に変換します。

```
SELECT CONVERT(char(4), 2008);
```

次の例では、現在の日付と時刻を可変文字データ型に変換します。

```
SELECT CONVERT(VARCHAR(30), GETDATE());

getdate
---------
2023-02-02 04:31:16
```

この例では、販売時間の列を時刻のみに変換し、各行から日付を削除します。

```
SELECT CONVERT(time, saletime), salesid
FROM sales order by salesid limit 10;
```

タイムスタンプをあるタイムゾーンから別のタイムゾーンに変換する方法については、「CONVERT_TIMEZONE 関数」を参照してください。その他の日付および時刻関数については、「日付および時刻関数」を参照してください。

次の例では、可変文字データを datetime オブジェクトに変換します。

```
SELECT CONVERT(datetime, '2008-02-18 02:36:48') as mysaletime;
```

> **Note**
>
> GEOMETRY データ型に対して CAST や CONVERT オペレーションを実行して別のデータ型に変更することはできません。ただし、GEOMETRY 引数を受け入れる関数への入力として、Extended Well-Known Binary (EWKB) 形式の 16 進数の文字列リテラルを指定できます。例えば、後続の ST_AsText 関数は GEOMETRY データ型を想定します。
>
> ```
> SELECT ST_AsText('01010000000000000000001C400000000000002040');
> ```
>
> ```
> st_astext
> ------------
>  POINT(7 8)
> ```
>
> また、明示的に GEOMETRY データ型を指定することもできます。
>
> ```
> SELECT ST_AsText('010100000000000000000014400000000000001840'::geometry);
> ```
>
> ```
> st_astext
> ```

```
------------
 POINT(5 6)
```

TO_CHAR

TO_CHAR は、タイムスタンプまたは数値式を文字列データ形式に変換します。

構文

```
TO_CHAR (timestamp_expression | numeric_expression , 'format')
```

引数

timestamp_expression

TIMESTAMP 型または TIMESTAMPTZ 型の値、またはタイムスタンプに暗黙的に強制変換できる値を生成する式。

numeric_expression

数値データ型の値、または数値型に暗黙的に強制変換できる値が生成される式。詳細については、「数値型」を参照してください。TO_CHAR は、数文字列の左側にスペースを挿入します。

> **Note**
>
> TO_CHAR は、128 ビットの DECIMAL 値をサポートしません。

format

新しい値の形式。有効な形式については、「日時形式の文字列」および「数値形式の文字列」を参照してください。

戻り型

VARCHAR

例

次の例では、月の名前を 9 文字に埋め込み、曜日の名前と日付の数字を指定した形式でタイムスタンプを日付と時刻の値に変換します。

```
select to_char(timestamp '2009-12-31 23:15:59', 'MONTH-DY-DD-YYYY HH12:MIPM');

to_char
-------------------------
DECEMBER -THU-31-2009 11:15PM
```

次の例では、タイムスタンプを日付の番号の値に変換します。

```
select to_char(timestamp '2009-12-31 23:15:59', 'DDD');

to_char
-------------------------
365
```

次の例では、タイムスタンプを曜日の番号の値に変換します。

```
select to_char(timestamp '2022-05-16 23:15:59', 'ID');

to_char
-------------------------
1
```

以下の例では、日付から月を抽出しています。

```
select to_char(date '2009-12-31', 'MONTH');

to_char
-------------------------
DECEMBER
```

次の例は、EVENT テーブル内の各 STARTTIME 値を、時、分、および秒から成る文字列に変換します。

```
select to_char(starttime, 'HH12:MI:SS')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00
08:00:00
```

```
02:30:00
02:30:00
07:00:00
```

次の例は、タイムスタンプの値全体を別の形式に変換します。

```
select starttime, to_char(starttime, 'MON-DD-YYYY HH12:MIPM')
from event where eventid=1;

      starttime      |       to_char
---------------------+---------------------
 2008-01-25 14:30:00 | JAN-25-2008 02:30PM
```

次の例は、タイムスタンプリテラルをキャラクタ文字列に変換します。

```
select to_char(timestamp '2009-12-31 23:15:59','HH24:MI:SS');

to_char
----------
23:15:59
```

次の例では、10 進数をキャラクタ文字列に変換します。

```
select to_char(125.8, '999.99');

to_char
---------
125.80
```

次の例では、10 進数をキャラクタ文字列に変換します。

```
select to_char(125.8, '999D99');

to_char
---------
125.80
```

次の例では、先頭にゼロを含む数字をキャラクタ文字列に変換します。

```
select to_char(125.8, '0999D99');
```

```
to_char
---------
0125.80
```

次の例では、数字を負の記号を末尾に付けた文字列に変換します。

```
select to_char(-125.8, '999D99S');

to_char
---------
125.80-
```

次の例では、指定された位置に正または負の記号を末尾に付けた文字列に数字を変換します。

```
select to_char(125.8, '999D99SG');

to_char
---------
125.80+
```

次の例では、指定された位置に正の記号を末尾に付けた文字列に数字を変換します。

```
select to_char(125.8, 'PL999D99');

to_char
---------
+ 125.80
```

次の例では、数字を通貨記号付きの文字列に変換します。

```
select to_char(-125.88, '$S999D99');

to_char
---------
$-125.88
```

次の例では、指定された位置に通貨記号を付けた文字列に数字を変換します。

```
select to_char(-125.88, 'S999D99L');
```

```
to_char
---------
-125.88$
```

次の例では、3 桁の区切り文字 (コンマ) を使用して数字をキャラクタ文字列に変換します。

```
select to_char(1125.8, '9,999.99');

to_char
---------
1,125.80
```

次の例では、負の数字に角括弧を使用して、数字を文字列に変換します。

```
select to_char(-125.88, '$999D99PR');

to_char
---------
$<125.88>
```

次の例では、数字をローマ字の文字列に変換します。

```
select to_char(125, 'RN');

to_char
---------
   CXXV
```

次の例では、日付を世紀コードに変換します。

```
select to_char(date '2020-12-31', 'CC');

to_char
---------
21
```

次の例では、曜日を表示します。

```
SELECT to_char(current_timestamp, 'FMDay, FMDD HH12:MI:SS');
```

```
to_char
-----------------------
Wednesday, 31 09:34:26
```

次の例では、数に対して序数のサフィックスを表示します。

```
SELECT to_char(482, '999th');

to_char
-----------------------
 482nd
```

次の例では、販売テーブル内の支払い価格からコミッションを減算します。誤差は四捨五入されて、ローマ数字に変換され、to_char 列に表示されます。

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'rn') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;

 salesid | pricepaid | commission | difference |     to_char
---------+-----------+------------+------------+-----------------
       1 |    728.00 |     109.20 |     618.80 |           dcxix
       2 |     76.00 |      11.40 |      64.60 |             lxv
       3 |    350.00 |      52.50 |     297.50 |        ccxcviii
       4 |    175.00 |      26.25 |     148.75 |           cxlix
       5 |    154.00 |      23.10 |     130.90 |           cxxxi
       6 |    394.00 |      59.10 |     334.90 |         cccxxxv
       7 |    788.00 |     118.20 |     669.80 |           dclxx
       8 |    197.00 |      29.55 |     167.45 |          clxvii
       9 |    591.00 |      88.65 |     502.35 |             dii
      10 |     65.00 |       9.75 |      55.25 |              lv
```

次の例では、さまざまな値に通貨記号を追加して、to_char 列に表示します。

```
select salesid, pricepaid, commission, (pricepaid - commission)
as difference, to_char(pricepaid - commission, 'l99999D99') from sales
group by sales.pricepaid, sales.commission, salesid
order by salesid limit 10;
```

```
salesid | pricepaid | commission | difference |  to_char
--------+-----------+------------+------------+------------
      1 |    728.00 |     109.20 |     618.80 | $   618.80
      2 |     76.00 |      11.40 |      64.60 | $    64.60
      3 |    350.00 |      52.50 |     297.50 | $   297.50
      4 |    175.00 |      26.25 |     148.75 | $   148.75
      5 |    154.00 |      23.10 |     130.90 | $   130.90
      6 |    394.00 |      59.10 |     334.90 | $   334.90
      7 |    788.00 |     118.20 |     669.80 | $   669.80
      8 |    197.00 |      29.55 |     167.45 | $   167.45
      9 |    591.00 |      88.65 |     502.35 | $   502.35
     10 |     65.00 |       9.75 |      55.25 | $    55.25
```

次の例では、各販売が行われた世紀を一覧で示します。

```
select salesid, saletime, to_char(saletime, 'cc') from sales
order by salesid limit 10;

 salesid |      saletime       | to_char
---------+---------------------+---------
       1 | 2008-02-18 02:36:48 | 21
       2 | 2008-06-06 05:00:16 | 21
       3 | 2008-06-06 08:26:17 | 21
       4 | 2008-06-09 08:38:52 | 21
       5 | 2008-08-31 09:17:02 | 21
       6 | 2008-07-16 11:59:24 | 21
       7 | 2008-06-26 12:56:06 | 21
       8 | 2008-07-10 02:12:36 | 21
       9 | 2008-07-22 02:23:17 | 21
      10 | 2008-08-06 02:51:55 | 21
```

次の例は、EVENT テーブル内の各 STARTTIME 値を、時、分、秒、およびタイムゾーンから成る文字列に変換します。

```
select to_char(starttime, 'HH12:MI:SS TZ')
from event where eventid between 1 and 5
order by eventid;

to_char
----------
02:30:00 UTC
08:00:00 UTC
02:30:00 UTC
```

```
02:30:00 UTC
07:00:00 UTC
```

以下の例では、秒、ミリ秒、マイクロ秒の形式を示しています。

```
select sysdate,
to_char(sysdate, 'HH24:MI:SS') as seconds,
to_char(sysdate, 'HH24:MI:SS.MS') as milliseconds,
to_char(sysdate, 'HH24:MI:SS:US') as microseconds;

timestamp           | seconds  | milliseconds | microseconds
--------------------+----------+--------------+----------------
2015-04-10 18:45:09 | 18:45:09 | 18:45:09.325 | 18:45:09:325143
```

TO_DATE 関数

TO_DATE は、文字列で表記された日付を DATE データ型に変換します。

> **Note**
>
> TO_DATE は、Q (四半期番号) の形式の文字列をサポートしていません。

構文

```
TO_DATE(string, format)
```

```
TO_DATE(string, format, is_strict)
```

引数

string

変換する文字列。

format

入力の文字列をその日付部分に基づいて定義する文字列リテラル。有効な日、月、年の形式一覧については、「日時形式の文字列」を参照してください。

is_strict

入力日付値が範囲外である場合にエラーを返すかどうかを指定するオプションのブール値。is_strict が TRUE に設定されている場合、範囲外の値があるとエラーが返されます。is_strict がデフォルトの FALSE に設定されている場合、オーバーフロー値が受け入れられます。

戻り型

TO_DATE は、format の値に応じて DATE を返します。

フォーマットへの変換が失敗すると、エラーが返されます。

例

次の SQL ステートメントは、日付 02 Oct 2001 を日付データ型に変換します。

```
select to_date('02 Oct 2001', 'DD Mon YYYY');

to_date
------------
2001-10-02
(1 row)
```

次の SQL ステートメントは、文字列 20010631 を日付に変換します。

```
select to_date('20010631', 'YYYYMMDD', FALSE);
```

結果は 2001 年 7 月 1 日です。これは、6 月が 30 日しかないためです。

```
to_date
------------
2001-07-01
```

次の SQL ステートメントは、文字列 20010631 を日付に変換します。

```
to_date('20010631', 'YYYYMMDD', TRUE);
```

結果はエラーになります。これは、6 月が 30 日しかないためです。

```
ERROR:  date/time field date value out of range: 2001-6-31
```

TO_NUMBER

TO_NUMBER は、文字列を数値 (10 進) に変換します。

> **Note**
> パディングの空白とゼロを抑制するには、フォーマット文字列で `FM` を使用することをお勧めします。有効な形式の一覧については、「数値形式の文字列」を参照してください。

構文

```
to_number(string, format)
```

引数

string

変換する文字列。形式はリテラル値である必要があります。

format

2 番目の引数は、数値を作成するために文字列を解析する方法を示す書式文字列です。例えば、形式 `'FM99D999'` では、変換する文字列が 5 つの数字で構成され、3 番目の位置に小数点が挿入されます。たとえば、`to_number('12.345','FM99D999')`は数値として `12.345` を返します。有効な形式の一覧については、「数値形式の文字列」を参照してください。

戻り型

TO_NUMBER は DECIMAL 型の数値を返します。

フォーマットへの変換が失敗すると、エラーが返されます。

例

次の例では、文字列 `12,454.8-` を数値に変換します。

```
select to_number('12,454.8-', 'FM99G999D9S');
```

```
to_number
-----------
-12454.8
```

次の例では、文字列 `$ 12,454.88` を数値に変換します。

```
select to_number('$ 12,454.88', 'FML99G999D99');

to_number
-----------
12454.88
```

次の例では、文字列 `$ 2,012,454.88` を数値に変換します。

```
select to_number('$ 2,012,454.88', 'FML9,999,999.99');

to_number
-----------
2012454.88
```

TEXT_TO_INT_ALT

TEXT_TO_INT_ALT は、Teradata スタイルの形式を使用して文字列を整数に変換します。結果の小数桁は切り捨てられます。

構文

```
TEXT_TO_INT_ALT (expression [ , 'format'])
```

引数

expression

1 つ以上の CHAR 値または VARCHAR 値 (列名、リテラル文字列など) になる式。null 値を変換すると、null が返されます。この関数は、空白または空の文字列を 0 に変換します。

format

入力式の形式を定義する文字列リテラル。指定できるフォーマット文字の詳細については、「数値データの Teradata スタイルの書式文字」を参照してください。

戻り型

TEXT_TO_INT_ALT は INTEGER 値を返します。

キャスト結果の小数部分は切り捨てられます。

指定した形式のフレーズへの変換に成功できなかった場合、Amazon Redshift はエラーを返します。

例

次の例では、入力式の文字列 '123-' を整数 -123 に変換します。

```
select text_to_int_alt('123-');
```

```
text_to_int_alt
----------
      -123
```

次の例では、入力式の文字列 '2147483647+' を整数 2147483647 に変換します。

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
----------
2147483647
```

次の例は、指数入力式の文字列 '-123E-2' を整数 -1 に変換します。

```
select text_to_int_alt('-123E-2');
```

```
text_to_int_alt
----------
        -1
```

次の例では、入力式の文字列 '2147483647+' を整数 2147483647 に変換します。

```
select text_to_int_alt('2147483647+');
```

```
text_to_int_alt
----------
2147483647
```

次の例では、形式フレーズ '999S' を含む入力式の文字列 '123' を整数 1,230 に変換します。S 文字は、符号付きゾーン 10 進数を示します。詳細については、「数値データの Teradata スタイルの書式文字」を参照してください。

```
select text_to_int_alt('123{', '999S');
```

```
text_to_int_alt
----------
      1230
```

次の例では、形式フレーズ「C9(I)」を含む入力式の文字列 'USD123' を整数 123 に変換します。「数値データの Teradata スタイルの書式文字」を参照してください。

```
select text_to_int_alt('USD123', 'C9(I)');
```

```
text_to_int_alt
----------
       123
```

次の例では、入力式としてテーブル列を指定しています。

```
select text_to_int_alt(a), text_to_int_alt(b) from t_text2int order by 1;
```

```
 text_to_int_alt | text_to_int_alt
-----------------+-----------------
            -123 |            -123
            -123 |            -123
             123 |             123
             123 |             123
```

以下は、この例のテーブル定義と挿入のステートメントです。

```
create table t_text2int (a varchar(200), b char(200));
```

```
insert into t_text2int VALUES('123', '123'),('123.123', '123.123'), ('-123', '-123'),
 ('123-', '123-');
```

TEXT_TO_NUMERIC_ALT

TEXT_TO_NUMERIC_ALT は、Teradata 形式のキャストオペレーションを実行して、文字列を数値データ形式に変換します。

構文

```
TEXT_TO_NUMERIC_ALT (expression [, 'format'] [, precision, scale])
```

引数

expression

1 つ以上の CHAR 値または VARCHAR 値 (列名、リテラルなど) に評価される式。null 値を変換すると、null が返されます。空白または空の文字列は 0 に変換されます。

format

入力式の形式を定義する文字列リテラル。詳細については、「数値データの Teradata スタイルの書式文字」を参照してください。

precision

数値結果の桁数。デフォルトは 38 です。

scale

数値結果の小数点の右側の桁数です。デフォルトは 0 です。

戻り型

TEXT_TO_NUMERIC_ALT は、10 進数値を返します。

指定した形式のフレーズへの変換に成功できなかった場合、Amazon Redshift はエラーを返します。

Amazon Redshift は、入力式の文字列を、精度オプションでその型に指定した最高の精度で数値型にキャストします。数値の長さが精度のために指定した値を超える場合、Amazon Redshift は次のルールに従って数値を四捨五入します。

- キャスト結果の長さが形式フレーズで指定した長さを超える場合、Amazon Redshift はエラーを返します。
- 結果が数値にキャストされる場合、結果は最も近い値に四捨五入されます。小数部分がキャスト結果のちょうど真ん中にある場合、結果は最も近い偶数値に四捨五入されます。

例

次の例では、入力式の文字列 '1.5' を数値 '2' に変換します。ステートメントはスケールを指定しないため、スケールはデフォルトで 0 になり、キャスト結果には分数の結果が含まれません。.5 は 1 と 2 の中間であるため、キャスト結果は偶数値 2 に四捨五入されます。

```
select text_to_numeric_alt('1.5');
```

```
 text_to_numeric_alt
---------------------
                   2
```

次の例では、入力式の文字列 '2.51' を数値 '3' に変換します。ステートメントはスケール値を指定しないため、スケールはデフォルトで 0 になり、キャスト結果には分数の結果が含まれません。.51 は 2 より 3 に近いので、キャスト結果は 3 の値に四捨五入されます。

```
select text_to_numeric_alt('2.51');
```

```
 text_to_numeric_alt
---------------------
                   3
```

次の例では、精度 10、スケール 2 の入力式で文字列 123.52501 を数値 123.53 に変換します。

```
select text_to_numeric_alt('123.52501', 10, 2);
```

```
 text_to_numeric_alt
---------------------
              123.53
```

次の例では、形式フレーズ '999S' を含む入力式の文字列 '123{' を数値 1230 に変換します。S 文字は、符号付きゾーン 10 進数を示します。詳細については、「数値データの Teradata スタイルの書式文字」を参照してください。

```
select text_to_numeric_alt('123{', '999S');
```

```
text_to_int_alt
----------
      1230
```

次の例では、形式フレーズ 'C9(I)' を含む入力式の文字列 'USD123' を数値 124 に変換します。「数値データの Teradata スタイルの書式文字」を参照してください。

```
select text_to_numeric_alt('USD123.9', 'C9(I)');
```

```
text_to_numeric_alt
----------
       124
```

次の例では、入力式としてテーブル列を指定しています。

```
select text_to_numeric_alt(a), text_to_numeric_alt(b) from t_text2numeric order by 1;
```

```
           text_to_numeric_alt           |           text_to_numeric_alt
-----------------------------------------+-----------------------------------------
 -99999999999999999999999999999999999999 | -99999999999999999999999999999999999999
                                  -12300 |                                  -12300
                                     123 |                                     123
                                     123 |                                     123
  99999999999999999999999999999999999999 |  99999999999999999999999999999999999999
```

以下は、この例のテーブル定義と挿入のステートメントです。

```
create table  t_text2numeric (a varchar(200), b char(200));
```

```
insert into  t_text2numeric values
('123', '123'),
('+123.456', '+123.456'),
('-' || repeat('9', 38), '-' || repeat('9', 38)),
(repeat('9', 38) || '+', repeat('9', 38) || '+'),
('-123E2', '-123E2');
```

日時形式の文字列

以下の日時形式の文字列のリファレンスを参照してください。

以下の形式の文字列は、TO_CHAR などの関数に適用されます。これらの文字列には、日時区切り記号 ('-'、'/'、':' など)、および次の「日付部分」と「時間部分」を含めることができます。

日付部分または時刻部分	意味
BC または B.C.、AD または A.D.、b.c. または bc、ad または a.d。	大文字および小文字の時代インジケータ
CC	2 桁の世紀数
YYYY、YYY、YY、Y	4 桁、3 桁、2 桁、1 桁の年数
Y,YYY	カンマ付きの 4 桁の年番号
IYYY、IYY、IY、I	4 桁、3 桁、2 桁、1 桁の International Organization for Standardization (ISO) 年番号
Q	四半期番号 (1 ~ 4)
MONTH、Month、month	月名 (大文字、大小混合文字、小文字、空白が埋め込まれて 9 文字になる)
MON、Mon、mon	月の略称 (大文字、大小混合文字、小文字、空白が埋め込まれて 3 文字になる)
MM	月番号 (01 ~ 12)
RM、rm	ローマ数字の月番号 (I ~ XII。大文字小文字を問わず I は 1 月)
W	月初からの週 (1 ~ 5。第 1 週はその月の 1 日目から始まる)
WW	年始からの週番号 (1 ~ 53。第 1 週は、その年の 1 日目から始まる)

日付部分または時刻部分	意味
IW	年始からの ISO 週番号 (新年の最初の木曜日が第 1 週になる)
DAY、Day、day	日の名前 (大文字、大小混合文字、小文字、空白が埋め込まれて 9 文字になる)
DY、Dy、dy	日の略称 (大文字、大小混合文字、小文字、空白が埋め込まれて 3 文字になる)
DDD	年始からの日 (001～366)
IDDD	ISO 8601 週番号年の日 (001-371。1 年の第 1 日は最初の ISO 週の月曜日になる)
DD	日にちを数字表示 (01～31)
D	週初めからの日 (1～7、日曜日が 1) **Note** 日付部分の D は、日時関数 DATE_PART および EXTRACT に使用される日付部分 day of week (DOW) とは動作が異なります。DOW は、整数 0～6 (日曜日が 0) に基づきます。詳細については、「日付関数またはタイムスタンプ関数の日付部分」を参照してください。
ID	ISO 8601 曜日。月曜日 (1) から日曜日 (7)
J	ユリウス日 (紀元前 4712 年 1 月 1 日からの日数)
HH24	時 (24 時間制、00～23)
HH または HH12	時 (12 時間制、01～12)

日付部分または時刻部分	意味
MI	分 (00～59)
SS	秒 (00～59)
MS	ミリ秒 (.000)
US	マイクロ秒 (.000000)
AM または PM、A.M. または P.M.、a.m. または p.m.、am または pm	大文字および小文字の午前/午後のインジケータ (12 時間制)
TZ、tz	大文字および小文字のタイムゾーンの省略形。TIMESTAMPTZ でのみ有効
OF	UTC からのオフセット。TIMESTAMPTZ でのみ有効

> **Note**
> 日時の区切り文字は一重引用符で囲む (例: '-'、'/'、':' など) 必要がありますが、前の表に示されている「日付部分」と「時間部分」は二重引用符で囲む必要があります。

例

日付を文字列の形式にする例については、「TO_CHAR」を参照してください。

数値形式の文字列

以下の数値形式の文字列のリファレンスを参照してください。

次の形式の文字列は、TO_NUMBER や TO_CHAR などの関数に適用されます。

- 文字列を数値の形式にする例については、「TO_NUMBER」を参照してください。
- 数字をも文字列の形式にする例については、「TO_CHAR」を参照してください。

形式	説明
9	指定された桁数の数値。
0	先頭に 0 が付いた数値。
. (ピリオド)、D	小数点。
, (カンマ)	桁区切り文字。
CC	世紀コード。例えば、21 世紀は 2001-01-01 から始まる (TO_CHAR のみでサポートされる)。
FM	フルモード。パディングとして使用されている空白とゼロを非表示にします。
PR	山括弧で囲まれた負の値。
S	数値にアンカーされる符号。
L	指定位置に挿入される貨幣記号。
G	グループ区切り文字。
MI	0 未満の数値の指定位置に挿入される負符号。
PL	0 より大きい数値の指定位置に挿入される正符号。
SG	指定位置に挿入される正または負符号。
RN	1～3999 のローマ数字 (TO_CHAR のみでサポートされる)。
TH または th	序数のサフィックス。0 未満の小数は変換されません。

数値データの Teradata スタイルの書式文字

以下に、TEXT_TO_INT_ALT 関数と TEXT_TO_NUMERIC_ALT 関数が入力式文字列の文字をどのように解釈するかを示します。形式フレーズで指定できる文字のリストもあります。さらに、形式オプションの Teradata スタイルの形式と Amazon Redshift の違いに関する説明を確認できます。

形式	説明
G	入力式文字列のグループ区切り文字としてはサポートされていません。この文字を形式のフレーズで指定することはできません。
D	基数記号。この文字は形式フレーズで指定できます。この文字は . (ピリオド) と同等です。 基数記号は、次の文字のいずれかを含む形式フレーズでは表示できません。 • . (ピリオド) • S (大文字 's') • V (大文字 'v')
/ , : %	挿入文字は、/ (スラッシュ)、コンマ (,)、: (コロン)、% (パーセント記号) です。 これらの文字を形式フレーズに含めることはできません。 Amazon Redshift は、入力式文字列内のこれらの文字を無視します。
.	基数文字としてのピリオド、つまり小数点です。 この文字は、次の文字のいずれかを含む形式のフレーズでは表示できません。 • D (大文字 'd') • S (大文字 's')

形式	説明
	• V (大文字 'v')
B	形式フレーズに空白文字 (B) を使用することはできません。入力式の文字列では、先頭と末尾のスペースは無視され、数字の間にスペースを使用することはできません。
+ -	形式フレーズにプラス記号 (+) またはマイナス記号 (-) を含めることはできません。ただし、入力式の文字列にプラス記号 (+) とマイナス記号 (-) が含まれている場合、数値の一部として暗黙的に解析されます。
V	小数点位置インジケータ。 この文字は、次の文字のいずれかを含む形式のフレーズでは表示できません。 • D (大文字 'd') • . (ピリオド)
Z	0 で省略された 10 進数。Amazon Redshift は、先頭に 0 をトリミングします。Z 文字は 9 文字の後には使用できません。小数部に 9 文字が含まれている場合、Z 文字は基数文字の左側にある必要があります。
9	10 進数。

形式	説明
CHAR(n)	この形式では、次のように指定できます。 • CHAR は Z または 9 文字で構成されます。Amazon Redshift では、CHAR 値に + (プラス) または - (マイナス) は使用できません。 • n は整数定数、I、または F です。I の場合、これは数値または整数データの整数部分を表示するために必要な文字数です。F の場合、数値データの小数部分を表示するために必要な文字数です。
-	ハイフン (-) 文字。 この文字を形式フレーズで使用することはできません。 Amazon Redshift は、入力式文字列のこの文字を無視します。

形式	説明
S	符号付きゾーンの 10 進数。S 文字は、形式フレーズの最後の 10 進数に続く必要があります。入力式文字列の最後の文字と対応する数値変換が 符号付きゾーン 10 進数、Teradata スタイルの数値データ形式のデータ形式文字 にリストされています。 S 文字は、次の文字のいずれかを含む形式のフレーズでは表示できません。 • + (プラス記号) • . (ピリオド) • D (大文字 'd') • Z (大文字 'z') • F (大文字 'f') • E (大文字 'e')
E	指数表記法。入力式の文字列には、指数文字を使用できます。形式フレーズの指数文字として E を指定することはできません。
FN9	Amazon Redshift ではサポートされていません。
FNE	Amazon Redshift ではサポートされていません。

形式	説明
$、USD、米ドル	ドル記号 ($)、ISO 通貨記号 (USD)、および通貨名 US ドル。 ISO 通貨記号 USD と通貨名 US ドルでは、大文字と小文字が区別されます。Amazon Redshift は USD 通貨のみをサポートします。入力式の文字列には、USD の通貨記号と数値の間にスペースを含めることができます (例: ‘$ 123E2’ または ‘123E2 $’)。
L	通貨記号。この通貨記号の文字は、形式のフレーズに 1 回だけ表示できます。繰り返される通貨記号の文字を指定することはできません。
C	ISO 通貨記号。この通貨記号の文字は、形式のフレーズに 1 回だけ表示できます。繰り返される通貨記号の文字を指定することはできません。
N	完全な通貨名。この通貨記号の文字は、形式のフレーズに 1 回だけ表示できます。繰り返される通貨記号の文字を指定することはできません。
O	二重通貨記号。この文字を形式のフレーズで指定することはできません。
U	二重 ISO 通貨記号。この文字を形式のフレーズで指定することはできません。
A	完全な二重通貨名。この文字を形式のフレーズで指定することはできません。

符号付きゾーン 10 進数、Teradata スタイルの数値データ形式のデータ形式文字

符号付きゾーンの 10 進数値には、TEXT_TO_INT_ALT 関数および TEXT_TO_NUMERIC_ALT 関数の形式句で次の文字を使用できます。

入力文字列の最後の文字	数値変換
{または 0	n … 0
A または 1	n … 1
B または 2	n … 2
C または 3	n … 3
D または 4	n … 4
E または 5	n … 5
F または 6	n … 6
G または 7	n … 7
H または 8	n … 8
I または 9	n … 9
}	-n … 0
J	-n … 1
K USD	-n … 2
L	-n … 3
M	-n … 4
N	-n … 5
O	-n … 6
P	-n … 7

入力文字列の最後の文字	数値変換
Q	-n … 8
R	-n … 9

日付および時刻関数

このセクションでは、Amazon Redshift がサポートする日付と時刻のスカラー関数についての情報を示します。

トピック

日付と時刻関数の概要

関数	構文	戻り値
+ (連結) 演算子 + 記号の両側のいずれかの方で日付を時刻に連結し、TIMESTAMP または TIMESTAMPTZ を返します。	date + time	`TIMESTAMP`、または `TIMESTAMPTZ`
ADD_MONTHS 日付またはタイムスタンプに、指定された月数を追加します。	ADD_MONTHS ({date|timestamp}, integer)	`TIMESTAMP`
AT TIME ZONE TIMESTAMP 式または TIMESTAMPTZ 式で使用するタイムゾーンを指定します。	AT TIME ZONE 'timezone'	`TIMESTAMP`、または `TIMESTAMPTZ`

関数	構文	戻り値
CONVERT_TIMEZONE タイムスタンプのタイムゾーンを別のタイムゾーンに変換します。	CONVERT_TIMEZONE (['timezone',] 'timezone', timestamp)	TIMESTAMP
CURRENT_DATE 現在のトランザクションの開始時の日付を、現在のセッションのタイムゾーン (デフォルトは UTC) で返します。	CURRENT_DATE	DATE
DATE_CMP 2 つの日付を比較し、日付が同一である場合は 0、date1 が大きい場合は 1、date2 が大きい場合は -1 を返します。	DATE_CMP (date1, date2)	INTEGER
DATE_CMP_TIMESTAMP 日付を時刻と比較し、値が同一の場合は 0、date が大きい場合は 1、timestamp が大きい場合は -1 を返します。	DATE_CMP_TIMESTAMP (date, timestamp)	INTEGER
DATE_CMP_TIMESTAMPTZ 日付とタイムゾーン付きのタイムスタンプを比較し、値が同一の場合は 0、date が大きい場合は 1、timestamptz が大きい場合は -1 を返します。	DATE_CMP_TIMESTAMPTZ (date, timestamptz)	INTEGER
DATE_PART_YEAR 日付から年を抽出します。	DATE_PART_YEAR (date)	INTEGER
DATEADD 指定された間隔で日付または時刻を増分します。	DATEADD (datepart, interval, {date|time|timetz|timestamp})	TIMESTAMP または TIME または TIMETZ

関数	構文	戻り値
DATEDIFF 日または月などの特定の日付部分の 2 つの日付または時刻の差を返します。	DATEDIFF (datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp})	BIGINT
DATE_PART 日付または時刻から日付部分の値を抽出します。	DATE_PART (datepart, {date|timestamp})	DOUBLE
DATE_TRUNC 日付部分に基づいてタイムスタンプを切り捨てます。	DATE_TRUNC ('datepart', timestamp)	TIMESTAMP
EXTRACT timestamp、timestamptz、time、または timetz から日付または時刻部分を抽出します。	EXTRACT (datepart FROM source)	INTEGER or DOUBLE
GETDATE 現在のセッションのタイムゾーン (デフォルトでは UTC) で現在の日付と時刻を返します。かっこが必要です。	GETDATE()	TIMESTAMP
INTERVAL_CMP 2 つの間隔を比較し、間隔が等しい場合は 0、interval1 が大きい場合は 1、interval2 が大きい場合は -1 を返します。	INTERVAL_CMP (interval1, interval2)	INTEGER
LAST_DAY date を含む月の最終日の日付を返します。	LAST_DAY(date)	DATE
MONTHS_BETWEEN 2 つの日付の間の月数を返します。	MONTHS_BETWEEN (date, date)	FLOAT8

関数	構文	戻り値
NEXT_DAY 指定の日付より後に指定の曜日となる最初のインスタンスの日付を返します。	NEXT_DAY (date, day)	DATE
SYSDATE 現在のトランザクション開始時の日付と時刻 (UTC) を返します。	SYSDATE	TIMESTAMP
TIMEOFDAY 現在のセッションのタイムゾーン (デフォルトでは UTC) で、現在の曜日、日付、時刻を文字列値として返します。	TIMEOFDAY()	VARCHAR
TIMESTAMP_CMP 2 つのタイムスタンプを比較し、タイムスタンプが等しい場合は `0`、timestamp1 が大きい場合は `1`、timestamp2 が大きい場合は `-1` を返します。	TIMESTAMP_CMP (timestamp1, timestamp2)	INTEGER
TIMESTAMP_CMP_DATE タイムスタンプと日付を比較し、値が同一の場合は `0`、timestamp が大きい場合は `1`、date が大きい場合は `-1` を返します。	TIMESTAMP_CMP_DATE (timestamp, date)	INTEGER
TIMESTAMP_CMP_TIMESTAMPTZ タイムスタンプをタイムゾーン付きのタイムスタンプと比較し、値が等しい場合は `0`、timestamp が大きい場合は `1`、timestamptz が大きい場合は `-1` を返します。	TIMESTAMP_CMP_TIMESTAMPTZ (timestamp, timestamptz)	INTEGER

関数	構文	戻り値
[TIMESTAMPTZ_CMP](#) タイムゾーン付きの 2 つのタイムスタンプの値を比較し、値が等しい場合は `0`、timestamptz1 がより大きい場合は `1`、timestamptz2 がより大きい場合は `-1` を返します。	TIMESTAMPTZ_CMP (timestamptz1, timestamptz2)	`INTEGER`
[TIMESTAMPTZ_CMP_DATE](#) タイムゾーン付きのタイムスタンプの値と日付を比較し、値が等しい場合は `0`、timestamptz が大きい場合は `1`、date が大きい場合は `-1` を返します。	TIMESTAMPTZ_CMP_DATE (timestamptz, date)	`INTEGER`
[TIMESTAMPTZ_CMP_TIMESTAMP](#) タイムゾーン付きのタイムスタンプをタイムスタンプと比較し、値が等しい場合は `0`、timestamptz が大きい場合は `1`、timestamp が大きい場合は `-1` を返します。	TIMESTAMPTZ_CMP_TIMESTAMP (timestamptz, timestamp)	`INTEGER`
[TIMEZONE](#) 指定されたタイムゾーンのタイムスタンプとタイムスタンプ値を返します。	TIMEZONE ('timezone' { timestamp | timestamptz)	`TIMESTAMP`、または `TIMESTAMPTZ`
[TO_TIMESTAMP](#) 指定されたタイムスタンプのタイムゾーンを含むタイムスタンプとタイムゾーン形式を返します。	TO_TIMESTAMP ('timestamp', 'format')	`TIMESTAMPTZ`
[TRUNC](#) タイムスタンプを切り捨て、日付を返します。	TRUNC(timestamp)	`DATE`

> **Note**
>
> うるう秒は経過時間の計算では考慮されません。

トランザクションにおける日付および時刻関数

トランザクションブロック (BEGIN … END) 内で次の関数を実行すると、関数は現在のステートメントではなく、現在のトランザクションの開始日または開始時刻を返します。

- SYSDATE
- TIMESTAMP
- CURRENT_DATE

次の関数は、トランザクションブロック内にある場合でも、現在のステートメントの開始日または開始時刻を常に返します。

- GETDATE
- TIMEOFDAY

廃止されたリーダーノード専用の関数

次の日付関数は、リーダーノードのみで実行されるため、非推奨となりました。詳細については、「リーダーノード専用関数」を参照してください。

- AGE。代わりに DATEDIFF 関数 を使用します。
- CURRENT_TIME。代わりに GETDATE 関数 または SYSDATE を使用します。
- CURRENT_TIMESTAMP。代わりに GETDATE 関数 または SYSDATE を使用します。
- LOCALTIME。代わりに GETDATE 関数 または SYSDATE を使用します。
- LOCALTIMESTAMP。代わりに GETDATE 関数 または SYSDATE を使用します。
- ISFINITE
- NOW。代わりに GETDATE 関数 または SYSDATE を使用します。

+ (連結) 演算子

+ 記号の両側のいずれかの方で DATE を TIME または TIMETZ に連結し、TIMESTAMP または TIMESTAMPTZ を返します。

構文

```
date + {time | timetz}
```

引数の順序は逆にすることができます。例: time + date。

引数

date

データ型 DATE の列または DATE 型に暗黙的に評価される式。

time

データ型 TIME の列または TIME 型に暗黙的に評価される式。

timetz

データ型 TIMETZ の列または TIMETZ 型に暗黙的に評価される式。

戻り型

入力が date + time の場合は TIMESTAMP です。

入力が date + timetz の場合は TIMESTAMPTZ です。

例

セットアップ例

例で使用されている TIME_TEST テーブルと TIMEZ_TEST テーブルをセットアップするには、次のコマンドを使用します。

```
create table time_test(time_val time);

insert into time_test values
('20:00:00'),
('00:00:00.5550'),
```

```
('00:58:00');

create table timetz_test(timetz_val timetz);

insert into timetz_test values
('04:00:00+00'),
('00:00:00.5550+00'),
('05:58:00+00');
```

時間列の例

次のテーブルの TIME_TEST の例には、3 つの値が挿入された列 TIME_VAL (タイプ TIME) があります。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

次の例では、日付リテラルと TIME_VAL 列を連結します。

```
select date '2000-01-02' + time_val as ts from time_test;

ts
---------------------
2000-01-02 20:00:00
2000-01-02 00:00:00.5550
2000-01-02 00:58:00
```

次の例では、日付リテラルと時刻リテラルを連結します。

```
select date '2000-01-01' + time '20:00:00' as ts;

         ts
---------------------
 2000-01-01 20:00:00
```

次の例では、時刻リテラルと日付リテラルを連結します。

```
select time '20:00:00' + date '2000-01-01' as ts;

         ts
---------------------
 2000-01-01 20:00:00
```

TIMETZ 列の例

次のテーブルの TIMETZ_TEST の例には、3 つの値が挿入された列 TIMETZ_VAL (タイプ TIMETZ) があります。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

次の例では、日付リテラルと TIMETZ_VAL 列を連結します。

```
select date '2000-01-01' + timetz_val as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

次の例では、TIMETZ_VAL 列と date リテラルを連結します。

```
select timetz_val + date '2000-01-01' as ts from timetz_test;
ts
---------------------
2000-01-01 04:00:00+00
2000-01-01 00:00:00.5550+00
2000-01-01 05:58:00+00
```

次の例では、DATE リテラルと TIMETZ 列を連結します。この例では、デフォルトで UTC タイムゾーンにある TIMESTAMPTZ が返されます。UTC は PST よりも 8 時間進んでいるため、結果は入力時間より 8 時間進みます。

```
select date '2000-01-01' + timetz '20:00:00 PST' as ts;

           ts
------------------------
 2000-01-02 04:00:00+00
```

ADD_MONTHS 関数

ADD_MONTHS は日付またはタイムスタンプの値または式に、指定された月数を加算します。DATEADD 関数は同様の機能を提供します。

構文

```
ADD_MONTHS( {date | timestamp}, integer)
```

引数

date | timestamp

データ型 DATE または TIMESTAMP の列、あるいは DATE 型または TIMESTAMP 型に暗黙的に評価される式。date がその月の最終日である場合、または結果の月が短い場合、関数は結果に月の最終日を返します。その他の日付の場合、結果には date 式と同じ日数が含まれます。

integer

データ型 INTEGER の値。負の数を使用し、日付から月を削除します。

戻り型

TIMESTAMP

例

次のクエリは、TRUNC 関数内の ADD_MONTHS 関数を使用します。TRUNC 関数は、ADD_MONTHS の結果から日付の時刻を削除します。ADD_MONTHS 関数は CALDATE 列の値ごとに 12 か月を追加します。CALDATE 列の値は日付です。

```
select distinct trunc(add_months(caldate, 12)) as calplus12,
trunc(caldate) as cal
from date
order by 1 asc;
```

```
 calplus12  |    cal
------------+------------
 2009-01-01 | 2008-01-01
 2009-01-02 | 2008-01-02
 2009-01-03 | 2008-01-03
...
(365 rows)
```

次の例では、ADD_MONTHS 関数を使用して、timestamp に 1 か月を加算しています。

```
select add_months('2008-01-01 05:07:30', 1);

add_months
---------------------
2008-02-01 05:07:30
```

次の例は、ADD_MONTHS 関数が異なる日数の月を持つ日付で実行される動作を示しています。この例は、関数が 3 月 31 日への 1 か月の加算と、4 月 30 日への 1 か月の加算をどのように処理するかを示しています。4 月は 30 日間あるので、3 月 31 日に 1 か月を加えると 4 月 30 日になります。5 月は 31 日間あるので、4 月 30 日に 1 か月を加えると 5 月 31 日になります。

```
select add_months('2008-03-31',1);

add_months
---------------------
2008-04-30 00:00:00

select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
```

AT TIME ZONE 関数

AT TIME ZONE は、TIMESTAMP 式または TIMESTAMPTZ 式で使用するタイムゾーンを指定します。

構文

```
AT TIME ZONE 'timezone'
```

引数

timezone

戻り値の TIMEZONE。タイムゾーンは、タイムゾーン名 (**'Africa/Kampala'** または **'Singapore'** など) またはタイムゾーンの略名 (**'UTC'** または **'PDT'** など) として指定できます。

サポートされるタイムゾーン名のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_names();
```

サポートされるタイムゾーン省略形のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_abbrevs();
```

詳細な説明と例については、「タイムゾーンの使用上の注意」を参照してください。

戻り型

TIMESTAMP 式で使用する場合は TIMESTAMPTZ。TIMESTAMPTZ 式で使用する場合は TIMESTAMP。

例

次の例では、タイムゾーンのないタイムゾーン値を変換して MST 時間 (POSIX では UTC+7) として解釈します。この例では、UTC タイムゾーンのデータ型 TIMESTAMPTZ の値を返します。デフォルトのタイムゾーンを UTC 以外のタイムゾーンに設定すると、異なる結果が表示される場合があります。

```
SELECT TIMESTAMP '2001-02-16 20:38:40' AT TIME ZONE 'MST';

timezone
------------------------
2001-02-17 03:38:40+00
```

次の例では、指定されたタイムゾーンが EST (POSIX では UTC+5) であるタイムゾーン値を持つ入力タイムスタンプを取得し、それを MST (POSIX では UTC+7) に変換します。この例では、データ型 TIMESTAMP の値を返します。

```
SELECT TIMESTAMPTZ '2001-02-16 20:38:40-05' AT TIME ZONE 'MST';

timezone
------------------------
2001-02-16 18:38:40
```

CONVERT_TIMEZONE 関数

CONVERT_TIMEZONE は、タイムスタンプのタイムゾーンを別のタイムゾーンに変換します。この関数は夏時間に合わせて自動的に調整されます。

構文

```
CONVERT_TIMEZONE( ['source_timezone',] 'target_timezone', 'timestamp')
```

引数

source_timezone

(オプション) 現在のタイムスタンプのタイムゾーン。デフォルトは UTC です。詳細については、「タイムゾーンの使用上の注意」を参照してください。

target_timezone

新しいタイムスタンプのタイムゾーン。詳細については、「タイムゾーンの使用上の注意」を参照してください。

timestamp

タイムスタンプの列、あるいは暗黙的にタイムスタンプに変換される式。

戻り型

TIMESTAMP

タイムゾーンの使用上の注意

source_timezone または target_timezone のいずれかをタイムゾーン名 (「Africa/Kampala」または「Singapore」など) またはタイムゾーンの略名 (「UTC」または「PDT」など) として指定できます。タイムゾーン名を名前に変換したり、略語を略語に変換したりする必要はありません。例えば、

ソースタイムゾーン名「Singapore」からタイムスタンプを選択して、タイムゾーンの略語「PDT」のタイムスタンプに変換できます。

> **Note**
>
> タイムゾーン名またはタイムゾーンの略名を使用した結果は、地域の季節ごとの時間 (夏時間など) に応じて異なる場合があります。

タイムゾーン名の使用

タイムゾーン名の完全なリストの最新版を表示するには、次のコマンドを実行します。

```
select pg_timezone_names();
```

各行には、タイムゾーン名、略名、UTC オフセット、およびタイムゾーンがサマータイムを採用しているかどうかのインジケーター (t または f) がカンマで区切られた文字列が含まれます。例えば、次のスニペットは、結果として 2 行が表示されます。最初の行はタイムゾーン `Europe/Paris` で略名は `CET`、UTC からのオフセットは `01:00:00` です。`f` によって夏時間が適用されないことが示されています。2 行目はタイムゾーン `Israel` で略名は `IST`、UTC からのオフセットは `02:00:00` です。`f` によって夏時間が適用されないことが示されています。

```
pg_timezone_names
-----------------
(Europe/Paris,CET,01:00:00,f)
(Israel,IST,02:00:00,f)
```

SQL ステートメントを実行してリスト全体を取得し、タイムゾーン名を検索します。約 600 行が返されます。返されるタイムゾーン名は大文字のイニシャルや頭字語 (例: GB、PRC、ROK) になっていますが、CONVERT_TIMEZONE 関数ではこれらはタイムゾーンの省略形ではなくタイムゾーン名として扱われます。

タイムゾーン名を使用してタイムゾーンを指定する場合、CONVERT_TIMEZONE は自動的に夏時間 (DST)、または 'timestamp' によって指定される日付と時刻で有効なその他の現地の季節の慣習 (サマータイム、標準時、冬時間) を調整します。例えば、'Europe/London' は冬季は UTC を表し、夏季は 1 時間を追加します。

タイムゾーンの略名の使用

タイムゾーン略名の完全なリストの最新版を表示するには、次のコマンドを実行します。

```
select pg_timezone_abbrevs();
```

結果には、タイムゾーン略名、UTC オフセット、およびタイムゾーンがサマータイムを採用しているかどうかのインジケーター (t または f) がカンマで区切られた文字列が含まれます。たとえば、次のスニペットでは、結果として 2 行が表示されます。最初の行には、UTC からのオフセットは -07:00:00 の太平洋夏時間の略語 PDT が含まれており、t は夏時間を採用していることがわかります。最初の行には、UTC からのオフセットは -08:00:00 の太平洋標準時間の略語 PST が含まれており、f は夏時間を採用していないことがわかります。

```
pg_timezone_abbrevs
--------------------
(PDT,-07:00:00,t)
(PST,-08:00:00,f)
```

SQL ステートメントを実行してリスト全体を取得し、そのオフセットとサマータイムインジケーターに基づいて略語を検索します。約 200 行が返されます。

タイムゾーンの略名は、UTC からの固定オフセットを表します。タイムゾーンの略名を使用してタイムゾーンを指定する場合、CONVERT_TIMEZONE は UTC からの固定オフセットを使用し、現地の季節の慣習を調整しません。

POSIX スタイル形式の使用

POSIX スタイルのタイムゾーン仕様は STDoffset または STDoffsetDST の形式であり、これらの STD はタイムゾーンの省略形、offset は UTC から西方向への時間単位の数値オフセット、および DST はオプションの夏時間ゾーンの省略形です。夏時間は、特定のオフセットより 1 時間早いことを前提としています。

POSIX スタイルのタイムゾーン形式では、グリニッジから西方向に正のオフセットが使用されます。これは、グリニッジから東方向に正のオフセットを使用する ISO-8601 規則とは対照的です。

POSIX スタイルのタイムゾーンの例を以下に示します。

- PST8
- PST8PDT
- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift は、POSIX スタイルのタイムゾーン仕様を検証しないため、タイムゾーンを無効な値に設定する可能性があります。例えば、次のコマンドはタイムゾーンを無効な値に設定しますが、エラーを返しません。
>
> ```
> set timezone to ‘xxx36’;
> ```

例

多くの例では、TICKIT サンプルデータセットを使用しています。詳細については、「[サンプルデータベース](#)」を参照してください。

次の例は、タイムスタンプ値をデフォルトの UTC タイムゾーンから PST に変換します。

```
select convert_timezone('PST', '2008-08-21 07:23:54');

 convert_timezone
------------------------
2008-08-20 23:23:54
```

次の例は、LISTTIME 列のタイムスタンプ値をデフォルトの UTC タイムゾーンから PST に変換します。タイムスタンプが夏時間の期間内であっても、変換後のタイムゾーンが略名 (PST) で指定されているため、標準時間に変換されます。

```
select listtime, convert_timezone('PST', listtime) from listing
where listid = 16;

     listtime       |   convert_timezone
--------------------+-------------------
2008-08-24 09:36:12     2008-08-24 01:36:12
```

次の例は、タイムスタンプの LISTTIME 列をデフォルトの UTC タイムゾーンから US/Pacific タイムゾーンに変換します。変換後のタイムゾーンはタイムゾーン名で指定されており、タイムスタンプは夏時間の期間内であるため、この関数は夏時間を返します。

```
select listtime, convert_timezone('US/Pacific', listtime) from listing
where listid = 16;
```

```
     listtime       |   convert_timezone
--------------------+---------------------
2008-08-24 09:36:12 | 2008-08-24 02:36:12
```

次の例は、タイムスタンプの文字列を EST から PST に変換します。

```
select convert_timezone('EST', 'PST', '20080305 12:25:29');

 convert_timezone
-------------------
2008-03-05 09:25:29
```

次の例は、変換後のタイムゾーンがタイムゾーン名 (America/New_York) で指定されており、タイムスタンプが標準時間の期間内にあるため、タイムスタンプを米国東部標準時に変換します。

```
select convert_timezone('America/New_York', '2013-02-01 08:00:00');

 convert_timezone
---------------------
2013-02-01 03:00:00
(1 row)
```

次の例は、変換後のタイムゾーンがタイムゾーン名 (America/New_York) で指定されており、タイムスタンプが夏時間の期間内にあるため、タイムスタンプを米国東部夏時間に変換します。

```
select convert_timezone('America/New_York', '2013-06-01 08:00:00');

 convert_timezone
---------------------
2013-06-01 04:00:00
(1 row)
```

次の例は、オフセットの使用を示しています。

```
SELECT CONVERT_TIMEZONE('GMT','NEWZONE +2','2014-05-17 12:00:00') as newzone_plus_2,
CONVERT_TIMEZONE('GMT','NEWZONE-2:15','2014-05-17 12:00:00') as newzone_minus_2_15,
CONVERT_TIMEZONE('GMT','America/Los_Angeles+2','2014-05-17 12:00:00') as la_plus_2,
CONVERT_TIMEZONE('GMT','GMT+2','2014-05-17 12:00:00') as gmt_plus_2;

   newzone_plus_2    | newzone_minus_2_15  |      la_plus_2      |     gmt_plus_2
```

```
---------------------+---------------------+---------------------+---------------------
2014-05-17 10:00:00 | 2014-05-17 14:15:00 | 2014-05-17 10:00:00 | 2014-05-17 10:00:00
(1 row)
```

CURRENT_DATE 関数

CURRENT_DATE は、現在のセッションのタイムゾーン (デフォルトは UTC) の日付をデフォルト形式 YYYY-MM-DD で返します。

> **Note**
>
> CURRENT_DATE は、現在のステートメントの開始日ではなく、現在のトランザクションの開始日を返します。複数のステートメントを含むトランザクションを 2008 年 10 月 1 日 23:59 に開始し、CURRENT_DATE を含むステートメントが 2008 年 10 月 2 日 00:00 に実行されるシナリオを考えてみましょう。CURRENT_DATE は `10/02/08` ではなく、`10/01/08` を返します。

構文

```
CURRENT_DATE
```

戻り型

DATE

例

次の例では、現在の日付を返します (関数が実行される AWS リージョン)。

```
select current_date;

   date
------------
2008-10-01
```

次の例では、テーブルを作成し、列 `todays_date` のデフォルトが CURRENT_DATE である行を挿入し、テーブル内のすべての行を選択します。

```
CREATE TABLE insert_dates(
    label varchar(128) NOT NULL,
    todays_date DATE DEFAULT CURRENT_DATE);

INSERT INTO insert_dates(label)
VALUES('Date row inserted');

SELECT * FROM insert_dates;


 label             | todays_date
-------------------+-------------
Date row inserted | 2023-05-10
```

DATE_CMP 関数

DATE_CMP は 2 つの日付を比較します。この関数は、日付が同一の場合は 0、date1 が大きい場合は 1、date2 が大きい場合は -1 を返します。

構文

```
DATE_CMP(date1, date2)
```

引数

date1

データ型 DATE の値または DATE 型と評価される式の列。

date2

データ型 DATE の値または DATE 型と評価される式の列。

戻り型

INTEGER

例

次のクエリは CALDATE 列の DATE 値を 2008 年 1 月 4 日の日付と比較し、CALDATE の値が 2008 年 1 月 4 日より前 (-1)、同じ (0)、より後 (1) であるかを返します。

```
select caldate, '2008-01-04',
date_cmp(caldate,'2008-01-04')
from date
order by dateid
limit 10;

 caldate   |  ?column?  | date_cmp
-----------+------------+----------
2008-01-01 | 2008-01-04 |       -1
2008-01-02 | 2008-01-04 |       -1
2008-01-03 | 2008-01-04 |       -1
2008-01-04 | 2008-01-04 |        0
2008-01-05 | 2008-01-04 |        1
2008-01-06 | 2008-01-04 |        1
2008-01-07 | 2008-01-04 |        1
2008-01-08 | 2008-01-04 |        1
2008-01-09 | 2008-01-04 |        1
2008-01-10 | 2008-01-04 |        1
(10 rows)
```

DATE_CMP_TIMESTAMP 関数

DATE_CMP_TIMESTAMP は、日付とタイムスタンプを比較し、値が同一の場合は `0`、date が時間的により大きい場合は `1`、timestamp がより大きい場合は `-1` を返します。

構文

```
DATE_CMP_TIMESTAMP(date, timestamp)
```

引数

date

データ型 `DATE` の値または `DATE` 型と評価される式の列。

timestamp

データ型 `TIMESTAMP` の値または `TIMESTAMP` 型と評価される式の列。

戻り型

INTEGER

例

次の例は日付 2008-06-18 と LISTTIME を比較します。LISTTIME 列の値はタイムスタンプです。この日付より前に作成されたリストは 1 を返し、この日付より後に作成されたリストは -1 を返します。

```
select listid, '2008-06-18', listtime,
date_cmp_timestamp('2008-06-18', listtime)
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      listtime       | date_cmp_timestamp
--------+------------+---------------------+--------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29 |                  1
      2 | 2008-06-18 | 2008-03-05 12:25:29 |                  1
      3 | 2008-06-18 | 2008-11-01 07:35:33 |                 -1
      4 | 2008-06-18 | 2008-05-24 01:18:37 |                  1
      5 | 2008-06-18 | 2008-05-17 02:29:11 |                  1
      6 | 2008-06-18 | 2008-08-15 02:08:13 |                 -1
      7 | 2008-06-18 | 2008-11-15 09:38:15 |                 -1
      8 | 2008-06-18 | 2008-11-09 05:07:30 |                 -1
      9 | 2008-06-18 | 2008-09-09 08:03:36 |                 -1
     10 | 2008-06-18 | 2008-06-17 09:44:54 |                  1
(10 rows)
```

DATE_CMP_TIMESTAMPTZ 関数

DATE_CMP_TIMESTAMPTZ は、日付とタイムゾーン付きのタイムスタンプを比較し、値が同一の場合は 0、date が時間的により大きい場合は 1、timestamptz がより大きい場合は -1 を返します。

構文

```
DATE_CMP_TIMESTAMPTZ(date, timestamptz)
```

引数

date

データ型 DATE の列または DATE 型に暗黙的に評価される式。

timestamptz

データ型 TIMESTAMPTZ の列または TIMESTAMPTZ 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例は日付 2008-06-18 と LISTTIME を比較します。この日付より前に作成されたリストは 1 を返し、この日付より後に作成されたリストは -1 を返します。

```
select listid, '2008-06-18', CAST(listtime AS timestamptz),
date_cmp_timestamptz('2008-06-18', CAST(listtime AS timestamptz))
from listing
order by 1, 2, 3, 4
limit 10;

 listid |  ?column?  |      timestamptz       | date_cmp_timestamptz
--------+------------+------------------------+----------------------
      1 | 2008-06-18 | 2008-01-24 06:43:29+00 |                    1
      2 | 2008-06-18 | 2008-03-05 12:25:29+00 |                    1
      3 | 2008-06-18 | 2008-11-01 07:35:33+00 |                   -1
      4 | 2008-06-18 | 2008-05-24 01:18:37+00 |                    1
      5 | 2008-06-18 | 2008-05-17 02:29:11+00 |                    1
      6 | 2008-06-18 | 2008-08-15 02:08:13+00 |                   -1
      7 | 2008-06-18 | 2008-11-15 09:38:15+00 |                   -1
      8 | 2008-06-18 | 2008-11-09 05:07:30+00 |                   -1
      9 | 2008-06-18 | 2008-09-09 08:03:36+00 |                   -1
     10 | 2008-06-18 | 2008-06-17 09:44:54+00 |                    1
(10 rows)
```

DATEADD 関数

指定された間隔で DATE、TIME、TIMETZ、または TIMESTAMP の値を増分します。

構文

```
DATEADD( datepart, interval, {date|time|timetz|timestamp} )
```

引数

datepart

関数が実行される日付部分 (例: 年、月、日、または時間)。詳細については、「[日付関数またはタイムスタンプ関数の日付部分]」を参照してください。

interval

ターゲット式に追加する間隔 (日数など) を指定する整数。負の整数は間隔を減算します。

date|time|timetz|timestamp

DATE、TIME、TIMETZ、または TIMESTAMP の列、あるいは暗黙的に DATE、TIME、TIMETZ、または TIMESTAMP に変換される式。DATE、TIME、TIMETZ、または TIMESTAMP の式には、指定した日付部分が含まれている必要があります。

戻り型

入力データ型に応じて TIMESTAMP、TIME、または TIMETZ を指定します。

DATE 列の例

次の例では、DATE テーブルに存在する 11 月の各日付に 30 日追加します。

```
select dateadd(day,30,caldate) as novplus30
from date
where month='NOV'
order by dateid;

novplus30
---------------------
2008-12-01 00:00:00
2008-12-02 00:00:00
2008-12-03 00:00:00
...
(30 rows)
```

次の例は、リテラル日付値に 18 か月を追加します。

```
select dateadd(month,18,'2008-02-28');
```

```
date_add
---------------------
2009-08-28 00:00:00
(1 row)
```

DATEDIFF 関数のデフォルトの列名は DATE_ADD です。日付の値に使用するデフォルトのタイムスタンプは `00:00:00` です。

次の例では、タイムスタンプを指定しない日付の値に 30 分を追加します。

```
select dateadd(m,30,'2008-02-28');

date_add
---------------------
2008-02-28 00:30:00
(1 row)
```

完全名または略名で日付部分に名前を付けることができます。この場合、m は月ではなく分を表します。

TIME 列の例

次のテーブルの TIME_TEST の例には、3 つの値が挿入された列 TIME_VAL (タイプ TIME) があります。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

次の例では、TIME_TEST テーブルの各 TIME_VAL に 5 分を追加します。

```
select dateadd(minute,5,time_val) as minplus5 from time_test;

minplus5
---------------
20:05:00
00:05:00.5550
```

```
01:03:00
```

次の例では、リテラル時刻値に 8 時間を追加します。

```
select dateadd(hour, 8, time '13:24:55');

date_add
---------------
21:24:55
```

次の例では、時刻が 24:00:00 を超えるか 00:00:00 を下回る場合を示しています。

```
select dateadd(hour, 12, time '13:24:55');

date_add
---------------
01:24:55
```

TIMETZ 列の例

これらの例の出力値は、デフォルトのタイムゾーンである UTC です。

次のテーブルの TIMETZ_TEST の例には、3 つの値が挿入された列 TIMETZ_VAL (タイプ TIMETZ) があります。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

次の例では、TIMETZ_TEST テーブルの各 TIMETZ_VAL に 5 分を追加します。

```
select dateadd(minute,5,timetz_val) as minplus5_tz from timetz_test;

minplus5_tz
---------------
04:05:00+00
00:05:00.5550+00
```

```
06:03:00+00
```

次の例では、リテラルの timetz 値に 2 時間を追加します。

```
select dateadd(hour, 2, timetz '13:24:55 PST');

date_add
---------------
23:24:55+00
```

TIMESTAMP 列の例

これらの例の出力値は、デフォルトのタイムゾーンである UTC です。

次のテーブルの TIMESTAMP_TEST の例には、3 つの値が挿入された列 TIMESTAMP_VAL (タイプ TIMESTAMP) があります。

```
SELECT timestamp_val FROM timestamp_test;

timestamp_val
------------------
1988-05-15 10:23:31
2021-03-18 17:20:41
2023-06-02 18:11:12
```

次の例では、2000 年より前の TIMESTAMP_TEST の TIMESTAMP_VAL 値にのみ 20 年を加算しています。

```
SELECT dateadd(year,20,timestamp_val)
FROM timestamp_test
WHERE timestamp_val < to_timestamp('2000-01-01 00:00:00', 'YYYY-MM-DD HH:MI:SS');

date_add
---------------
2008-05-15 10:23:31
```

次の例では、秒インジケータなしで書き込まれたリテラルタイムスタンプ値に 5 秒を加算します。

```
SELECT dateadd(second, 5, timestamp '2001-06-06');
```

```
date_add
---------------
2001-06-06 00:00:05
```

使用に関する注意事項

DATEADD(month, ...) および ADD_MONTHS 関数は、異なる月末になる日付を処理します。

- ADD_MONTHS: 追加している日付が月の最終日である場合、結果は月の期間にかかわらず、常に結果の月の最終日になります。例えば、4 月 30 日 + 1 か月は 5 月 31 日になります。

```
select add_months('2008-04-30',1);

add_months
---------------------
2008-05-31 00:00:00
(1 row)
```

- DATEADD: 追加している日付が結果の月より短い場合、結果は月の最終日ではなく、結果の月の対応する日付になります。例えば、4 月 30 日 + 1 か月は 5 月 30 日になります。

```
select dateadd(month,1,'2008-04-30');

date_add
---------------------
2008-05-30 00:00:00
(1 row)
```

DATEADD 関数では dateadd(month, 12,…) または dateadd(year, 1, …) を使用するとき、うるう年の日付 02-29 は扱いが異なります。

```
select dateadd(month,12,'2016-02-29');

date_add
---------------------
2017-02-28 00:00:00

select dateadd(year, 1, '2016-02-29');

date_add
---------------------
```

```
2017-03-01 00:00:00
```

DATEDIFF 関数

DATEDIFF は 2 つの日付または時刻式の日付部分の差を返します。

構文

```
DATEDIFF( datepart, {date|time|timetz|timestamp}, {date|time|timetz|timestamp} )
```

引数

datepart

関数が実行される日付または時刻値 (年、月、または日、時、分、秒、ミリ秒、またはマイクロ秒) の特定部分。詳細については、「[日付関数またはタイムスタンプ関数の日付部分]」を参照してください。

特に、DATEDIFF は 2 つの式の間で越える日付部分の境界の数を決定します。例えば、`12-31-2008` と `01-01-2009` の 2 つの日付間で年の差を計算しているとします。この場合、これらの日付は 1 日しか離れていないにもかかわらず、関数は 1 年を返します。2 つのタイムスタンプ (`01-01-2009 8:30:00` と `01-01-2009 10:00:00`) の間で時間の差が分かっている場合、結果は 2 時間になります。2 つのタイムスタンプ (`8:30:00` と `10:00:00`) の間で時間の差が分かっている場合、結果は 2 時間になります。

date|time|timetz|timestamp

DATE、TIME、TIMETZ、または TIMESTAMP の列、あるいは暗黙的に DATE、TIME、TIMETZ、または TIMESTAMP に変換される式。両方の式には、指定した日付部分または時刻部分を含める必要があります。2 番目の日付または時刻が 1 番目の日付または時刻よりも後である場合、結果は正です。2 番目の日付または時刻が 1 番目の日付または時刻よりも前である場合、結果は負です。

戻り型

BIGINT

DATE 列の例

次の例では、2 つの日付リテラル値の間の差 (週単位) を取得します。

```
select datediff(week,'2009-01-01','2009-12-31') as numweeks;

numweeks
----------
52
(1 row)
```

次の例は、2 つの日付リテラル値の差 (時間単位) を検出します。日付の時刻値を指定しなかった場合、デフォルトで 00:00:00 に設定されます。

```
select datediff(hour, '2023-01-01', '2023-01-03 05:04:03');

date_diff
----------
53
(1 row)
```

次の例は、2 つの日付リテラル TIMESTAMETZ 値の差 (日単位) を検出します。

```
Select datediff(days, 'Jun 1,2008  09:59:59 EST', 'Jul 4,2008  09:59:59 EST')

date_diff
----------
33
```

次の例は、テーブルの同じ行の 2 つの日付の差 (日単位) を検出します。

```
select * from date_table;

start_date |   end_date
-----------+-----------
2009-01-01 | 2009-03-23
2023-01-04 | 2024-05-04
(2 rows)

select datediff(day, start_date, end_date) as duration from date_table;

duration
---------
      81
     486
```

```
(2 rows)
```

次の例では、過去のリテラル値と今日の日付の間の差 (四半期単位) を取得します。この例では、現在の日付を 2008 年 6 月 5 日とします。完全名または略名で日付部分に名前を付けることができます。DATEDIFF 関数のデフォルトの列名は DATE_DIFF です。

```
select datediff(qtr, '1998-07-01', current_date);

date_diff
-----------
40
(1 row)
```

次の例では、SALES テーブルと LISTING テーブルを結合し、リスト 1000 から 1005 に対してチケットをリストしてから何日後に販売されたかを計算します。これらのリストの販売を最長の待機期間は 15 日であり、最小は 1 日より短いです (0 日)。

```
select priceperticket,
datediff(day, listtime, saletime) as wait
from sales, listing where sales.listid = listing.listid
and sales.listid between 1000 and 1005
order by wait desc, priceperticket desc;

priceperticket | wait
---------------+------
 96.00         |   15
 123.00        |   11
 131.00        |    9
 123.00        |    6
 129.00        |    4
 96.00         |    4
 96.00         |    0
(7 rows)
```

この例は、販売者が任意およびすべてのチケット販売を待機する平均時間を計算します。

```
select avg(datediff(hours, listtime, saletime)) as avgwait
from sales, listing
where sales.listid = listing.listid;

avgwait
```

```
---------
465
(1 row)
```

TIME 列の例

次のテーブルの TIME_TEST の例には、3 つの値が挿入された列 TIME_VAL (タイプ TIME) があります。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

次の例では、TIME_VAL 列と時刻リテラル間の時間数の差を検出します。

```
select datediff(hour, time_val, time '15:24:45') from time_test;

 date_diff
-----------
        -5
        15
        15
```

次の例では、2 つのリテラル時間値の分数の差を検出します。

```
select datediff(minute, time '20:00:00', time '21:00:00') as nummins;

nummins
----------
60
```

TIMETZ 列の例

次のテーブルの TIMETZ_TEST の例には、3 つの値が挿入された列 TIMETZ_VAL (タイプ TIMETZ) があります。

```
select timetz_val from timetz_test;
```

```
timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

次の例では、TIMETZ リテラルと timetz_val の間の時間数の差を検出します。

```
select datediff(hours, timetz '20:00:00 PST', timetz_val) as numhours from timetz_test;

numhours
----------
0
-4
1
```

次の例では、2 つのリテラル TIMETZ 値間の時間数の差を検出します。

```
select datediff(hours, timetz '20:00:00 PST', timetz '00:58:00 EST') as numhours;

numhours
----------
1
```

DATE_PART 関数

DATE_PART は式から日付部分の値を抽出します。DATE_PART は PGDATE_PART 関数のシノニムです。

構文

```
DATE_PART(datepart, {date|timestamp})
```

引数

datepart

関数が実行される日付の値の特定部分 (例: 年、月、または日) の識別子リテラルまたは文字列。詳細については、「日付関数またはタイムスタンプ関数の日付部分」を参照してください。

{date|timestamp}

日付列、タイムスタンプ列、または暗黙的に日付またはタイムスタンプに変換される式。日付またはタイムスタンプの列または式には、datepart で指定された日付部分が含まれている必要があります。

戻り型

DOUBLE

例

DATE_PART 関数のデフォルトの列名は pgdate_part です。

次の例の一部で使用されるデータの詳細については、「サンプルデータベース」を参照してください。

次の例では、タイムスタンプリテラルから分を見つけます。

```
SELECT DATE_PART(minute, timestamp '20230104 04:05:06.789');

pgdate_part
-----------
          5
```

次の例では、タイムスタンプリテラルから週番号を見つけます。週番号の計算は、ISO 8601 標準に従います。詳細については、Wikipedia の「ISO 8601」を参照してください。

```
SELECT DATE_PART(week, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         18
```

次の例では、タイムスタンプリテラルから日付を見つけます。

```
SELECT DATE_PART(day, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          2
```

次の例では、タイムスタンプリテラルから曜日を見つけます。曜日番号の計算は、日曜日から始まる 0～6 の整数です。

```
SELECT DATE_PART(dayofweek, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          1
```

次の例では、タイムスタンプリテラルから世紀を見つけます。世紀の計算は、ISO 8601 標準に従います。詳細については、Wikipedia の「ISO 8601」を参照してください。

```
SELECT DATE_PART(century, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
         21
```

次の例は、タイムスタンプリテラルからミレニアムを検出します。ミレニアムの計算は、ISO 8601 標準に従います。詳細については、Wikipedia の「ISO 8601」を参照してください。

```
SELECT DATE_PART(millennium, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
          3
```

次の例は、タイムスタンプリテラルからマイクロ秒を検出します。マイクロ秒の計算は、ISO 8601 標準に従います。詳細については、Wikipedia の「ISO 8601」を参照してください。

```
SELECT DATE_PART(microsecond, timestamp '20220502 04:05:06.789');

pgdate_part
-----------
     789000
```

次の例では、日付リテラルから月を見つけます。

```
SELECT DATE_PART(month, date '20220502');
```

```
pgdate_part
-----------
         5
```

次の例は、DATE_PART 関数をテーブルの列に適用します。

```
SELECT date_part(w, listtime) AS weeks, listtime
FROM listing
WHERE listid=10


weeks |      listtime
------+---------------------
 25   | 2008-06-17 09:44:54
(1 row)
```

完全形あるいは省略形の日付部分に名前を付けることができます。この場合、w は週を指します。

曜日の日付部分は、0～6 の整数を返します (0 は日曜日)。dow (曜日) とともに DATE_PART を使用し、土曜のイベントを表示します。

```
SELECT date_part(dow, starttime) AS dow, starttime
FROM event
WHERE date_part(dow, starttime)=6
ORDER BY 2,1;

 dow |      starttime
-----+---------------------
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
   6 | 2008-01-05 14:00:00
...
(1147 rows)
```

DATE_PART_YEAR 関数

DATE_PART_YEAR 関数は日付から年を抽出します。

構文

```
DATE_PART_YEAR(date)
```

引数

date

データ型 DATE の列または DATE 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、日付リテラルから年を見つけます。

```
SELECT DATE_PART_YEAR(date '20220502 04:05:06.789');

date_part_year
---------------
2022
```

次の例は CALDATE 列から年を抽出します。CALDATE 列の値は日付です。この例で使用されているデータの詳細については、「サンプルデータベース」を参照してください。

```
select caldate, date_part_year(caldate)
from date
order by
dateid limit 10;

 caldate    | date_part_year
-----------+----------------
2008-01-01 |           2008
2008-01-02 |           2008
2008-01-03 |           2008
2008-01-04 |           2008
2008-01-05 |           2008
2008-01-06 |           2008
2008-01-07 |           2008
2008-01-08 |           2008
2008-01-09 |           2008
2008-01-10 |           2008
(10 rows)
```

DATE_TRUNC 関数

DATE_TRUNC 関数は、指定した日付部分 (時、日、月など) に基づいてタイムスタンプの式またはリテラルを切り捨てます。

構文

```
DATE_TRUNC('datepart', timestamp)
```

引数

datepart

タイムスタンプの値を切り捨てる日付部分。入力タイムスタンプは、入力 datepart の精度で切り捨てられます。例えば、`month` は、その月の初日で切り捨てられます。有効な形式は次のとおりです。

- microsecond、microseconds
- millisecond、milliseconds
- second、seconds
- minute、minutes
- hour、hours
- day、days
- week、weeks
- month、months
- quarter、quarters
- year、years
- decade、decades
- century、centuries
- millennium、millennia

形式の略語の詳細については、「日付関数またはタイムスタンプ関数の日付部分」を参照してください。

timestamp

タイムスタンプの列、あるいは暗黙的にタイムスタンプに変換される式。

戻り型

TIMESTAMP

例

入力タイムスタンプを秒単位で切り捨てます。

```
SELECT DATE_TRUNC('second', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:06
```

入力タイムスタンプを分単位で切り捨てます。

```
SELECT DATE_TRUNC('minute', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:05:00
```

入力タイムスタンプを時間単位で切り捨てます。

```
SELECT DATE_TRUNC('hour', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 04:00:00
```

入力タイムスタンプを日単位で切り捨てます。

```
SELECT DATE_TRUNC('day', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-30 00:00:00
```

入力タイムスタンプを月の初日で切り捨てます。

```
SELECT DATE_TRUNC('month', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

入力タイムスタンプを四半期の初日で切り捨てます。

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-04-01 00:00:00
```

入力タイムスタンプを 1 年の初日で切り捨てます。

```
SELECT DATE_TRUNC('year', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2020-01-01 00:00:00
```

入力タイムスタンプを 1 世紀の初日で切り捨てます。

```
SELECT DATE_TRUNC('millennium', TIMESTAMP '20200430 04:05:06.789');
date_trunc
2001-01-01 00:00:00
```

入力タイムスタンプを月曜日に切り捨てます。

```
select date_trunc('week', TIMESTAMP '20220430 04:05:06.789');
date_trunc
2022-04-25 00:00:00
```

次の例では、DATE_TRUNC 関数が 'week' の日付部分を使用して、各週の月曜日の日付を返します。

```
select date_trunc('week', saletime), sum(pricepaid) from sales where
saletime like '2008-09%' group by date_trunc('week', saletime) order by 1;

date_trunc  |    sum
------------+-------------
2008-09-01  | 2474899
2008-09-08  | 2412354
2008-09-15  | 2364707
2008-09-22  | 2359351
2008-09-29  |  705249
```

EXTRACT 関数

EXTRACT 関数は、TIMESTAMP、TIMESTAMPTZ、TIME、TIMETZ、 INTERVAL YEAR TO MONTH、または INTERVAL DAY TO SECOND の値から日付または時刻のパートを返します。例としては、タイムスタンプの日、月、年、時、分、秒、ミリ秒、マイクロ秒などがあります。

構文

```
EXTRACT(datepart FROM source)
```

引数

datepart

日、月、年、時、分、秒、ミリ秒、マイクロ秒など、抽出する日付または時刻のサブフィールド。有効な値については、「日付関数またはタイムスタンプ関数の日付部分」を参照してください。

source

評価結果が TIMESTAMP、TIMESTAMPTZ、TIME、TIMETZ、INTERVAL YEAR TO MONTH、または INTERVAL DAY TO SECOND のデータ型になる列または式。

戻り型

source 値が TIMESTAMP、TIME、TIMETZ、INTERVAL YEAR TO MONTH、または INTERVAL DAY TO SECOND のデータ型として評価される場合は INTEGER。

source 値がデータ型 TIMESTAMPTZ として評価される場合は、DOUBLE PRECISION。

TIMESTAMP の例

次の例では、支払価格が 10,000 USD 以上であった販売の週の数を判定します。この例では、TICKIT データを使用します。詳細については、「サンプルデータベース」を参照してください。

```
select salesid, extract(week from saletime) as weeknum
from sales
where pricepaid > 9999
order by 2;

salesid | weeknum
--------+---------
 159073 |       6
 160318 |       8
 161723 |      26
```

次の例では、リテラルタイムスタンプの値から分の値を返します。

```
select extract(minute from timestamp '2009-09-09 12:08:43');

date_part
```

```
-----------
8
```

次の例は、リテラルタイムスタンプ値からミリ秒の値を返します。

```
select extract(ms from timestamp '2009-09-09 12:08:43.101');

date_part
-----------
101
```

TIMESTAMPTZ の例

次の例は、リテラル timestamptz 値から年の値を返します。

```
select extract(year from timestamptz '1.12.1997 07:37:16.00 PST');

date_part
-----------
1997
```

TIME の例

次のテーブルの TIME_TEST の例には、3 つの値が挿入された列 TIME_VAL (タイプ TIME) があります。

```
select time_val from time_test;

time_val
---------------------
20:00:00
00:00:00.5550
00:58:00
```

次の例は、各 time_val から分を抽出します。

```
select extract(minute from time_val) as minutes from time_test;

minutes
-----------
         0
```

```
          0
          58
```

次の例は、各 time_val から時間を抽出します。

```
select extract(hour from time_val) as hours from time_test;

hours
-----------
         20
          0
          0
```

次の例では、リテラル値からミリ秒を抽出します。

```
select extract(ms from time '18:25:33.123456');

 date_part
-----------
     123
```

TIMETZ の例

次のテーブルの TIMETZ_TEST の例には、3 つの値が挿入された列 TIMETZ_VAL (タイプ TIMETZ) があります。

```
select timetz_val from timetz_test;

timetz_val
------------------
04:00:00+00
00:00:00.5550+00
05:58:00+00
```

次の例では、各 timetz_val から時間を抽出します。

```
select extract(hour from timetz_val) as hours from time_test;

hours
-----------
          4
```

```
         0
         5
```

次の例では、リテラル値からミリ秒を抽出します。抽出が処理される前には、リテラルは UTC に変換されません。

```
select extract(ms from timetz '18:25:33.123456 EST');

 date_part
-----------
     123
```

次の例は、リテラル timetz 値から、UTC からのタイムゾーンオフセット時を返します。

```
select extract(timezone_hour from timetz '1.12.1997 07:37:16.00 PDT');

date_part
-----------
-7
```

INTERVAL YEAR TO MONTH と INTERVAL DAY TO SECOND の例

次の例では、36 時間 (1 日と 12 時間) を定義する INTERVAL DAY TO SECOND から日のパートである 1 を抽出します。

```
select EXTRACT('days' from INTERVAL '36 hours' DAY TO SECOND)

 date_part
------------------
 1
```

次の例では、15 か月 (1 年と 3 か月) を定義する YEAR TO MONTH から月のパートである 3 を抽出します。

```
select EXTRACT('month' from INTERVAL '15 months' YEAR TO MONTH)

 date_part
------------------
 3
```

次の例では、30 か月 (2 年と 6 か月) から月のパートである 6 を抽出します。

```
select EXTRACT('month' from INTERVAL '30' MONTH)

 date_part
------------------
 6
```

次の例では、50 時間 (2 日と 2 時間) から時間のパートである 2 を抽出します。

```
select EXTRACT('hours' from INTERVAL '50' HOUR)

 date_part
------------------
 2
```

次の例では、1 時間 11 分 11.123 秒から分のパートである 11 を抽出します。

```
select EXTRACT('minute' from INTERVAL '70 minutes 70.123 seconds' MINUTE TO SECOND)

 date_part
------------------
 11
```

次の例では、1 日 1 時間 1 分 1.11 秒から秒のパートである 1.11 を抽出します。

```
select EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 date_part
------------------
 1.11
```

次の例では、INTERVAL の合計時間数を抽出します。各パートが抽出され、合計に追加されます。

```
select EXTRACT('days' from INTERVAL '50' HOUR) * 24 + EXTRACT('hours' from INTERVAL
 '50' HOUR)

 ?column?
------------------
 50
```

次の例では、INTERVAL の合計秒数を抽出します。各パートが抽出され、合計に追加されます。

```
select EXTRACT('days' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 86400 +
       EXTRACT('hours' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 3600 +
       EXTRACT('minutes' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND) * 60 +
       EXTRACT('seconds' from INTERVAL '1 day 1:1:1.11' DAY TO SECOND)

 ?column?
------------------
 90061.11
```

GETDATE 関数

GETDATE は、現在のセッションのタイムゾーン (デフォルトでは UTC) で現在の日付と時刻を返します。トランザクションブロック内にある場合でも、現在のステートメントの開始日または開始時刻を返します。

構文

```
GETDATE()
```

かっこが必要です。

戻り型

TIMESTAMP

例

次の例は、GETDATE 関数を使用して現在の日付の完全なタイムスタンプを返します。

```
select getdate();

timestamp
---------------------
2008-12-04 16:10:43
```

次の例では、TRUNC 関数内の GETDATE 関数を使用し、時刻のない現在の日付を返します。

```
select trunc(getdate());

trunc
```

```
------------
2008-12-04
```

INTERVAL_CMP 関数

INTERVAL_CMP は 2 つの間隔を比較し、1 番目の間隔が大きい場合は 1、2 番目の間隔が大きい場合は -1、間隔が等しい場合は 0 をそれぞれ返します。詳細については、「修飾子構文を使用しない間隔リテラルの例」を参照してください。

構文

```
INTERVAL_CMP(interval1, interval2)
```

引数

interval1

間隔のリテラル値。

interval2

間隔のリテラル値。

戻り型

INTEGER

例

次の例は `3 days` の値と `1 year` を比較しています。

```
select interval_cmp('3 days','1 year');

interval_cmp
--------------
-1
```

この例では値 `7 days` と `1 week` を比較しています。

```
select interval_cmp('7 days','1 week');

interval_cmp
```

```
--------------
0
```

次の例は 1 year の値と 3 days を比較しています。

```
select interval_cmp('1 year','3 days');

interval_cmp
--------------
1
```

LAST_DAY 関数

LAST_DAY は、date を含む月の最終日の日付を返します。戻り値の型は、date 引数のデータ型にかかわらず常に DATE です。

特定の日付部分の取得の詳細については、「DATE_TRUNC 関数」を参照してください。

構文

```
LAST_DAY( { date | timestamp } )
```

引数

date | timestamp

データ型 DATE または TIMESTAMP の列、あるいは DATE 型または TIMESTAMP 型に暗黙的に評価される式。

戻り型

DATE

例

次の例では、現在の月の最終日の日付が返されます。

```
select last_day(sysdate);

  last_day
------------
 2014-01-31
```

次の例では、月の最後の 7 日間それぞれに販売されたチケット数が返されます。SALETIME 列の値はタイムスタンプです。

```
select datediff(day, saletime, last_day(saletime)) as "Days Remaining", sum(qtysold)
from sales
where datediff(day, saletime, last_day(saletime)) < 7
group by 1
order by 1;

days remaining |  sum
---------------+-------
             0 | 10140
             1 | 11187
             2 | 11515
             3 | 11217
             4 | 11446
             5 | 11708
             6 | 10988
(7 rows)
```

MONTHS_BETWEEN 関数

MONTHS_BETWEEN は、2 つの日付の間の月数を算出します。

1 番目の日付が 2 番目の日付より値の場合、結果は正になります。それ以外の場合、結果は負になります。

どちらかの引数が null の場合、結果は NULL になります。

構文

```
MONTHS_BETWEEN( date1, date2 )
```

引数

date1

データ型 DATE の列または DATE 型に暗黙的に評価される式。

date2

データ型 DATE の列または DATE 型に暗黙的に評価される式。

戻り型

FLOAT8

結果の整数部分は、日付の年の値と月の値の差分に基づいて決まります。結果の小数部分は、日付の曜日およびタイムスタンプの値から計算され、1 か月が 31 日であると想定しています。

date1 と date2 のどちらにも月の同じ日付が含まれているか (1/15/14 と 2/15/14 など)、月の末日が含まれている場合 (8/31/14 と 9/30/14 など)、タイムスタンプ部分 (存在する場合) が一致するかどうかに関係なく、結果は日付の年と月の値に基づく整数となります。

例

次の例では、1969 年 1 月 18 日から 1969 年 3 月 18 日までの月数を返します。

```
select months_between('1969-01-18', '1969-03-18')
as months;

months
----------
-2
```

次の例では、1969 年 1 月 18 日から 1969 年 1 月 18 日までの月数を返します。

```
select months_between('1969-01-18', '1969-01-18')
as months;

months
----------
0
```

次の例では、イベントの最初の開催から最後の開催までの月数を返します。

```
select eventname,
min(starttime) as first_show,
max(starttime) as last_show,
months_between(max(starttime),min(starttime)) as month_diff
from event
group by eventname
order by eventname
limit 5;

eventname         first_show             last_show              month_diff
```

```
-----------------------------------------------------------------------------
.38 Special        2008-01-21 19:30:00.0  2008-12-25 15:00:00.0  11.12
3 Doors Down       2008-01-03 15:00:00.0  2008-12-01 19:30:00.0  10.94
70s Soul Jam       2008-01-16 19:30:00.0  2008-12-07 14:00:00.0  10.7
A Bronx Tale       2008-01-21 19:00:00.0  2008-12-15 15:00:00.0  10.8
A Catered Affair   2008-01-08 19:30:00.0  2008-12-19 19:00:00.0  11.35
```

NEXT_DAY 関数

NEXT_DAY は、指定の日付より後に指定の曜日となる最初の日付を返します。

day 値が指定の日付と同じ曜日の場合、その曜日の次の出現が返されます。

構文

```
NEXT_DAY( { date | timestamp }, day )
```

引数

date | timestamp

データ型 DATE または TIMESTAMP の列、あるいは DATE 型または TIMESTAMP 型に暗黙的に評価される式。

日

任意の曜日の名前を含む文字列。大文字化は関係ありません。

有効な値は次のとおりです。

日	値
日曜日	Su, Sun, Sunday
月曜日	M, Mo, Mon, Monday
火曜日	Tu、Tue、Tues、Tuesday
水曜日	W、We、Wed、Wednesday
木曜日	Th, Thu, Thurs, Thursday
金曜日	F, Fr, Fri, Friday

日	値
土曜日	Sa, Sat, Saturday

戻り型

DATE

例

次の例では、2014 年 8 月 20 日後の最初の火曜日を返します。

```
select next_day('2014-08-20','Tuesday');

next_day
-----------
2014-08-26
```

次の例では、2008 年 1 月 1 日以降の第 1 火曜日の日付を 5 時 54 分 44 秒に返します。

```
select listtime, next_day(listtime, 'Tue') from listing limit 1;

listtime            | next_day
--------------------+-----------
2008-01-01 05:54:44 | 2008-01-08
```

次の例は、第 3 四半期のターゲットマーケティング日を取得します。

```
select username, (firstname ||' '|| lastname) as name,
eventname, caldate, next_day (caldate, 'Monday') as marketing_target
from sales, date, users, event
where sales.buyerid = users.userid
and sales.eventid = event.eventid
and event.dateid = date.dateid
and date.qtr = 3
order by marketing_target, eventname, name;

username  |     name          |     eventname         |    caldate   |
 marketing_target
----------+-------------------+----------------------+--------------
+-------------------
```

```
MBO26QSG   |   Callum Atkinson | .38 Special            |  2008-07-06   | 2008-07-07
WCR50YIU   |   Erasmus Alvarez | A Doll's House         |  2008-07-03   | 2008-07-07
CKT70OIE   |   Hadassah Adkins | Ana Gabriel            |  2008-07-06   | 2008-07-07
VVG07OUO   |   Nathan Abbott   | Armando Manzanero      |  2008-07-04   | 2008-07-07
GEW77SII   |   Scarlet Avila   | August: Osage County |  2008-07-06   | 2008-07-07
ECR71CVS   |   Caryn Adkins    | Ben Folds              |  2008-07-03   | 2008-07-07
KUW82CYU   |   Kaden Aguilar   | Bette Midler           |  2008-07-01   | 2008-07-07
WZE78DJZ   |   Kay Avila       | Bette Midler           |  2008-07-01   | 2008-07-07
HXY04NVE   |   Dante Austin    | Britney Spears         |  2008-07-02   | 2008-07-07
URY81YWF   |   Wilma Anthony   | Britney Spears         |  2008-07-02   | 2008-07-07
```

SYSDATE 関数

SYSDATE は、現在のセッションのタイムゾーン (デフォルトでは UTC) で現在の日付と時刻を返します。

> **Note**
>
> SYSDATE は現在のステートメントではなく、現在のトランザクションの開始日と時刻を返します。

構文

```
SYSDATE
```

この関数に引数は必要ありません。

戻り型

TIMESTAMP

例

次の例は、SYSDATE 関数を使用して現在の日付の完全なタイムスタンプを返します。

```
select sysdate;

timestamp
----------------------------
2008-12-04 16:10:43.976353
```

次の例は TRUNC 関数内で SYSDATE 関数を使用し、時刻のない現在の日付を返します。

```
select trunc(sysdate);

trunc
------------
2008-12-04
```

次のクエリは、クエリが発行された日付からその 120 日前までの間の日付の販売情報を返します。

```
select salesid, pricepaid, trunc(saletime) as saletime, trunc(sysdate) as now
from sales
where saletime between trunc(sysdate)-120 and trunc(sysdate)
order by saletime asc;

 salesid | pricepaid |  saletime  |    now
---------+-----------+------------+------------
91535    |    670.00 | 2008-08-07 | 2008-12-05
91635    |    365.00 | 2008-08-07 | 2008-12-05
91901    |   1002.00 | 2008-08-07 | 2008-12-05
...
```

TIMEOFDAY 関数

TIMEOFDAY は文字列値として曜日、日付、および時刻を返すために使用される特別なエイリアスです。トランザクションブロック内にある場合でも、現在のステートメントの日付文字列の時刻を返します。

構文

```
TIMEOFDAY()
```

戻り型

VARCHAR

例

次の例では、TIMEOFDAY 関数を使用して、現在の日付と時刻を返します。

```
select timeofday();
```

```
timeofday
------------
Thu Sep 19 22:53:50.333525 2013 UTC
```

TIMESTAMP_CMP 関数

2 つのタイムスタンプの値を比較し、整数を返します。タイムスタンプが同じである場合、関数は `0` を返します。最初のタイムスタンプがより大きい場合、関数は `1` を返します。2 番目のタイムスタンプがより大きい場合、関数は `-1` を返します。

構文

```
TIMESTAMP_CMP(timestamp1, timestamp2)
```

引数

timestamp1

データ型 `TIMESTAMP` の列または `TIMESTAMP` 型に暗黙的に評価される式。

timestamp2

データ型 `TIMESTAMP` の列または `TIMESTAMP` 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、タイムスタンプ間を比較し、比較の結果を示します。

```
SELECT TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-01-24 06:43:29'),
 TIMESTAMP_CMP('2008-01-24 06:43:29', '2008-02-18 02:36:48'), TIMESTAMP_CMP('2008-02-18
 02:36:48', '2008-01-24 06:43:29');

timestamp_cmp  | timestamp_cmp | timestamp_cmp
---------------+---------------+---------------
             0 |            -1 |             1
```

次の例では、出品の LISTTIME と SALETIME を比較します。販売のタイムスタンプは出品のタイムスタンプより後であるため、すべての出品で TIMESTAMP_CMP の値は `-1` です。

```
select listing.listid, listing.listtime,
sales.saletime, timestamp_cmp(listing.listtime, sales.saletime)
from listing, sales
where listing.listid=sales.listid
order by 1, 2, 3, 4
limit 10;

 listid |      listtime       |      saletime       | timestamp_cmp
--------+---------------------+---------------------+---------------
      1 | 2008-01-24 06:43:29 | 2008-02-18 02:36:48 |            -1
      4 | 2008-05-24 01:18:37 | 2008-06-06 05:00:16 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-06 08:26:17 |            -1
      5 | 2008-05-17 02:29:11 | 2008-06-09 08:38:52 |            -1
      6 | 2008-08-15 02:08:13 | 2008-08-31 09:17:02 |            -1
     10 | 2008-06-17 09:44:54 | 2008-06-26 12:56:06 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-10 02:12:36 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-16 11:59:24 |            -1
     10 | 2008-06-17 09:44:54 | 2008-07-22 02:23:17 |            -1
     12 | 2008-07-25 01:45:49 | 2008-08-04 03:06:36 |            -1
(10 rows)
```

この例は、両方のタイムスタンプが同一の場合に TIMESTAMP_CMP が 0 を返すことを示しています。

```
select listid, timestamp_cmp(listtime, listtime)
from listing
order by 1 , 2
limit 10;

 listid | timestamp_cmp
--------+---------------
      1 |             0
      2 |             0
      3 |             0
      4 |             0
      5 |             0
      6 |             0
      7 |             0
      8 |             0
      9 |             0
     10 |             0
(10 rows)
```

TIMESTAMP_CMP_DATE 関数

TIMESTAMP_CMP_DATE は、タイムスタンプの値と日付を比較します。タイムスタンプと日付の値が同じである場合、関数は 0 を返します。タイムスタンプが時間的により大きい場合、関数は 1 を返します。日付がより大きい場合、関数は -1 を返します。

構文

```
TIMESTAMP_CMP_DATE(timestamp, date)
```

引数

timestamp

データ型 TIMESTAMP の列または TIMESTAMP 型に暗黙的に評価される式。

date

データ型 DATE の列または DATE 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例は LISTTIME と日付 2008-06-18 を比較します。この日付より後に作成されたリストは 1 を返し、この日付より前に作成されたリストは -1 を返します。LISTTIME の値はタイムスタンプです。

```
select listid, listtime,
timestamp_cmp_date(listtime, '2008-06-18')
from listing
order by 1, 2, 3
limit 10;


 listid |      listtime       | timestamp_cmp_date
--------+---------------------+--------------------
      1 | 2008-01-24 06:43:29 |              -1
      2 | 2008-03-05 12:25:29 |              -1
      3 | 2008-11-01 07:35:33 |               1
```

```
       4 | 2008-05-24 01:18:37 |              -1
       5 | 2008-05-17 02:29:11 |              -1
       6 | 2008-08-15 02:08:13 |               1
       7 | 2008-11-15 09:38:15 |               1
       8 | 2008-11-09 05:07:30 |               1
       9 | 2008-09-09 08:03:36 |               1
      10 | 2008-06-17 09:44:54 |              -1
(10 rows)
```

TIMESTAMP_CMP_TIMESTAMPTZ 関数

TIMESTAMP_CMP_TIMESTAMPTZ は、タイムスタンプ式の値と、タイムゾーン式を含むタイムスタンプを比較します。タイムスタンプと、タイムゾーン付きのタイムスタンプの値が同じである場合、関数は `0` を返します。タイムスタンプが時間的により大きい場合、関数は `1` を返します。タイムゾーン付きのタイムスタンプがより大きい場合、関数は `–1` を返します。

構文

```
TIMESTAMP_CMP_TIMESTAMPTZ(timestamp, timestamptz)
```

引数

timestamp

　データ型 `TIMESTAMP` の値または `TIMESTAMP` 型に暗黙的に評価される式。

timestamptz

　データ型 `TIMESTAMPTZ` の列または `TIMESTAMPTZ` 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、タイムスタンプと、タイムゾーン付のタイムスタンプを比較し、比較の結果を示します。

```
SELECT TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-01-24 06:43:29+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-01-24 06:43:29', '2008-02-18 02:36:48+00'),
 TIMESTAMP_CMP_TIMESTAMPTZ('2008-02-18 02:36:48', '2008-01-24 06:43:29+00');
```

```
timestamp_cmp_timestamptz  | timestamp_cmp_timestamptz | timestamp_cmp_timestamptz
---------------------------+---------------------------+--------------------------
             0             |            -1             |            1
```

TIMESTAMPTZ_CMP 関数

TIMESTAMPTZ_CMP は、タイムゾーン値を含む 2 つのタイムスタンプの値を比較し、整数を返します。タイムスタンプが同じである場合、関数は 0 を返します。最初のタイムスタンプが時間的により大きい場合、関数は 1 を返します。2 番目のタイムスタンプがより大きい場合、関数は –1 を返します。

構文

```
TIMESTAMPTZ_CMP(timestamptz1, timestamptz2)
```

引数

timestamptz1

データ型 `TIMESTAMPTZ` の列または `TIMESTAMPTZ` 型に暗黙的に評価される式。

timestamptz2

データ型 `TIMESTAMPTZ` の列または `TIMESTAMPTZ` 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、タイムゾーン付きのタイムスタンプ間を比較し、比較の結果を示します。

```
SELECT TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29+00'),
 TIMESTAMPTZ_CMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48+00'),
 TIMESTAMPTZ_CMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29+00');

timestamptz_cmp  | timestamptz_cmp | timestamptz_cmp
-----------------+-----------------+----------------
        0        |       -1        |       1
```

TIMESTAMPTZ_CMP_DATE 関数

TIMESTAMPTZ_CMP_DATE は、タイムスタンプの値と日付を比較します。タイムスタンプと日付の値が同じである場合、関数は 0 を返します。タイムスタンプが時間的により大きい場合、関数は 1 を返します。日付がより大きい場合、関数は –1 を返します。

構文

```
TIMESTAMPTZ_CMP_DATE(timestamptz, date)
```

引数

timestamptz

データ型 TIMESTAMPTZ の列または TIMESTAMPTZ 型に暗黙的に評価される式。

date

データ型 DATE の列または DATE 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、LISTTIME (タイムゾーン付きのタイムスタンプ) と日付 2008-06-18 を比較します。この日付より後に作成されたリストは 1 を返し、この日付より前に作成されたリストは -1 を返します。

```
select listid, CAST(listtime as timestamptz) as tstz,
timestamp_cmp_date(tstz, '2008-06-18')
from listing
order by 1, 2, 3
limit 10;


 listid |          tstz          | timestamptz_cmp_date
--------+------------------------+----------------------
      1 | 2008-01-24 06:43:29+00 |              -1
      2 | 2008-03-05 12:25:29+00 |              -1
      3 | 2008-11-01 07:35:33+00 |               1
```

```
      4 | 2008-05-24 01:18:37+00 |               -1
      5 | 2008-05-17 02:29:11+00 |               -1
      6 | 2008-08-15 02:08:13+00 |                1
      7 | 2008-11-15 09:38:15+00 |                1
      8 | 2008-11-09 05:07:30+00 |                1
      9 | 2008-09-09 08:03:36+00 |                1
     10 | 2008-06-17 09:44:54+00 |               -1
(10 rows)
```

TIMESTAMPTZ_CMP_TIMESTAMP 関数

TIMESTAMPTZ_CMP_TIMESTAMP は、タイムゾーン式を含むタイムスタンプの値とタイムスタンプ式を比較します。タイムゾーン付きのタイムスタンプと、タイムスタンプ値が同じである場合、関数は `0` を返します。タイムゾーン付きのタイムスタンプが時間的に大きい場合、関数は `1` を返します。タイムスタンプがより大きい場合、関数は `–1` を返します。

構文

```
TIMESTAMPTZ_CMP_TIMESTAMP(timestamptz, timestamp)
```

引数

timestamptz

データ型 `TIMESTAMPTZ` の列または `TIMESTAMPTZ` 型に暗黙的に評価される式。

timestamp

データ型 `TIMESTAMP` の列または `TIMESTAMP` 型に暗黙的に評価される式。

戻り型

INTEGER

例

次の例では、タイムゾーン付きタイムスタンプとタイムスタンプを比較し、比較の結果を示しています。

```
SELECT TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-01-24 06:43:29'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-01-24 06:43:29+00', '2008-02-18 02:36:48'),
 TIMESTAMPTZ_CMP_TIMESTAMP('2008-02-18 02:36:48+00', '2008-01-24 06:43:29');
```

```
timestamptz_cmp_timestamp  | timestamptz_cmp_timestamp | timestamptz_cmp_timestamp
---------------------------+---------------------------+---------------------------
              0            |             -1            |             1
```

TIMEZONE 関数

TIMEZONE は、指定されたタイムゾーンのタイムスタンプとタイムスタンプ値を返します。

タイムゾーンの設定方法の詳細と例については、「timezone」を参照してください。

タイムゾーンの変換方法の詳細と例については、「CONVERT_TIMEZONE」を参照してください。

構文

```
TIMEZONE('timezone', { timestamp | timestamptz })
```

引数

timezone

戻り値のタイムゾーン。タイムゾーンは、タイムゾーン名 (**'Africa/Kampala'** または **'Singapore'** など) またはタイムゾーンの略名 (**'UTC'** または **'PDT'** など) として指定できます。サポートされるタイムゾーン名のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_names();
```

サポートされるタイムゾーン省略形のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_abbrevs();
```

詳細な説明と例については、「タイムゾーンの使用上の注意」を参照してください。

timestamp | timestamptz

TIMESTAMP 型または TIMESTAMPTZ 型となる式、あるいは暗黙的にタイムスタンプまたはタイムゾーン付きのタイムスタンプに強制変換できる値。

戻り型

TIMESTAMP 式で使用する場合は TIMESTAMPTZ。

TIMESTAMPTZ 式で使用する場合は TIMESTAMP。

例

次の場合は、PST タイムゾーンからタイムスタンプ `2008-06-17 09:44:54` を使用して UTC タイムゾーンのタイムスタンプを返します。

```
SELECT TIMEZONE('PST', '2008-06-17 09:44:54');

timezone
------------------------
2008-06-17 17:44:54+00
```

次の場合は、UTC タイムゾーン `2008-06-17 09:44:54+00` を含むタイムスタンプを使用して PST タイムゾーンのタイムスタンプを返します。

```
SELECT TIMEZONE('PST', timestamptz('2008-06-17 09:44:54+00'));

timezone
------------------------
2008-06-17 01:44:54
```

TO_TIMESTAMP 関数

TO_TIMESTAMP は TIMESTAMP 文字列を TIMESTAMPTZ に返します。Amazon Redshift のその他の日付および時刻関数のリストについては、「日付および時刻関数」を参照してください。

構文

```
to_timestamp(timestamp, format)
```

```
to_timestamp (timestamp, format, is_strict)
```

引数

timestamp

format により指定された形式でタイムスタンプ値を表す文字列。この引数を空のままにすると、タイムスタンプ値はデフォルトで `0001-01-01 00:00:00` に設定されます。

format

timestamp 値の形式を定義する文字列リテラル。タイムゾーン (**TZ**、**tz**、または **OF**) を含む形式は、入力としてサポートされていません。有効なタイムスタンプ形式については、「日時形式の文字列」を参照してください。

is_strict

入力タイムスタンプ値が範囲外である場合にエラーを返すかどうかを指定するオプションのブール値。is_strict が TRUE に設定されている場合、範囲外の値があるとエラーが返されます。is_strict がデフォルトの FALSE に設定されている場合、オーバーフロー値が受け入れられます。

戻り型

TIMESTAMPTZ

例

次の例は、TO_TIMESTAMP 関数を使用して TIMESTAMP 文字列を TIMESTAMPTZ に変換する方法を示しています。

```
select sysdate, to_timestamp(sysdate, 'YYYY-MM-DD HH24:MI:SS') as second;

timestamp                  | second
--------------------------   ----------------------
2021-04-05 19:27:53.281812 | 2021-04-05 19:27:53+00
```

日付の TO_TIMESTAMP 部分を渡すこともできます。残りの日付部分はデフォルト値に設定されます。時刻は出力に含まれません。

```
SELECT TO_TIMESTAMP('2017','YYYY');

to_timestamp
--------------------------
2017-01-01 00:00:00+00
```

次の SQL ステートメントは、文字列「2011-12-18 24:38:15」を TIMESTAMPTZ に変換します。その結果、時間数が 24 時間を超えるため、翌日に該当する TIMESTAMPTZ になります。

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS');
```

```
to_timestamp
----------------------
2011-12-19 00:38:15+00
```

次の SQL ステートメントは、文字列「2011-12-18 24:38:15」を TIMESTAMPTZ に変換します。タイムスタンプの時刻値が 24 時間を超えているため、結果はエラーになります。

```
SELECT TO_TIMESTAMP('2011-12-18 24:38:15', 'YYYY-MM-DD HH24:MI:SS', TRUE);

ERROR:  date/time field time value out of range: 24:38:15.0
```

TRUNC 関数

TIMESTAMP を切り捨て、DATE を返します。

この関数は数値を切り捨てることもできます。詳細については、「TRUNC 関数」を参照してください。

構文

```
TRUNC(timestamp)
```

引数

timestamp

データ型 TIMESTAMP の列または TIMESTAMP 型に暗黙的に評価される式。

00:00:00 を時刻とするタイムスタンプ値を返すには、関数の結果を TIMESTAMP にキャストします。

戻り型

DATE

例

次の例では、SYSDATE 関数 (タイムスタンプを返す) の結果から日付部分を返します。

```
SELECT SYSDATE;
```

```
+----------------------------+
|         timestamp          |
+----------------------------+
| 2011-07-21 10:32:38.248109 |
+----------------------------+

SELECT TRUNC(SYSDATE);

+------------+
|   trunc    |
+------------+
| 2011-07-21 |
+------------+
```

次の例では、TRUNC 関数を TIMESTAMP 列に適用します。戻り型は日付です。

```
SELECT TRUNC(starttime) FROM event
ORDER BY eventid LIMIT 1;

+------------+
|   trunc    |
+------------+
| 2008-01-25 |
+------------+
```

次の例は、TRUNC 関数の結果を TIMESTAMP にキャストして、00:00:00 を時刻とするタイムスタンプ値を返します。

```
SELECT CAST((TRUNC(SYSDATE)) AS TIMESTAMP);

+---------------------+
|        trunc        |
+---------------------+
| 2011-07-21 00:00:00 |
+---------------------+
```

日付関数またはタイムスタンプ関数の日付部分

次のテーブルは、次の関数に対する引数として受け取る、日付部分および時刻部分の名前と略名を指定します。

- DATEADD
- DATEDIFF
- DATE_PART
- EXTRACT

日付部分または時刻部分	省略形
millennium、millennia	mil、mils
century、centuries	c、cent、cents
decade、decades	dec、decs
epoch	epoch (EXTRACT がサポート)
year、years	y、yr、yrs
quarter、quarters	qtr、qtrs
month、months	mon、mons
week、weeks	w
day of week	dayofweek、dow、dw、weekday (DATE_PART と EXTRACT 関数 がサポート) 0～6 の整数 (0 は日曜日) を返します。 Note 日付部分 DOW の動作は、日時形式の文字列に使用される日付部分 day of week (D) とは異なります。D は、整数 1～7 (日曜日が 1) に基づきます。詳細については、「日時形式の文字列」を参照してください。
day of year	dayofyear、doy、dy、yearday (EXTRACT がサポート)
day、days	d

日付部分または時刻部分	省略形
hour、hours	h、hr、hrs
minute、minutes	m、min、mins
second、seconds	s、sec、secs
millisecond、milliseconds	ms、msec、msecs、msecond、mseconds、millisec、millisecs、millisecon
microsecond、microseconds	microsec、microsecs、microsecond、usecond、useconds、us、usec、usecs
timezone、timezone_hour、timezone_minute	タイムゾーン付きタイムスタンプ (TIMESTAMPTZ) の EXTRACT でのみサポートされます。

結果のバリエーション (秒、ミリ秒、マイクロ秒)

異なる日付関数が秒、ミリ秒、またはマイクロ秒を日付部分として指定する場合、クエリ結果にわずかな違いが生じます。

- EXTRACT 関数は、上位および下位の日付部分は無視し、指定された日付部分のみの整数を返します。指定された日付部分が秒の場合、ミリ秒およびマイクロ秒は結果に含まれません。指定された日付部分がミリ秒の場合、秒およびマイクロ秒は結果に含まれません。指定された日付部分がマイクロ秒の場合、秒およびミリ秒は結果に含まれません。
- DATE_PART 関数は、指定された日付部分にかかわらず、タイムスタンプの完全な秒部分を返します。必要に応じて小数値または整数を返します。

例えば、次のクエリの結果を比較します。

```
create table seconds(micro timestamp);

insert into seconds values('2009-09-21 11:10:03.189717');

select extract(sec from micro) from seconds;

date_part
-----------
```

```
3

select date_part(sec, micro) from seconds;

pgdate_part
-------------
3.189717
```

CENTURY、EPOCH、DECADE、および MIL ノート

CENTURY または CENTURIES

Amazon Redshift は CENTURY を ###1 の年に始まり ###0 の年に終了すると解釈します。

```
select extract (century from timestamp '2000-12-16 12:21:13');
date_part
-----------
20

select extract (century from timestamp '2001-12-16 12:21:13');
date_part
-----------
21
```

EPOCH

Amazon Redshift の EPOCH の実装は、クラスターのあるタイムゾーンから独立した 1970-01-01 00:00:00.000000 に関連します。クラスターが設置されているタイムゾーンによって、時差による結果を補正する必要がある場合があります。

次の内容の例を以下に示します。

1. EVENT テーブルに基づいて EVENT_EXAMPLE というテーブルを作成します。この CREATE AS コマンドは、DATE_PART 関数を使用してデータ列 (デフォルトでは PGDATE_PART と名づけられる) を作成し、各イベントのエポック値を保存します。
2. PG_TABLE_DEF から EVENT_EXAMPLE の列とデータタイプを選択します。
3. EVENT_EXAMPLE テーブルから EVENTNAME、STARTTIME と PGDATE_PART を選択して、日付と時間の複数の形式を表示します。
4. EVENTNAME と STARTTIME をそれぞれ選択します。1 秒間隔のタイムゾーンを指定しないタイムスタンプを使って PGDATE_PART のエポック値を変換し、CONVERTED_TIMESTAMP という列にその結果を返します。

```
create table event_example
as select eventname, starttime, date_part(epoch, starttime) from event;

select "column", type from pg_table_def where tablename='event_example';

     column     |              type
---------------+-----------------------------
 eventname      | character varying(200)
 starttime      | timestamp without time zone
 pgdate_part    | double precision
(3 rows)
```

```
select eventname, starttime, pgdate_part from event_example;

   eventname           |      starttime      | pgdate_part
----------------------+---------------------+-------------
 Mamma Mia!            | 2008-01-01 20:00:00 |  1199217600
 Spring Awakening      | 2008-01-01 15:00:00 |  1199199600
 Nas                   | 2008-01-01 14:30:00 |  1199197800
 Hannah Montana        | 2008-01-01 19:30:00 |  1199215800
 K.D. Lang             | 2008-01-01 15:00:00 |  1199199600
 Spamalot              | 2008-01-02 20:00:00 |  1199304000
 Macbeth               | 2008-01-02 15:00:00 |  1199286000
 The Cherry Orchard    | 2008-01-02 14:30:00 |  1199284200
 Macbeth               | 2008-01-02 19:30:00 |  1199302200
 Demi Lovato           | 2008-01-02 19:30:00 |  1199302200


select eventname,
starttime,
timestamp with time zone 'epoch' + pgdate_part * interval '1 second' AS
 converted_timestamp
from event_example;

       eventname       |      starttime      | converted_timestamp
----------------------+---------------------+--------------------
 Mamma Mia!            | 2008-01-01 20:00:00 | 2008-01-01 20:00:00
 Spring Awakening      | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Nas                   | 2008-01-01 14:30:00 | 2008-01-01 14:30:00
 Hannah Montana        | 2008-01-01 19:30:00 | 2008-01-01 19:30:00
 K.D. Lang             | 2008-01-01 15:00:00 | 2008-01-01 15:00:00
 Spamalot              | 2008-01-02 20:00:00 | 2008-01-02 20:00:00
 Macbeth               | 2008-01-02 15:00:00 | 2008-01-02 15:00:00
```

```
The Cherry Orchard   | 2008-01-02 14:30:00 | 2008-01-02 14:30:00
Macbeth              | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
Demi Lovato          | 2008-01-02 19:30:00 | 2008-01-02 19:30:00
...
```

DECADE または DECADES

Amazon Redshift は共通カレンダーに基づいて DECADE または DECADES DATEPART を解釈します。例えば、共通カレンダーが年 1 から始まるため、最初の 10 年 (decade 1) は 0001-01-01 から 0009-12-31 であり、2 番目の 10 年 (decade 2) は 0010-01-01 から 0019-12-31 です。例えば、decade 201 は 2000-01-01 から 2009-12-31 の期間に及びます。

```
select extract(decade from timestamp '1999-02-16 20:38:40');
date_part
-----------
200

select extract(decade from timestamp '2000-02-16 20:38:40');
date_part
-----------
201

select extract(decade from timestamp '2010-02-16 20:38:40');
date_part
-----------
202
```

MIL または MILS

Amazon Redshift は MIL を #001 の年の初めの日に始まり #000 の年の最後の日に終了すると解釈します。

```
select extract (mil from timestamp '2000-12-16 12:21:13');
date_part
-----------
2

select extract (mil from timestamp '2001-12-16 12:21:13');
date_part
-----------
3
```

ハッシュ関数

トピック

ハッシュ関数は、数値入力値を別の値に変換する数学関数です。

CHECKSUM 関数

ハッシュインデックスを作成するためにチェックサム値を計算します。

構文

```
CHECKSUM(expression)
```

引数

expression

入力式のデータ型は、VARCHAR、INTEGER、または DECIMAL である必要があります。

戻り型

CHECKSUM 関数は引数を返します。

例

次の例では、COMMISSION 列のチェックサム値を計算します。

```
select checksum(commission)
from sales
order by salesid
limit 10;

checksum
----------
10920
1140
5250
2625
2310
5910
11820
2955
8865
975
(10 rows)
```

farmFingerprint64 関数

`Fingerprint64` 関数を使用して、入力引数の farmHash の値を計算する。

構文

```
farmFingerprint64(expression)
```

引数

expression

入力式は、`VARCHAR` または `VARBYTE` データ型となります。

戻り型

この `farmFingerprint64` 関数は `BIGINT` を返します。

例

次の例では、`VARCHAR` データ型として入力された `Amazon Redshift` の `farmFingerprint64` の値を返します。

```
SELECT farmFingerprint64('Amazon Redshift');
```

```

  farmfingerprint64
---------------------
 8085098817162212970
```

次の例では、VARBYTE データ型として入力された Amazon Redshift の farmFingerprint64 の値を返します。

```
SELECT farmFingerprint64('Amazon Redshift'::varbyte);
```

```

  farmfingerprint64
---------------------
 8085098817162212970
```

FUNC_SHA1 関数

SHA1 関数のシノニム。

「SHA1 関数」を参照してください。

FNV_HASH 関数

すべての基本データ型について、64 ビット FNV-1a 非暗号化ハッシュ関数を計算します。

構文

```
FNV_HASH(value [, seed])
```

引数

value

ハッシュされる入力値。Amazon Redshift は、値のバイナリ表現を使用して入力値をハッシュします。例えば、INTEGER 値は 4 バイトを使用してハッシュされ、BIGINT 値は 8 バイトを使用してハッシュされます。また、CHAR および VARCHAR 入力をハッシュしても、末尾のスペースは無視されません。

seed

ハッシュ関数の BIGINT シードはオプションです。指定しない場合、Amazon Redshift はデフォルトの FNV シードを使用します。これにより、変換や連結を行わずに、複数の列のハッシュを組み合わせることができます。

戻り型

BIGINT

例

次の例では、数値の FNV ハッシュ、文字列「Amazon Redshift」、および 2 つの連結が返されます。

```
select fnv_hash(1);
        fnv_hash
----------------------
 -5968735742475085980
(1 row)
```

```
select fnv_hash('Amazon Redshift');
      fnv_hash
---------------------
 7783490368944507294
(1 row)
```

```
select fnv_hash('Amazon Redshift', fnv_hash(1));
       fnv_hash
----------------------
 -2202602717770968555
(1 row)
```

使用に関する注意事項

- 複数の列を持つテーブルのハッシュを計算するには、最初の列の FNV ハッシュを計算し、2 番目の列のハッシュにシードとして渡します。次に、2 番目の列の FNV ハッシュを、3 番目の列のハッシュにシードとして渡します。

  次の例では、複数の列を持つテーブルをハッシュするシードを作成します。

```
select fnv_hash(column_3, fnv_hash(column_2, fnv_hash(column_1))) from sample_table;
```

- 同じプロパティを使用して、文字列の連結のハッシュを計算することができます。

```
select fnv_hash('abcd');
       fnv_hash
---------------------
 -281581062704388899
(1 row)
```

```
select fnv_hash('cd', fnv_hash('ab'));
      fnv_hash
---------------------
 -281581062704388899
(1 row)
```

- ハッシュ関数は、入力のタイプを使用して、ハッシュするバイト数を決定します。必要に応じて、キャストを使用して特定の型を適用します。

  次の例では、異なるタイプの入力を使用して別の結果を生成します。

```
select fnv_hash(1::smallint);
      fnv_hash
--------------------
 589727492704079044
(1 row)
```

```
select fnv_hash(1);
       fnv_hash
----------------------
 -5968735742475085980
(1 row)
```

```
select fnv_hash(1::bigint);
       fnv_hash
----------------------
 -8517097267634966620
(1 row)
```

MD5 関数

MD5 暗号化ハッシュ関数を使用して、可変長文字列を 32 文字の文字列に変換します。この 32 文字の文字列は、128 ビットチェックサムの 16 進値をテキストで表記したものです。

構文

```
MD5(string)
```

引数

string

可変長文字列。

戻り型

MD5 関数は、32 文字の文字列を返します。この 32 文字の文字列は、128 ビットチェックサムの 16 進値をテキストで表記したものです。

例

以下の例では、文字列 'Amazon Redshift' の 128 ビット値を示します。

```
select md5('Amazon Redshift');
md5
----------------------------------
f7415e33f972c03abd4f3fed36748f7a
(1 row)
```

SHA 関数

SHA1 関数のシノニム。

「SHA1 関数」を参照してください。

SHA1 関数

SHA1 関数は、SHA1 暗号化ハッシュ関数を使用して、可変長文字列を 40 文字の文字列に変換します。この 40 文字の文字列は、160 ビットのチェックサムの 16 進値をテキストで表記したものです。

構文

SHA1 は SHA 関数 および FUNC_SHA1 関数 のシノニムです。

```
SHA1(string)
```

引数

string

可変長文字列。

戻り型

SHA1 関数は、40 文字の文字列を返します。この 40 文字の文字列は、160 ビットのチェックサムの 16 進値をテキストで表記したものです。

例

以下の例は、単語 'Amazon Redshift' の 160 ビット値を返します。

```
select sha1('Amazon Redshift');
```

SHA2 関数

SHA2 関数は、SHA2 暗号化ハッシュ関数を使用して、可変長文字列を文字列に変換します。この文字列は、指定されたビット数のチェックサムの 16 進値をテキストで表記したものです。

構文

```
SHA2(string, bits)
```

引数

string

可変長文字列。

integer

ハッシュ関数のビット数。有効な値は 0 (256 と同じ)、224、256、384、および 512 です。

戻り型

SHA2 関数は、チェックサムの 16 進値をテキストで表記した文字列を返します。また、ビット数が無効な場合は、空の文字列を返します。

例

次の例では、単語 'Amazon Redshift' の 256 ビット値を返します。

```
select sha2('Amazon Redshift', 256);
```

MURMUR3_32_HASH

MURMUR3_32_HASH 関数は、数値型や文字列型を含むすべての一般的なデータ型の 32 ビット Murmur3A 非暗号化ハッシュを計算します。

構文

```
MURMUR3_32_HASH(value [, seed])
```

引数

value

ハッシュする入力値。Amazon Redshift は、入力値のバイナリ表現をハッシュします。この動作は FNV_HASH 関数 と似ていますが、値は Apache Iceberg の 32 ビット Murmur3 ハッシュ仕様で指定されているバイナリ表現に変換されます。

seed

ハッシュ関数の INT シード。この引数はオプションです。指定しない場合、Amazon Redshift はデフォルトシードの 0 を使用します。これにより、変換や連結を行わずに、複数の列のハッシュを組み合わせることができます。

戻り型

この関数は INT を返します。

例

次の例では、数値の Murmur3 ハッシュ、文字列「Amazon Redshift」、および両方の連結を返します。

```
select MURMUR3_32_HASH(1);

    MURMUR3_32_HASH
----------------------
 1392991556
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift');

    MURMUR3_32_HASH
----------------------
 -1563580564
(1 row)
```

```
select MURMUR3_32_HASH('Amazon Redshift', MURMUR3_32_HASH(1));

    MURMUR3_32_HASH
----------------------
 -1346554171
(1 row)
```

使用に関する注意事項

複数の列を持つテーブルのハッシュを計算するには、最初の列の Murmur3 ハッシュを計算し、それをシードとして 2 番目の列のハッシュに渡します。次に、2 番目の列の Murmur3 ハッシュをシードとして 3 番目の列のハッシュに渡します。

次の例では、複数の列を持つテーブルをハッシュするシードを作成します。

```
select MURMUR3_32_HASH(column_3, MURMUR3_32_HASH(column_2, MURMUR3_32_HASH(column_1)))
 from sample_table;
```

同じプロパティを使用して、文字列の連結のハッシュを計算することができます。

```
select MURMUR3_32_HASH('abcd');

   MURMUR3_32_HASH
---------------------
 1139631978
(1 row)
```

```
select MURMUR3_32_HASH('cd', MURMUR3_32_HASH('ab'));

   MURMUR3_32_HASH
---------------------
 1711522338
(1 row)
```

ハッシュ関数は、入力のタイプを使用して、ハッシュするバイト数を決定します。必要に応じて、キャストを使用して特定の型を適用します。

以下の例では、複数の異なる入力タイプを使用して複数の異なる結果を生成します。

```
select MURMUR3_32_HASH(1, MURMUR3_32_HASH(1));

   MURMUR3_32_HASH
--------------------
 -1193428387
(1 row)
```

```
select MURMUR3_32_HASH(1);

   MURMUR3_32_HASH
----------------------
 1392991556
(1 row)
```

```
select MURMUR3_32_HASH(1, MURMUR3_32_HASH(2));

   MURMUR3_32_HASH
----------------------
 1179621905
(1 row)
```

HyperLogLog 関数

以下に、Amazon Redshift でサポートされる SQL の HyperLogLog 関数の説明を示します。

トピック

- HLL 関数

- [HLL_CREATE_SKETCH 関数]
- [HLL_CARDINALITY 関数]
- [HLL_COMBINE 関数]
- [HLL_COMBINE_SKETCHES 関数]

HLL 関数

HLL 関数は、入力式の値の HyperLogLog 基数を返します。HLL 関数は、HLLSKETCH データ型以外のすべてのデータ型で動作します。HLL 関数は NULL 値を無視します。テーブルに行がない場合、またはすべての行が NULL の場合、結果の基数は 0 になります。

構文

```
HLL (aggregate_expression)
```

引数

aggregate_expression

集計する値を返す任意の有効な式 (列名など)。この関数は、HLLSKETCH、GEOMETRY、GEOGRAPHY、および VARBYTE を除く,すべてのデータ型を入力としてサポートします。

戻り型

HLL 関数は、BIGINT または INT8 の値を返します。

例

次の例では、テーブル `a_table` の列 `an_int` の基数を返します。

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll(an_int) AS cardinality FROM a_table;
cardinality
-------------
4
```

HLL_CREATE_SKETCH 関数

HLL_CREATE_SKETCH 関数は、入力式の値をカプセル化する HLLSKETCH データ型を返します。HLL_CREATE_SKETCH 関数は、任意のデータ型で動作し、NULL 値を無視します。テーブルに行がない場合、またはすべての行が NULL の場合、結果のスケッチには `{"version":1,"logm":15,"sparse":{"indices":[],"values":[]}}` などのインデックスと値のペアがありません。

構文

```
HLL_CREATE_SKETCH (aggregate_expression)
```

引数

aggregate_expression

集計する値を返す任意の有効な式 (列名など)。NULL 値は無視されます。この関数は、HLLSKETCH、GEOMETRY、GEOGRAPHY、および VARBYTE を除く,すべてのデータ型を入力としてサポートします。

戻り型

HLL_CREATE_SKETCH 関数は、HLLSKETCH 値を返します。

例

次の例では、テーブル `a_table` で列 `an_int` の HLLSKETCH タイプを返します。JSON オブジェクトは、スケッチのインポート、エクスポート、または印刷時に、スパースの HyperLogLog スケッチを表すために使用されます。デンス HyperLogLog スケッチを表すには、文字列表現 (Base64 形式) が使用されます。

```
CREATE TABLE a_table(an_int INT);
INSERT INTO a_table VALUES (1), (2), (3), (4);

SELECT hll_create_sketch(an_int) AS sketch FROM a_table;
sketch
-------------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,47158030],"values":[1,2,1,1]}}
(1 row)
```

HLL_CARDINALITY 関数

HLL_CARDINALITY 関数は、入力 HLLSKETCH データ型の基数を返します。

構文

```
HLL_CARDINALITY (hllsketch_expression)
```

引数

hllsketch_expression

列名など、HLLSKETCH 型に評価される有効な式。入力値は HLLSKETCH データ型です。

戻り型

HLL_CARDINALITY 関数は、BIGINT または INT8 の値を返します。

例

次の例では、テーブル `hll_table` の列 `sketch` の基数を返します。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_cardinality(sketch) AS cardinality FROM hll_table;
cardinality
-------------
6
4
(2 rows)
```

HLL_COMBINE 関数

HLL_COMPINE 集計関数は、すべての入力 HLLSKETCH 値を組み合わせた HLLSKETCH データ型を返します。

2 つ以上の HyperLogLog スケッチの組み合わせは、各入力スケッチが表す個別値の結合に関する情報をカプセル化する新しい HLLSKETCH です。スケッチを組み合わせた後、Amazon Redshift は 2 つ以上のデータセット結合の基数を抽出します。複数のスケッチを組み合わせる方法の詳細については、「[例: 複数のスケッチを組み合わせて HyperLogLog スケッチを返す]」を参照してください。

構文

```
HLL_COMBINE (hllsketch_expression)
```

引数

hllsketch_expression

列名など、HLLSKETCH 型に評価される有効な式。入力値は HLLSKETCH データ型です。

戻り型

HLL_COMPINE 関数は、HLLSKETCH 型を返します。

例

次の例は、テーブル `hll_table` 内の結合された HLLSKETCH 値を返します。

```
CREATE TABLE a_table(an_int INT, b_int INT);
INSERT INTO a_table VALUES (1,1), (2,1), (3,1), (4,1), (1,2), (2,2), (3,2), (4,2),
 (5,2), (6,2);

CREATE TABLE hll_table (sketch HLLSKETCH);
INSERT INTO hll_table select hll_create_sketch(an_int) from a_table group by b_int;

SELECT hll_combine(sketch) AS sketches FROM hll_table;
sketches
----------------------------------------------------------------------------------------------------
{"version":1,"logm":15,"sparse":{"indices":
[20812342,20850007,22362299,40314817,42650774,47158030],"values":[1,2,1,3,2,1]}}
(1 row)
```

HLL_COMBINE_SKETCHES 関数

HLL_COMBINE_SKETCHES は、2 つの HLLSKETCH 値を入力として使用し、それらを単一の HLLSKETCH に組み合わせるスカラー関数です。

2 つ以上の HyperLogLog スケッチの組み合わせは、各入力スケッチが表す個別値の結合に関する情報をカプセル化する新しい HLLSKETCH です。

構文

```
HLL_COMBINE_SKETCHES (hllsketch_expression1, hllsketch_expression2)
```

引数

hllsketch_expression1 と hllsketch_expression2

列名など、HLLSKETCH 型に評価される有効な式。

戻り型

HLL_COMBINE_SKETCHES 関数は、HLLSKETCH 型を返します。

例

次の例は、テーブル `hll_table` 内の結合された HLLSKETCH 値を返します。

```
WITH tbl1(x, y)
     AS (SELECT Hll_create_sketch(1),
                Hll_create_sketch(2)
         UNION ALL
         SELECT Hll_create_sketch(3),
                Hll_create_sketch(4)
         UNION ALL
         SELECT Hll_create_sketch(5),
                Hll_create_sketch(6)
         UNION ALL
         SELECT Hll_create_sketch(7),
                Hll_create_sketch(8)),
     tbl2(x, y)
     AS (SELECT Hll_create_sketch(9),
                Hll_create_sketch(10)
         UNION ALL
         SELECT Hll_create_sketch(11),
                Hll_create_sketch(12)
         UNION ALL
         SELECT Hll_create_sketch(13),
```

```
                Hll_create_sketch(14)
         UNION ALL
         SELECT Hll_create_sketch(15),
                Hll_create_sketch(16)
         UNION ALL
         SELECT Hll_create_sketch(NULL),
                Hll_create_sketch(NULL)),
     tbl3(x, y)
     AS (SELECT *
         FROM   tbl1
         UNION ALL
         SELECT *
         FROM   tbl2)
SELECT Hll_combine_sketches(x, y)
FROM   tbl3;
```

JSON 関数

トピック

- JSON_PARSE 関数
- CAN_JSON_PARSE 関数
- JSON_SERIALIZE 関数
- JSON_SERIALIZE_TO_VARBYTE 関数
- テキストベースの JSON 関数

> **Note**
>
> JSON を操作するには、次の関数を使用することをお勧めします。
>
> - JSON_PARSE 関数
> - CAN_JSON_PARSE 関数
> - JSON_SERIALIZE 関数
> - JSON_SERIALIZE_TO_VARBYTE 関数
>
> JSON_PARSE では、取り込み時に JSON テキストを SUPER 型の値に 1 回変換するだけで済みます。その後は、SUPER 値に対して操作を行うことができます。Amazon Redshift は、テキストベースの JSON 関数の出力である VARCHAR よりも、SUPER 値をより効率的

に解析します。SUPER データ型の操作の詳細については、「Amazon Redshift の半構造化データ」を参照してください。

相対的に小さい一連のキーと値のペアを格納する必要がある場合は、データを JSON 形式で格納すると、スペースを節約できます。JSON 文字列は単一の列に格納できるため、データを表形式で格納するより、JSON を使用する方が効率的である可能性があります。例えば、スパース表があるとします。その表では、すべての属性を完全に表すために列が多数必要ですが、指定された行または指定された列のほとんどの列値が NULL であるとします。格納に JSON を使用することによって、行のデータを単一の JSON 文字列にキーと値のペアで格納できる可能性があり、格納データの少ない表の列を除去できる可能性があります。

さらに、JSON スキーマの変更時に、テーブルに列を追加することなく、JSON 文字列を簡単に変更して追加のキーと値のペアを保存できます。

JSON の使用は控えめにすることをお勧めします。JSON では、単一の列にさまざまなデータが保存され、Amazon Redshift の列保存アーキテクチャが使用されません。したがってこの形式は、大きいデータセットの保存には適していません。Amazon Redshift は CHAR 列と VARCHAR 列で JSON 関数をサポートしていますが、JSON のシリアル化形式でデータを処理するには、SUPER を使用することをお勧めします。SUPER では、階層データを効率的にクエリできる、ポスト解析スキーマレス表現が使用されています。SUPER データ型の詳細については、Amazon Redshift の半構造化データを参照してください。

JSON は UTF-8 でエンコードされたテキスト文字列を使用するため、JSON 文字列を CHAR データ型または VARCHAR データ型として格納できます。

JSON 文字列は、以下のルールに従って、正しくフォーマットされた JSON である必要があります。

- ルートレベルの JSON には、JSON オブジェクトまたは JSON 配列を使用できます。JSON オブジェクトは、順序が設定されていない一連のキーと値のペアがカンマで区切られ、中括弧で囲まれたものです。

  例えば、`{"one":1, "two":2}`
- JSON 配列は、順序付けられた一連のカンマ区切り値が角括弧で囲まれたものです。

  例は次のとおりです: `["first", {"one":1}, "second", 3, null]`
- JSON 配列は、0 から始まるインデックスを使用します。配列内の最初の要素の位置は 0 です。JSON のキーと値のペアでは、キーは二重引用符で囲まれた文字列です。

- JSON 値には次のいずれかを指定できます。
  - JSON オブジェクト
  - array
  - string
    - 二重引用符で表される
  - 数値
    - 整数、小数、浮動小数点数を含む
  - ブール値
  - null
- 空のオブジェクトおよび空の配列は、有効な JSON 値です。
- JSON フィールドでは、大文字と小文字が区別されます。
- JSON 構造要素 ({ }, [ ] など) は無視されます。

Amazon Redshift JSON 関数および Amazon Redshift COPY コマンドは、同じメソッドを使用して JSON 形式のデータを操作します。JSON の操作方法の詳細については、「JSON 形式からの COPY」を参照してください。

JSON_PARSE 関数

JSON_PARSE 関数は、JSON 形式のデータを解析して、そのデータを SUPER 表現に変換します。

INSERT または UPDATE コマンドを使用して、データを SUPER 型として取り込むには、JSON_PARSE 関数を使用します。JSON_PARSE() を使用して JSON 文字列を SUPER 値に解析する場合、特定の制限が適用されます。詳細については、「SUPER の解析オプション」を参照してください。

構文

```
JSON_PARSE( {json_string | binary_value} )
```

引数

json_string

シリアル化された JSON を VARBYTE 型または VARCHAR 型として返す式。

binary_value

VARBYTE データ型のバイナリ値。

戻り型

SUPER

例

JSON 配列 [10001,10002,"abc"] を SUPER データ型に変換するには、次の例を使用します。

```
SELECT JSON_PARSE('[10001,10002,"abc"]');

+---------------------+
|     json_parse      |
+---------------------+
| [10001,10002,"abc"] |
+---------------------+
```

関数が JSON 配列を SUPER データ型に変換したことを確認するには、次の例を使用します。詳細については、「JSON_TYPEOF 関数」を参照してください。

```
SELECT JSON_TYPEOF(JSON_PARSE('[10001,10002,"abc"]'));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

CAN_JSON_PARSE 関数

CAN_JSON_PARSE 関数は JSON 形式でデータを解析し、JSON_PARSE 関数を使用して結果を SUPER 値に変換できる場合は true を返します。

構文

```
CAN_JSON_PARSE( {json_string | binary_value} )
```

引数

json_string

シリアル化された JSON を VARCHAR 形式で返す式。

binary_value

VARBYTE データ型のバイナリ値。

戻り型

BOOLEAN

使用に関する注意事項

- CAN_JSON_PARSE は空の文字列に対して false を返します。入力引数が null の場合は、NULL を返します。

例

次の例は、CASE 条件を使用して適切に形成された JSON 配列に対する CAN_JSON_PARSE の実行を示しています。これは true を返すため、Amazon Redshift はサンプル値に対して JSON_PARSE 関数を実行します。

```
SELECT CASE
            WHEN CAN_JSON_PARSE('[10001,10002,"abc"]')
            THEN JSON_PARSE('[10001,10002,"abc"]')
        END;

 case
---------------------
'[10001,10002,"abc"]'
```

次の例は、CASE 条件を使用して JSON 形式ではない値に対する CAN_JSON_PARSE の実行を示しています。これは false を返すため、Amazon Redshift は代わりに CASE 条件の ELSE 句でセグメントを返します。

```
SELECT CASE
            WHEN CAN_JSON_PARSE('This is a string.')
            THEN JSON_PARSE('This is a string.')
```

```
            ELSE 'This is not JSON.'
        END;

 case
---------------------
"This is not JSON."
```

JSON_SERIALIZE 関数

JSON_SERIALIZE 関数は、RFC 8259 に従って、SUPER 式をテキスト JSON 表現でシリアル化します。詳細については、The JavaScript Object Notation (JSON) Data Interchange Formatを参照してください。

SUPER のサイズ制限はブロックでの制限とほぼ同じで、VARCHAR の制限は SUPER でのサイズ制限よりも小さくなっています。したがって、JSON_SERIALIZE 関数は、JSON 形式がシステムの VARCHAR 制限を超えた場合はエラーを返します。SUPER 式のサイズを確認するには、JSON_SIZE 関数を参照してください。

構文

```
JSON_SERIALIZE(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

VARCHAR

例

SUPER 値を文字列にシリアル化するには、次の例を使用します。

```
SELECT JSON_SERIALIZE(JSON_PARSE('[10001,10002,"abc"]'));

+---------------------+
|   json_serialize    |
+---------------------+
```

```
| [10001,10002,"abc"] |
+---------------------+
```

JSON_SERIALIZE_TO_VARBYTE 関数

JSON_SERIALIZE_TO_VARBYTE 関数は、SUPER 値を JSON_SERIALIZE() と同様の JSON 文字列に変換しますが、この関数は VARBYTE 値に保存されます。

構文

```
JSON_SERIALIZE_TO_VARBYTE(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

VARBYTE

例

SUPER 値をシリアル化し、結果を VARBYTE 形式で返すには、次の例を使用します。

```
SELECT JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'));

+-----------------------------------------+
|        json_serialize_to_varbyte        |
+-----------------------------------------+
| 5b31303030312c31303030322c22616263225d |
+-----------------------------------------+
```

SUPER 値をシリアル化し、結果を VARCHAR 形式にキャストするには、次の例を使用します。詳細については、「CAST 関数」を参照してください。

```
SELECT CAST((JSON_SERIALIZE_TO_VARBYTE(JSON_PARSE('[10001,10002,"abc"]'))) AS VARCHAR);

+---------------------------+
```

```
| json_serialize_to_varbyte |
+---------------------------+
| [10001,10002,"abc"]       |
+---------------------------+
```

テキストベースの JSON 関数

以下の関数は、JSON 値を VARCHAR として解析します。JSON を解析するには、JSON 値を SUPER として解析する以下の関数を代わりに使用することをお勧めします。Amazon Redshift は、VARCHAR よりも SUPER 値をより効率的に解析します。

- JSON_PARSE 関数
- CAN_JSON_PARSE 関数
- JSON_SERIALIZE 関数
- JSON_SERIALIZE_TO_VARBYTE 関数

トピック

IS_VALID_JSON 関数

> **Note**
>
> CAN_JSON_PARSE および関連する関数は、JSON 値を SUPER として解析します。Amazon Redshift は、VARCHAR よりも SUPER をより効率的に解析します。IS_VALID_JSON を使用する代わりに、CAN_JSON_PARSE 関数 を使用して JSON 文字列を検証することをお勧めします。

IS_VALID_JSON 関数は、JSON 文字列を確認します。この関数は、文字列が正しい JSON 形式になっている場合、`true` のブール値を返し、正しい形式ではない場合は、`false` を返します。JSON 配列を検証するには、IS_VALID_JSON_ARRAY 関数を使用します。

詳細については、「JSON 関数」を参照してください。

構文

```
IS_VALID_JSON('json_string')
```

引数

json_string

評価して JSON 文字列を返す文字列または式。

戻り型

BOOLEAN

例

テーブルを作成し、テスト用の JSON 文字列を挿入するには、次の例を使用します。

```
CREATE TABLE test_json(id int IDENTITY(0,1), json_strings VARCHAR);

-- Insert valid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{"a":2}'),
('{"a":{"b":{"c":1}}}'),
('{"a": [1,2,"b"]}');

-- Insert invalid JSON strings --
INSERT INTO test_json(json_strings) VALUES
('{{}}'),
('{1:"a"}'),
('[1,2,3]');
```

前の例の文字列を検証するには、次の例を使用します。

```
SELECT id, json_strings, IS_VALID_JSON(json_strings)
FROM test_json
ORDER BY id;

+----+---------------------+---------------+
```

```
| id |     json_strings      | is_valid_json |
+----+---------------------+---------------+
|  0 | {"a":2}             | true          |
|  4 | {"a":{"b":{"c":1}}} | true          |
|  8 | {"a": [1,2,"b"]}    | true          |
| 12 | {{}}                | false         |
| 16 | {1:"a"}             | false         |
| 20 | [1,2,3]             | false         |
+----+---------------------+---------------+
```

IS_VALID_JSON_ARRAY 関数

> **Note**
>
> JSON_PARSE および関連する関数は、JSON 値を SUPER として解析します。Amazon Redshift は、VARCHAR よりも SUPER をより効率的に解析します。IS_VALID_JSON_ARRAY を使用する代わりに、JSON_PARSE 関数 を使用して JSON 文字列を解析し、SUPER 値を取得することをお勧めします。次に、IS_ARRAY 関数 関数を使用して、配列が適切に形成されていることを確認します。

IS_VALID_JSON_ARRAY 関数は、JSON 配列を確認します。この関数は、配列が正しい JSON 形式になっている場合、`true` のブール値を返し、正しい形式ではない場合は、`false` を返します。JSON 文字列を検証するには、IS_VALID_JSON 関数を使用します。

詳細については、「JSON 関数」を参照してください。

構文

```
IS_VALID_JSON_ARRAY('json_array')
```

引数

json_array

評価して JSON 配列を返す文字列または式。

戻り型

BOOLEAN

例

テーブルを作成し、テスト用の JSON 文字列を挿入するには、次の例を使用します。

```
CREATE TABLE test_json_arrays(id int IDENTITY(0,1), json_arrays VARCHAR);

-- Insert valid JSON array strings --
INSERT INTO test_json_arrays(json_arrays)
VALUES('[]'),
('["a","b"]'),
('["a",["b",1,["c",2,3,null]]]');

-- Insert invalid JSON array strings --
INSERT INTO test_json_arrays(json_arrays)
VALUES('{"a":1}'),
('a'),
('[1,2,]');
```

前の例の文字列を検証するには、次の例を使用します。

```
SELECT json_arrays, IS_VALID_JSON_ARRAY(json_arrays)
FROM test_json_arrays ORDER BY id;

+------------------------------+--------------------+
|         json_arrays          | is_valid_json_array |
+------------------------------+--------------------+
| []                           | true               |
| ["a","b"]                    | true               |
| ["a",["b",1,["c",2,3,null]]] | true               |
| {"a":1}                      | false              |
| a                            | false              |
| [1,2,]                       | false              |
+------------------------------+--------------------+
```

JSON_ARRAY_LENGTH 関数

> **Note**
>
> JSON_PARSE および関連する関数は、JSON 値を SUPER として解析します。Amazon Redshift は、VARCHAR よりも SUPER をより効率的に解析します。

JSON_ARRAY_LENGTH を使用する代わりに、JSON_PARSE 関数 を使用して JSON 文字列を解析し、SUPER 値を取得することをお勧めします。次に、get_array_length 関数 を使用して配列の長さを取得します。

JSON_ARRAY_LENGTH 関数は、JSON 文字列の外部配列内の要素数を返します。null_if_invalid の引数が `true` に設定され、JSON 文字列が無効になっている場合、この関数はエラーを返す代わりに `NULL` を返します。

詳細については、「JSON 関数」を参照してください。

構文

```
JSON_ARRAY_LENGTH('json_array' [, null_if_invalid ] )
```

引数

json_array

正しくフォーマットされた JSON 配列。

null_if_invalid

(オプション) 入力 JSON 文字列が無効である場合に、エラーを返す代わりに `NULL` を返すかどうかを指定する `BOOLEAN` 値。JSON が無効な場合に `NULL` を返すには、`true` (`t`) を指定します。JSON が無効な場合にエラーを返すには、`false` (f) を指定します。デフォルト: `false`。

戻り型

`INTEGER`

例

配列内の要素数を返すには、次の例を使用します。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14]');

+-------------------+
| json_array_length |
+-------------------+
|                 5 |
```

```
+--------------------+
```

JSON が無効であるためエラーを返すには、次の例を使用します。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14');

ERROR: invalid json array object [11,12,13,{"f1":21,"f2":[25,26]},14
```

null_if_invalid を true に設定し、ステートメントが無効な JSON のエラーを返す代わりに NULL を返すようにするには、次の例を使用します。

```
SELECT JSON_ARRAY_LENGTH('[11,12,13,{"f1":21,"f2":[25,26]},14',true);

+--------------------+
| json_array_length |
+--------------------+
| NULL              |
+--------------------+
```

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 関数

> **Note**
>
> JSON_PARSE および関連する関数は、JSON 値を SUPER として解析します。Amazon Redshift は、VARCHAR よりも SUPER をより効率的に解析します。JSON_EXTRACT_ARRAY_ELEMENT_TEXT を使用する代わりに、JSON_PARSE 関数 を使用して JSON 文字列を解析し、SUPER 値を取得することをお勧めします。次に、`value[element position]` 構文を使用し、必要とする配列インデックスが含まれている要素をクエリします。SUPER 値の配列要素に対するクエリ実行の詳細については、「半構造化データのクエリ」を参照してください。

JSON_EXTRACT_ARRAY_ELEMENT_TEXT 関数は、JSON 文字列の最外部の配列内の JSON 配列要素 (0 から始まるインデックスを使用) を返します。配列内の最初の要素の位置は 0 です。インデックスが負または範囲外である場合、JSON_EXTRACT_ARRAY_ELEMENT_TEXT は空の文字列を返します。null_if_invalid の引数が `true` に設定され、JSON 文字列が無効になっている場合、この関数はエラーを返す代わりに `NULL` を返します。

詳細については、「JSON 関数」を参照してください。

構文

```
JSON_EXTRACT_ARRAY_ELEMENT_TEXT('json string', pos [, null_if_invalid ] )
```

引数

json_string

正しくフォーマットされた JSON 文字列。

pos

返される配列要素のインデックスを表す INTEGER (0 から始まる配列インデックスを使用)。

null_if_invalid

(オプション) 入力 JSON 文字列が無効である場合に、エラーを返す代わりに NULL を返すかどうかを指定する BOOLEAN 値。JSON が無効な場合に NULL を返すには、true (t) を指定します。JSON が無効な場合にエラーを返すには、false (f) を指定します。デフォルト: false。

戻り型

VARCHAR

pos によって参照される JSON 配列要素を表す VARCHAR 文字列。

例

配列の位置 2 の要素 (0 から始まる配列インデックスの 3 番目の要素) を返すには、次の例を使用します。

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('[111,112,113]', 2);

+---------------------------------+
| json_extract_array_element_text |
+---------------------------------+
|                             113 |
+---------------------------------+
```

JSON が無効であるためエラーを返すには、次の例を使用します。

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1);

ERROR: invalid json array object ["a",["b",1,["c",2,3,null,]]]
```

null_if_invalid を true に設定し、ステートメントが無効な JSON のエラーを返す代わりに NULL を返すようにするには、次の例を使用します。

```
SELECT JSON_EXTRACT_ARRAY_ELEMENT_TEXT('["a",["b",1,["c",2,3,null,]]]',1,true);

+-------------------------------+
| json_extract_array_element_text |
+-------------------------------+
| NULL                          |
+-------------------------------+
```

JSON_EXTRACT_PATH_TEXT 関数

> **Note**
>
> JSON_PARSE および関連する関数は、JSON 値を SUPER として解析します。Amazon Redshift は、VARCHAR よりも SUPER をより効率的に解析します。JSON_EXTRACT_PATH_TEXT を使用する代わりに、JSON_PARSE 関数 を使用して JSON 文字列を解析し、SUPER 値を取得することをお勧めします。次に、`value.attribute` 構文を使用する対象の要素をクエリします。SUPER 値の配列要素に対するクエリ実行の詳細については、「半構造化データのクエリ」を参照してください。

JSON_EXTRACT_PATH_TEXT 関数は、JSON 文字列内の一連のパス要素から参照されるキーと値のペアを値として返します。JSON パスは、最大 5 レベルの深さまでネストできます。パス要素では、大文字と小文字が区別されます。JSON 文字列にパス要素が存在しない場合、JSON_EXTRACT_PATH_TEXT は NULL を返します。

null_if_invalid の引数が `true` に設定され、JSON 文字列が無効になっている場合、この関数はエラーを返す代わりに NULL を返します。

JSON_EXTRACT_PATH_TEXT のデータサイズは最大 64 KB です。JSON レコードのサイズが 64 KB を超えると、JSON_EXTRACT_PATH_TEXT での処理エラーになります。

その他の JSON 関数については、「JSON 関数」を参照してください。JSON の操作方法の詳細については、「JSON 形式からの COPY」を参照してください。

構文

```
JSON_EXTRACT_PATH_TEXT('json_string', 'path_elem' [,'path_elem'[, …] ]
 [, null_if_invalid ] )
```

引数

json_string

正しくフォーマットされた JSON 文字列。

path_elem

JSON 文字列内のパス要素。1 つのパス要素が必要です。パス要素は、最大 5 レベルの深さまで、追加で指定できます。

null_if_invalid

(オプション) 入力 JSON 文字列が無効である場合に、エラーを返す代わりに NULL を返すかどうかを指定する BOOLEAN 値。JSON が無効な場合に NULL を返すには、true (t) を指定します。JSON が無効な場合にエラーを返すには、false (f) を指定します。デフォルト: false。

JSON 文字列では、Amazon Redshift は \n を改行文字として、\t をタブ文字として認識します。バックスラッシュをロードするには、バックスラッシュをバックスラッシュでエスケープします (\\)。詳細については、「JSON のエスケープ文字」を参照してください。

戻り型

VARCHAR

パス要素から参照される JSON 値を表す VARCHAR 文字列。

例

パス 'f4', 'f6' の値を返すには、次の例を使用します。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}','f4', 'f6');

+------------------------+
| json_extract_path_text |
+------------------------+
```

```
| star                   |
+------------------------+
```

JSON が無効であるためエラーを返すには、次の例を使用します。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4', 'f6');

ERROR: invalid json object {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}
```

null_if_invalid を true に設定し、ステートメントがエラーを返す代わりに NULL を返すようにするには、次の例を使用します。

```
SELECT JSON_EXTRACT_PATH_TEXT('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}','f4',
 'f6',true);

+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
+------------------------+
```

パス `'farm', 'barn', 'color'` の値を返すには (取得される値は第 3 レベルにある)、次の例を使用します。読みやすくするために、このサンプルは JSON Lint ツールを使用してフォーマットされています。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}', 'farm', 'barn', 'color');
+------------------------+
| json_extract_path_text |
+------------------------+
| red                    |
+------------------------+
```

`'color'` 要素がないため NULL を返すには、次の例を使用します。このサンプルは JSON Lint ツールでフォーマットされています。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
    "farm": {
        "barn": {}
    }
}', 'farm', 'barn', 'color');

+------------------------+
| json_extract_path_text |
+------------------------+
| NULL                   |
+------------------------+
```

JSON が有効な場合、欠落している要素を抽出しようとすると NULL が返されます。

パス 'house', 'appliances', 'washing machine', 'brand' の値を返すには、次の例を使用します。

```
SELECT JSON_EXTRACT_PATH_TEXT('{
  "house": {
    "address": {
      "street": "123 Any St.",
      "city": "Any Town",
      "state": "FL",
      "zip": "32830"
    },
    "bathroom": {
      "color": "green",
      "shower": true
    },
    "appliances": {
      "washing machine": {
        "brand": "Any Brand",
        "color": "beige"
      },
      "dryer": {
        "brand": "Any Brand",
        "color": "white"
      }
    }
  }
}', 'house', 'appliances', 'washing machine', 'brand');

+------------------------+
```

```
| json_extract_path_text |
+------------------------+
| Any Brand              |
+------------------------+
```

次の例は、サンプルテーブルを作成し、SUPER 値を入力して、両方の行の `'f2'` パスの値を返します。

```
CREATE TABLE json_example(id INT, json_text SUPER);

INSERT INTO json_example VALUES
(1, JSON_PARSE('{"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}')),
(2, JSON_PARSE('{
    "farm": {
        "barn": {
            "color": "red",
            "feed stocked": true
        }
    }
}'));

SELECT * FROM json_example;
id          | json_text
------------+--------------------------------------------
1           | {"f2":{"f3":1},"f4":{"f5":99,"f6":"star"}}
2           | {"farm":{"barn":{"color":"red","feed stocked":true}}}


SELECT id, JSON_EXTRACT_PATH_TEXT(JSON_SERIALIZE(json_text), 'f2') FROM json_example;

id          | json_text
------------+--------------------------------------------
1           | {"f3":1}
2           |
```

機械学習機能

Amazon Redshift 機械学習(ML) を使用すると、SQL ステートメントを使用して機械学習モデルをトレーニングし、予測のために SQL クエリでそれらを呼び出すことができます。Amazon Redshift モデルの説明可能性には、トレーニングデータの各属性が予測結果にどのように寄与しているかを理解するのに役立つ特徴量の重要度の値が含まれています。

以下に、Amazon Redshift でサポートされる SQL の機械学習 関数について示します。

トピック

- EXPLAIN_MODEL関数

EXPLAIN_MODEL関数

EXPLAIN_MODEL 関数は、JSON 形式のモデルの説明可能性レポートを含む SUPER データ型を返します。説明可能性レポートには、すべてのモデル機能の Shapley 値に関する情報が含まれています。

EXPLAIN_MODEL 関数は、現在、XGBoost モデルのAUTO ON oまたはAUTO OFF のみをサポートしています。

説明可能性レポートが利用できない場合、関数はモデルの進行状況を示すステータスを返します。たとえば`Waiting for training job to complete`、`Waiting for processing job to complete`、および`Processing job failed`に適用されます。

CREATE MODEL ステートメントを実行すると、説明の状態は次のようになります`Waiting for training job to complete`。モデルがトレーニングされ、説明要求が送信されると、説明の状態は次のようになります`Waiting for processing job to complete`。モデルの説明が正常に完了すると、完全な説明可能性レポートが使用可能になります。それ以外の場合、状態は次のようになります`Processing job failed`。

CREATE MODEL ステートメントを実行するときに、オプションの `MAX_RUNTIME` パラメータを使用してトレーニングにかかる最大時間を指定できます。モデルの作成がその時間に達すると、Amazon Redshift はモデルの作成を停止します。オートパイロットモデルの作成中にその制限時間に達すると、Amazon Redshift はその時点までの最高のモデルを返します。モデルトレーニングが終了すると、モデルの説明可能性を使用できるようになります。`MAX_RUNTIME` を短い時間に設定すると、説明可能性レポートは利用できない場合があります。トレーニング時間はさまざまであり、モデルの複雑さ、データサイズ、その他の要因によって異なります。

構文

```
EXPLAIN_MODEL ('schema_name.model_name')
```

引数

schema_name

スキーマの名前。スキーマ名 が指定されていない場合は、現在のスキーマが選択されます。

model_name

モデルの名前です。スキーマ内のモデル名は一意でなければなりません。

戻り型

EXPLAIN_MODEL関数は、次に示すように、SUPERデータ型を返します。

```
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05,"x1":0.10,"x2":0.30,"x3":0.15},"expected_value":0.50}}}}
```

例

次の例では、説明状態を返します`waiting for training job to complete`。

```
select explain_model('customer_churn_auto_model');
                 explain_model
---------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

モデルの説明が正常に完了すると、次のように完全な説明可能性レポートが利用できます。

```
select explain_model('customer_churn_auto_model');
                                       explain_model
--------------------------------------------------------------------------------------------------
{"version":"1.0","explanations":{"kernel_shap":{"label0":{"global_shap_values":
{"x0":0.05386043365892927,"x1":0.10801289723274592,"x2":0.23227865827017378,"x3":0.0676685133394
(1 row)
```

EXPLAIN_MODEL 関数は SUPER データ型を返すため、説明可能性レポートをクエリできます。これにより、`global_shap_values`、`expected_value`を抽出できるほか、特徴固有の Shapley 値を指定することができます。

次の例では、このモデル`global_shap_values`を抽出します。

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
                                                        global_shap_values
-------------------------------------------------------------------------------------------
{"state":0.10983770427197151,"account_length":0.1772441398408543,"area_code":0.08626823968639591
(1 row)
```

次の例では、特徴x0global_shap_valuesを抽出します。

```
select json_table.report.explanations.kernel_shap.label0.global_shap_values.x0 from
 (select explain_model('customer_churn_auto_model') as report) as json_table;
          x0
------------------------
  0.05386043365892927
(1 row)
```

モデルが特定のスキーマで作成され、作成したモデルにアクセスできる場合は、次に示すように、そのモデルの説明をクエリできます。

```
-- Check the current schema
SHOW search_path;
   search_path
------------------
  $user, public
(1 row)
-- If you have the privilege to access the model explanation
-- in `test_schema`
SELECT explain_model('test_schema.test_model_name');
                       explain_model
---------------------------------------------------------
{"explanations":"waiting for training job to complete"}
(1 row)
```

数学関数

トピック

- 数学演算子の記号
- ABS 関数
- ACOS 関数
- ASIN 関数

このセクションでは、Amazon Redshift でサポートされる数学演算子および数学関数について説明します。

数学演算子の記号

次の表に、サポートされる数学演算子の一覧を示します。

サポートされている演算子

演算子	説明	例	結果
+	加算	2 + 3	5
-	減算	2 - 3	-1
*	乗算	2 * 3	6
/	除算	4 / 2	2
%	モジュロ	5 % 4	1
^	べき算	2.0 ^ 3.0	8
|/	平方根	| / 25.0	5
||/	立方根	|| / 27.0	3
@	絶対値	@ -5.0	5
<<	ビット単位で左にシフト	1 << 4	16
>>	ビット単位で右にシフト	8 >> 2	2
&	ビット単位 AND	8 & 2	0

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

特定の取引において支払われたコミッションに 2.00 USD を加算するには、次の例を使用します。

```
SELECT
```

```
    commission,
    (commission + 2.00) AS comm
FROM
    sales
WHERE
    salesid = 10000;

+------------+-------+
| commission | comm  |
+------------+-------+
|      28.05 | 30.05 |
+------------+-------+
```

特定の取引において販売価格の 20% を計算するには、次の例を使用します。

```
SELECT pricepaid, (pricepaid * .20) as twentypct
FROM sales
WHERE salesid=10000;

+-----------+-----------+
| pricepaid | twentypct |
+-----------+-----------+
|       187 |      37.4 |
+-----------+-----------+
```

継続的な成長パターンに基づいてチケット販売を予測するには、次の例を使用します。次の例では、サブクエリによって、2008 年に販売されたチケット数が返されます。その結果に、10 年にわたって継続する成長率 5% が指数関数的に乗算されます。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid AND year=2008)^((5::float/100)*10) AS qty10years;

+------------------+
|    qty10years    |
+------------------+
| 587.664019657491 |
+------------------+
```

日付 ID が 2000 以上である販売の合計支払額および合計コミッションを求めるには、次の例を使用します。その後、合計支払額から合計コミッションを減算します。

```
SELECT SUM(pricepaid) AS sum_price, dateid,
```

```
SUM(commission) AS sum_comm, (SUM(pricepaid) - SUM(commission)) AS value
FROM sales
WHERE dateid >= 2000
GROUP BY dateid
ORDER BY dateid
LIMIT 10;

+-----------+--------+----------+-----------+
| sum_price | dateid | sum_comm |   value   |
+-----------+--------+----------+-----------+
|    305885 |   2000 | 45882.75 | 260002.25 |
|    316037 |   2001 | 47405.55 | 268631.45 |
|    358571 |   2002 | 53785.65 | 304785.35 |
|    366033 |   2003 | 54904.95 | 311128.05 |
|    307592 |   2004 |  46138.8 |  261453.2 |
|    333484 |   2005 |  50022.6 |  283461.4 |
|    317670 |   2006 |  47650.5 |  270019.5 |
|    351031 |   2007 | 52654.65 | 298376.35 |
|    313359 |   2008 | 47003.85 | 266355.15 |
|    323675 |   2009 | 48551.25 | 275123.75 |
+-----------+--------+----------+-----------+
```

ABS 関数

ABS は、数値の絶対値を計算します。この数値は、リテラルでも、数値に評価される式でも構いません。

構文

```
ABS(number)
```

引数

number

数値、または数値に評価される式。SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8、または SUPER 型を使用できます。

戻り型

ABS は、その引数と同じデータ型を返します。

例

-38 の絶対値を計算するには、次の例を使用します。

```
SELECT ABS(-38);

+-----+
| abs |
+-----+
|  38 |
+-----+
```

(14-76) の絶対値を計算するには、次の例を使用します。

```
SELECT ABS(14-76);

+-----+
| abs |
+-----+
|  62 |
+-----+
```

ACOS 関数

ACOS は、数値のアークコサイン (逆余弦) を返す三角関数です。戻り値はラジアンで、0～PI の範囲内です。

構文

```
ACOS(number)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

DOUBLE PRECISION

例

-1 のアークコサイン (逆余弦) を返すには、次の例を使用します。

```
SELECT ACOS(-1);

+-------------------+
|       acos        |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

.5 のアークコサイン (逆余弦) を、その値に相当する度数に変換するには、次の例を使用します。

```
SELECT (ACOS(.5) * 180/(SELECT PI())) AS degrees;

+-------------------+
|      degrees      |
+-------------------+
| 60.00000000000001 |
+-------------------+
```

ASIN 関数

ASIN は、数値のアークサイン (逆正弦) を返す三角関数です。戻り値はラジアンで、PI/2～-PI/2 の範囲内です。

構文

```
ASIN(number)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

DOUBLE PRECISION

例

1 のアークサイン (正弦) を返すには、次の例を使用します。

```
SELECT ASIN(1) AS halfpi;

+--------------------+
|       halfpi       |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

.5 のアークサイン (正弦) を、その値に相当する度数に変換するには、次の例を使用します。

```
SELECT (ASIN(.5) * 180/(SELECT PI())) AS degrees;

+--------------------+
|      degrees       |
+--------------------+
| 30.000000000000004 |
+--------------------+
```

ATAN 関数

ATAN は、数値のアークタンジェント (逆正接) を返す三角関数です。戻り値はラジアンで、-PI～PI の範囲内です。

構文

```
ATAN(number)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

DOUBLE PRECISION

例

1 のアークタンジェント (逆正接) を返し、その値に 4 を乗算するには、次の例を使用します。

```
SELECT ATAN(1) * 4 AS pi;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

1 のアークタンジェント (逆正弦) を、その値に相当する度数に変換するには、次の例を使用します。

```
SELECT (ATAN(1) * 180/(SELECT PI())) AS degrees;

+---------+
| degrees |
+---------+
|      45 |
+---------+
```

ATAN2 関数

ATAN2 は、一方の数値をもう一方の数値で除算した値のアークタンジェント (逆正接) を返す三角関数です。戻り値はラジアンで、`PI/2`～`-PI/2` の範囲内です。

構文

```
ATAN2(number1, number2)
```

引数

number1

`DOUBLE PRECISION` 数。

number2

`DOUBLE PRECISION` 数。

戻り型

DOUBLE PRECISION

例

2/2 のアークタンジェント (逆正接) を返し、その値に 4 を乗算するには、次の例を使用します。

```
SELECT ATAN2(2,2) * 4 AS PI;

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

1/0 のアークタンジェント (逆正接) (0 に評価される) を、その値に相当する度数に変換するには、次の例を使用します。

```
SELECT (ATAN2(1,0) * 180/(SELECT PI())) AS degrees;

+---------+
| degrees |
+---------+
|      90 |
+---------+
```

CBRT 関数

CBRT 関数は、指定された数値の立方根を計算する数学関数です。

構文

```
CBRT(number)
```

引数

CBRT は DOUBLE PRECISION 数を引数として取ります。

戻り型

DOUBLE PRECISION

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

特定の取引において支払われたコミッションの立方根を計算するには、次の例を使用します。

```
SELECT CBRT(commission) FROM sales WHERE salesid=10000;

+--------------------+
|        cbrt        |
+--------------------+
| 3.0383953904884344 |
+--------------------+
```

CEILING (または CEIL) 関数

CEILING 関数または CEIL 関数は、数値を最も近い整数に切り上げるために使用します。(FLOOR 関数は、数値を最も近い整数に切り下げます。)

構文

```
{CEIL | CEILING}(number)
```

引数

number

数値、または数値に評価される式。SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8、または SUPER 型を使用できます。

戻り型

CEILING および CEIL は、引数と同じデータ型を返します。

入力が SUPER 型の場合、出力は入力と同じ動的型を保持しますが、静的型は SUPER 型のままです。SUPER の動的型が数値でない場合、Amazon Redshift は null を返します。

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

特定の取引において支払われたコミッションの天井値を計算するには、次の例を使用します。

```
SELECT CEILING(commission) FROM sales
WHERE salesid=10000;

+---------+
| ceiling |
+---------+
|      29 |
+---------+
```

COS 関数

COS は、数値のコサイン (余弦) を返す三角関数です。戻り値はラジアンで、-1 以上 1 以下です。

構文

```
COS(double_precision)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

COS 関数は DOUBLE PRECISION 数です。

例

0 のコサイン (余弦) を返すには、次の例を使用します。

```
SELECT COS(0);

+-----+
| cos |
```

```
+-----+
|   1 |
+-----+
```

pi のコサイン (余弦) を返すには、次の例を使用します。

```
SELECT COS(PI());

+-----+
| cos |
+-----+
|  -1 |
+-----+
```

COT 関数

COT は、数値のコタンジェント (余接) を返す三角関数です。入力パラメータは 0 以外である必要があります。

構文

```
COT(number)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

DOUBLE PRECISION

例

1 のコタンジェント (余接) を返すには、次の例を使用します。

```
SELECT COT(1);

+-------------------+
|        cot        |
```

```
+--------------------+
| 0.6420926159343306 |
+--------------------+
```

DEGREES 関数

ラジアンで指定された角度を、その値に相当する度数に変換します。

構文

```
DEGREES(number)
```

引数

number

入力パラメータは `DOUBLE PRECISION` 数です。

戻り型

`DOUBLE PRECISION`

例

0.5 ラジアンに相当する度数を返すには、次の例を使用します。

```
SELECT DEGREES(.5);

+-------------------+
|      degrees      |
+-------------------+
| 28.64788975654116 |
+-------------------+
```

PI ラジアンを度数に変換するには、次の例を使用します。

```
SELECT DEGREES(pi());

+---------+
| degrees |
+---------+
```

```
|     180 |
+---------+
```

DEXP 関数

DEXP 関数は、倍精度の数値を指数値で返します。DEXP 関数と EXP 関数の唯一の違いは、DEXP のパラメータは `DOUBLE PRECISION` である必要がある点です。

構文

```
DEXP(number)
```

引数

number

入力パラメータは `DOUBLE PRECISION` 数です。

戻り型

`DOUBLE PRECISION`

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

DEXP 関数は、継続的な成長パターンに基づいてチケット販売を予測するために使用します。次の例では、サブクエリによって、2008 年に販売されたチケット数が返されます。その結果に、10 年にわたって継続する成長率 7% を指定する DEXP 関数の結果が乗算されます。

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * DEXP((7::FLOAT/100)*10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 695447.4837722216 |
+-------------------+
```

DLOG1 関数

DLOG1 関数は、入力パラメータの自然対数を返します。LN 関数 のシノニム。

DLOG10 関数

DLOG10 は、入力パラメータの 10 を底とする対数を返します。

LOG 関数 のシノニム。

構文

```
DLOG10(number)
```

引数

number

入力パラメータは DOUBLE PRECISION 数です。

戻り型

DOUBLE PRECISION

例

数値 100 の 10 を底とする対数を返すには、次の例を使用します。

```
SELECT DLOG10(100);

+--------+
| dlog10 |
+--------+
|      2 |
+--------+
```

EXP 関数

EXP 関数では、数値式、または式の累乗で累乗された自然対数 e の基数に対して指数関数を実装します。EXP 関数は、LN 関数の逆です。

構文

```
EXP(expression)
```

引数

expression

式は、INTEGER、DECIMAL、または DOUBLE PRECISION データ型である必要があります。

戻り型

DOUBLE PRECISION

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

EXP 関数は、継続的な成長パターンに基づいてチケット販売を予測するために使用します。次の例では、サブクエリによって、2008 年に販売されたチケット数が返されます。その結果に、10 年にわたって継続する成長率 7% を指定する EXP 関数の結果が乗算されます。

```
SELECT (SELECT SUM(qtysold)
FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * EXP((7::FLOAT/100)*10) qty2018;

+--------------------+
|      qty2018       |
+--------------------+
| 695447.4837722216 |
+--------------------+
```

FLOOR 関数

FLOOR 関数は、数値を次の整数に切り捨てます。

構文

```
FLOOR(number)
```

引数

number

数値、または数値に評価される式。SMALLINT、INTEGER、BIGINT、DECIMAL、FLOAT4、FLOAT8、または SUPER 型を使用できます。

戻り型

FLOOR は、その引数と同じデータ型を返します。

入力が SUPER 型の場合、出力は入力と同じ動的型を保持しますが、静的型は SUPER 型のままです。SUPER の動的型が数値でない場合、Amazon Redshift は NULL を返します。

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

FLOOR 関数を使用する前と後に特定の販売取引に対して支払われたコミッションの値を示すには、次の例を使用します。

```
SELECT commission
FROM sales
WHERE salesid=10000;

+------------+
| commission |
+------------+
|      28.05 |
+------------+

SELECT FLOOR(commission)
FROM sales
WHERE salesid=10000;

+-------+
| floor |
+-------+
|    28 |
+-------+
```

LN 関数

入力パラメータの自然対数を返します。

DLOG1 関数 のシノニム。

構文

```
LN(expression)
```

引数

expression

関数の対象となる列または式。

> **Note**
>
> この関数は、式の参照先が Amazon Redshift のユーザー作成テーブルである場合、あるいは Amazon Redshift の STL または STV システムテーブルである場合に、データ型のエラーを返します。

式の参照先がユーザー作成テーブルまたはシステムテーブルである場合、式のデータ型が以下のいずれかであるときに、エラーが発生します。以下のデータ型の式は、リーダーノードで排他的に実行されます。

- `BOOLEAN`
- `CHAR`
- `DATE`
- `DECIMAL` または `NUMERIC`
- `TIMESTAMP`
- `VARCHAR`

以下のデータ型の式は、ユーザー作成テーブルおよび STL または STV システムテーブルで、正常に実行されます。

- `BIGINT`
- `DOUBLE PRECISION`
- `INTEGER`

- REAL
- SMALLINT

戻り型

LN 関数は、入力式と同じ型を返します。

例

数値 2.718281828 の自然対数、または e を底とする対数を返すには、次の例を使用します。

```
SELECT LN(2.718281828);

+--------------------+
|         ln         |
+--------------------+
| 0.9999999998311267 |
+--------------------+
```

解は 1 の近似値になることに注意してください。

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

USERS テーブル内の userid 列の値の自然対数を返すには、次の例を使用します。

```
SELECT username, LN(userid) FROM users ORDER BY userid LIMIT 10;

+----------+--------------------+
| username |         ln         |
+----------+--------------------+
| JSG99FHE |                  0 |
| PGL08LJI | 0.6931471805599453 |
| IFT66TXU | 1.0986122886681098 |
| XDZ38RDD | 1.3862943611198906 |
| AEB55QTM | 1.6094379124341003 |
| NDQ15VBM |  1.791759469228055 |
| OWY35QYB | 1.9459101490553132 |
| AZG78YIP | 2.0794415416798357 |
| MSD36KVR | 2.1972245773362196 |
| WKW41AIW |  2.302585092994046 |
+----------+--------------------+
```

LOG 関数

数値の対数を返します。

この関数を使用して 10 を底とする対数を計算する場合は、DLOG10 関数 も使用できます。

構文

```
LOG([base, ]argument)
```

パラメータ

base

(オプション) 対数関数の基数。この数値は正でなければならず、1 と等しくすることはできません。このパラメータを省略すると、Amazon Redshift は引数の 10 を底とする対数を計算します。

argument

対数関数の引数。この数字は正でなければなりません。引数が 1 値である場合、この関数は `0` を返します。

戻り型

LOG 関数は `DOUBLE PRECISION` 数を返します。

例

100 の 2 を底とする対数を求めるには、次の例を使用します。

```
SELECT LOG(2, 100);
+--------------------+
|        log         |
+--------------------+
| 6.643856189774725  |
+--------------------+
```

100 の 10 を底とする対数を求めるには、次の例を使用します。ベースパラメータを省略した場合、Amazon Redshift は底が 10 であると仮定することに注意してください。

```
SELECT LOG(100);
```

```
+-----+
| log |
+-----+
|   2 |
+-----+
```

MOD 関数

2 つの数値の余りを返します。モジュロ演算とも呼ばれます。結果を計算するには、最初のパラメータを 2 番目のパラメータで除算します。

構文

```
MOD(number1, number2)
```

引数

number1

最初の入力パラメータは、INTEGER、SMALLINT、BIGINT、または DECIMAL 数です。一方のパラメータが DECIMAL 型である場合は、もう一方のパラメータも DECIMAL 型である必要があります。一方のパラメータが INTEGER である場合、もう一方のパラメータは INTEGER、SMALLINT、または BIGINT のいずれかにします。両方のパラメータを SMALLINT または BIGINT にすることもできますが、一方のパラメータが BIGINT である場合に、もう一方のパラメータを SMALLINT にすることはできません。

number2

2 番目のパラメータは、INTEGER、SMALLINT、BIGINT、または DECIMAL 数です。number1 と同じデータ型ルールが number2 に適用されます。

戻り型

両方の入力パラメータが同じ型である場合、MOD 関数の戻り型は、入力パラメータと同じ数値型になります。ただし、一方の入力パラメータが INTEGER である場合は、戻り型も INTEGER になります。有効な戻り値の型は、DECIMAL、INT、SMALLINT、および BIGINT です。

使用に関する注意事項

% をモジュロ演算子として使用できます。

例

ある数値を別の数値で除算したときの余りを返すには、次の例を使用します。

```
SELECT MOD(10, 4);

+------+
| mod  |
+------+
|    2 |
+------+
```

MOD 関数を使用するときに DECIMAL の結果を返すには、次の例を使用します。

```
SELECT MOD(10.5, 4);

+------+
| mod  |
+------+
| 2.5  |
+------+
```

MOD 関数を実行する前に数値をキャストするには、次の例を使用します。詳細については、「CAST 関数」を参照してください。

```
SELECT MOD(CAST(16.4 AS INTEGER), 5);

+------+
| mod  |
+------+
|    1 |
+------+
```

最初のパラメータを 2 で割って偶数かどうかをチェックするには、次の例を使用します。

```
SELECT mod(5,2) = 0 AS is_even;

+---------+
| is_even |
+---------+
| false   |
+---------+
```

モジュロ演算子として % を使用するには、次の例を使用します。

```
SELECT 11 % 4 as remainder;

 +-----------+
| remainder |
+-----------+
|         3 |
+-----------+
```

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATEGORY テーブル内の奇数カテゴリの情報を返すには、次の例を使用します。

```
SELECT catid, catname
FROM category
WHERE MOD(catid,2)=1
ORDER BY 1,2;

+-------+-----------+
| catid |  catname  |
+-------+-----------+
|     1 | MLB       |
|     3 | NFL       |
|     5 | MLS       |
|     7 | Plays     |
|     9 | Pop       |
|    11 | Classical |
+-------+-----------+
```

PI 関数

pi 関数は、PI の値を小数第 14 位まで返します。

構文

```
PI()
```

戻り型

`DOUBLE PRECISION`

例

pi の値を返すには、次の例を使用します。

```
SELECT PI();

+-------------------+
|        pi         |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

POWER 関数

POWER 関数は指数関数であり、最初の数値式がべき乗の底、2 番目の数値式がべき乗の指数です。例えば、2 の 3 乗は POWER(2,3) と表され、結果は 8 になります。

構文

```
{POW | POWER}(expression1, expression2)
```

引数

expression1

べき乗の底とする数値式。INTEGER、DECIMAL、または FLOAT データ型である必要があります。

expression2

expression1 を底とするべき乗の指数。INTEGER、DECIMAL、または FLOAT データ型である必要があります。

戻り型

DOUBLE PRECISION

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

次の例では、2008 年に販売されたチケット数 (サブクエリの結果) に基づいて今後 10 年間のチケット販売の状況を予測するために、POWER 関数が使用されています。この例の成長率は、1 年当たり 7% に設定されています。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * POW((1+7::FLOAT/100),10) qty2010;

+-------------------+
|      qty2010      |
+-------------------+
| 679353.7540885945 |
+-------------------+
```

次の例は、成長率が 1 年あたり 7% である前の例を基に、間隔を月単位で (120 か月間、つまり 10 年間に) 設定したものです。

```
SELECT (SELECT SUM(qtysold) FROM sales, date
WHERE sales.dateid=date.dateid
AND year=2008) * POW((1+7::FLOAT/100/12),120) qty2010;

+-----------------+
|     qty2010     |
+-----------------+
| 694034.54678046 |
+-----------------+
```

RADIANS 関数

RADIANS 関数は、角度 (度数) をそれに相当するラジアンに変換します。

構文

```
RADIANS(number)
```

引数

number

入力パラメータは `DOUBLE PRECISION` 数です。

戻り型

DOUBLE PRECISION

例

180 度に相当するラジアンを返すには、次の例を使用します。

```
SELECT RADIANS(180);

+-------------------+
|      radians      |
+-------------------+
| 3.141592653589793 |
+-------------------+
```

RANDOM 関数

RANDOM 関数は、0.0 (この値を含む) ~ 1.0 (この値は含まない) のランダム値を生成します。

構文

```
RANDOM()
```

戻り型

DOUBLE PRECISION

使用に関する注意事項

SET コマンドでシード値を設定した後、RANDOM を呼び出すと、RANDOM が予測可能な順序で数値を生成します。

例

0 ~ 99 のランダム値を計算するには、次の例を使用します。ランダムな数値が 0 ~ 1 である場合、このクエリは、0 ~ 100 のランダムな数値を生成します。

```
SELECT CAST(RANDOM() * 100 AS INT);

+------+
| int4 |
+------+
|   59 |
```

```
+------+
```

次の例では、SETコマンドを使用して SEED 値を設定します。これにより RANDOM が、予測可能な順序で数値を生成します。

SEED 値を設定せずに 3 つの RANDOM 整数を返すには、次の例を使用します。

```
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|    6 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   68 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   56 |
+------+
```

SEED 値を .25 に設定し、さらに 3 つの RANDOM 数を返すには、次の例を使用します。

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   21 |
+------+

SELECT CAST(RANDOM() * 100 AS INT);
+------+
| int4 |
+------+
|   79 |
```

```
+-------+

SELECT CAST(RANDOM() * 100 AS INT);
+-------+
| int4 |
+-------+
|   12 |
+-------+
```

SEED 値を .25 にリセットして、RANDOM が前の 3 つの呼び出しと同じ結果を返すことを検証するには、次の例を使用します。

```
SET SEED TO .25;
SELECT CAST(RANDOM() * 100 AS INT);
+-------+
| int4 |
+-------+
|   21 |
+-------+

SELECT CAST(RANDOM() * 100 AS INT);
+-------+
| int4 |
+-------+
|   79 |
+-------+

SELECT CAST(RANDOM() * 100 AS INT);
+-------+
| int4 |
+-------+
|   12 |
+-------+
```

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブルから、10 個の項目の一様なランダムサンプルを取得するには、次の例を使用します。

```
SELECT *
FROM sales
```

```
ORDER BY RANDOM()
LIMIT 10;

+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
|   45422 |  51114 |     5983 |   24482 |    4369 |   2118 |       1 |       195 |
 29.25 | 2008-10-19 05:20:07 |
|   42481 |  47638 |     4573 |    6198 |    6479 |   1987 |       4 |      1140 |
   171 | 2008-06-10 09:39:19 |
|   31494 |  34759 |    18895 |    4719 |    7753 |   2090 |       4 |      1024 |
 153.6 | 2008-09-21 03:44:26 |
|  119388 | 136685 |    21815 |   41905 |    2071 |   1884 |       1 |       359 |
 53.85 | 2008-02-27 10:43:10 |
|  166990 | 225037 |    18529 |    7628 |     746 |   2113 |       1 |      2009 |
 301.35 | 2008-10-14 10:07:44 |
|   11146 |  12096 |    42685 |    6619 |    1876 |   2123 |       1 |        29 |
  4.35 | 2008-10-24 06:23:54 |
|  148537 | 172056 |    15102 |   11787 |    6122 |   1923 |       2 |       480 |
    72 | 2008-04-07 03:58:23 |
|   68945 |  78387 |     7359 |   18323 |    6636 |   1910 |       1 |       457 |
 68.55 | 2008-03-25 08:31:03 |
|   52796 |  59576 |     9909 |   15102 |    7958 |   1951 |       1 |       479 |
 71.85 | 2008-05-05 02:25:08 |
|   90684 | 103522 |    38052 |   21549 |    7384 |   2117 |       1 |       313 |
 46.95 | 2008-10-18 05:43:11 |
+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

10 個のアイテムのランダムサンプルを取得するが、料金に比例してアイテムを選択するには、次の例を使用します。例えば、別の料金の 2 倍のアイテムは、クエリ結果に表示される可能性が 2 倍になります。

```
SELECT *
FROM sales
ORDER BY -LOG(RANDOM()) / pricepaid
LIMIT 10;

+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

```
| salesid | listid | sellerid | buyerid | eventid | dateid | qtysold | pricepaid |
 commission |      saletime       |
+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
|  158340 | 208208 |    17082 |   42018 |    1211 |   2160 |       4 |      6852 |
 1027.8 | 2008-11-30 12:21:43 |
|   53250 |  60069 |    12644 |    7066 |    7942 |   1838 |       4 |      1528 |
 229.2 | 2008-01-12 11:24:56 |
|   22929 |  24938 |    47314 |    6503 |     179 |   2000 |       3 |       741 |
 111.15 | 2008-06-23 08:04:50 |
|  164980 | 221181 |     1949 |   19670 |    1471 |   1906 |       1 |      1330 |
 199.5 | 2008-03-21 07:59:51 |
|  159641 | 211179 |    44897 |   16652 |    7458 |   2128 |       1 |      1019 |
 152.85 | 2008-10-29 02:02:15 |
|   73143 |  83439 |     5716 |    5727 |    7314 |   1903 |       1 |       248 |
  37.2 | 2008-03-18 11:07:42 |
|   84778 |  96749 |    46608 |   32980 |    3883 |   1999 |       2 |       958 |
 143.7 | 2008-06-22 12:13:31 |
|  171096 | 232929 |    43683 |    8536 |    8353 |   1870 |       1 |       929 |
 139.35 | 2008-02-13 01:36:36 |
|   74212 |  84697 |    39809 |   15569 |    5525 |   2105 |       2 |       896 |
 134.4 | 2008-10-06 11:47:50 |
|  158011 | 207556 |    25399 |   16881 |     232 |   2088 |       2 |      2526 |
 378.9 | 2008-09-19 06:00:26 |
+---------+--------+---------+---------+---------+--------+---------+-----------
+------------+---------------------+
```

ROUND 関数

ROUND 関数は、数値を四捨五入して、最も近い整数または 10 進数にします。

ROUND 関数にはオプションで、2 番目の引数として `INTEGER` を指定できます。この整数は、四捨五入後の小数点以下または小数点以上の桁数を指定します。2 番目の引数を指定しない場合、関数は最も近い整数に四捨五入されます。2 番目の引数 integer が指定されている場合、この関数は小数点以下 integer 桁の精度で最も近い数値に四捨五入されます。

構文

```
ROUND(number [ , integer ] )
```

引数

number

数値、または数値に評価される式。DECIMAL、FLOAT8、または SUPER 型を使用できます。Amazon Redshift は、他の数値データ型を暗黙的に変換できます。

integer

(オプション) 小数点以上または小数点以下の桁数を示す INTEGER。SUPER データ型は、この引数ではサポートされていません。

戻り型

ROUND は、入力と同じ number データ型を返します。

入力が SUPER 型の場合、出力は入力と同じ動的型を保持しますが、静的型は SUPER 型のままです。SUPER の動的型が数値でない場合、Amazon Redshift は NULL を返します。

例

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

特定の取引において支払われたコミッションを四捨五入して、最も近い整数にするには、次の例を使用します。

```
SELECT commission, ROUND(commission)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    28 |
+------------+-------+
```

特定の取引において支払われたコミッションを四捨五入して、小数点以下第 1 位までの数値にするには、次の例を使用します。

```
SELECT commission, ROUND(commission, 1)
FROM sales WHERE salesid=10000;
```

```
+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |  28.1 |
+------------+-------+
```

精度を前の例と逆方向に拡張するには、次の例を使用します。

```
SELECT commission, ROUND(commission, -1)
FROM sales WHERE salesid=10000;

+------------+-------+
| commission | round |
+------------+-------+
|      28.05 |    30 |
+------------+-------+
```

SIN 関数

SIN は、数値のサイン (正弦) を返す三角関数です。戻り値は、-1～1 です。

構文

```
SIN(number)
```

引数

number

ラジアン単位の DOUBLE PRECISION 数。

戻り型

DOUBLE PRECISION

例

-PI のサイン (正弦) を返すには、次の例を使用します。

```
SELECT SIN(-PI());
```

```
+-------------------------+
|           sin           |
+-------------------------+
| -0.00000000000000012246 |
+-------------------------+
```

SIGN 関数

SIGN 関数は、数の符号 (正または負) を返します。SIGN 関数の結果は、引数が正の場合は 1、引数が負の場合は -1、引数が 0 の場合は 0 になります。

構文

```
SIGN(number)
```

引数

number

数値、または数値に評価される式。DECIMAL、FLOAT8、または SUPER 型を使用できます。その他のデータ型は、暗黙的変換ルールに従って Amazon Redshift によって変換できます。

戻り型

SIGN は、入力引数と同じ数値データ型を返します。入力が DECIMAL の場合、出力は DECIMAL(1,0) になります。

入力が SUPER 型の場合、出力は入力と同じ動的型を保持しますが、静的型は SUPER 型のままです。SUPER の動的型が数値でない場合、Amazon Redshift は NULL を返します。

例

次の例は、入力が DOUBLE PRECISION であるため、テーブル t2 の列 d の型が DOUBLE PRECISION であり、入力が NUMERIC であるため、テーブル t2 の列 n の出力は NUMERIC(1,0) であることを示しています。

```
CREATE TABLE t1(d DOUBLE PRECISION, n NUMERIC(12, 2));
INSERT INTO t1 VALUES (4.25, 4.25), (-4.25, -4.25);
CREATE TABLE t2 AS SELECT SIGN(d) AS d, SIGN(n) AS n FROM t1;
SELECT table_name, column_name, data_type FROM SVV_REDSHIFT_COLUMNS WHERE
 table_name='t1' OR table_name='t2';
```

```
+------------+-------------+-----------------------+
| table_name | column_name |       data_type       |
+------------+-------------+-----------------------+
| t1         | d           | double precision      |
| t1         | n           | numeric(12,2)         |
| t2         | d           | double precision      |
| t2         | n           | numeric(1,0)          |
| t1         | col1        | character varying(20) |
+------------+-------------+-----------------------+
```

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブルから、特定の取引において支払われたコミッションの符号を判別するには、次の例を使用します。

```
SELECT commission, SIGN(commission)
FROM sales WHERE salesid=10000;

+------------+------+
| commission | sign |
+------------+------+
|      28.05 |    1 |
+------------+------+
```

SQRT 関数

SQRT 関数は、NUMERIC 値の平方根を返します。平方根は、与えられた値を得るためにそれ自体を掛けた数値です。

構文

```
SQRT(expression)
```

引数

expression

式には INTEGER、DECIMAL、または FLOAT データ型、あるいはそれらのデータ型に暗黙的に変換されるデータ型が必要です。式 には関数を含めることができます。

戻り型

DOUBLE PRECISION

例

16 の平方根を返すには、次の例を使用します。

```
SELECT SQRT(16);

+------+
| sqrt |
+------+
|    4 |
+------+
```

暗黙的な型変換を使用して文字列 16 の平方根を返すには、次の例を使用します。

```
SELECT SQRT('16');

+------+
| sqrt |
+------+
|    4 |
+------+
```

ROUND 関数を使用した後に 16.4 の平方根を返すには、次の例を使用します。

```
SELECT SQRT(ROUND(16.4));

+------+
| sqrt |
+------+
|    4 |
+------+
```

円のエリアを指定したときの半径の長さを返すには、次の例を使用します。例えば、エリアを平方インチで指定すると、半径をインチで計算します。サンプルのエリアは 20 です。

```
SELECT SQRT(20/PI()) AS radius;

+--------------------+
```

```
|       radius        |
+---------------------+
| 2.5231325220201604 |
+---------------------+
```

次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブルから COMMISSION 値の平方根を返すには、次の例を使用します。COMMISSION 列は DECIMAL 型列です。この例は、より複雑な条件ロジックを含むクエリで関数を使用する方法を示しています。

```
SELECT SQRT(commission)
FROM sales WHERE salesid < 10 ORDER BY salesid;

+---------------------+
|        sqrt         |
+---------------------+
| 10.449880382090505 |
| 3.3763886032268267 |
|  7.245688373094719 |
|  5.123475382979799 |
|  4.806245936279167 |
|  7.687652437513028 |
| 10.871982339941507 |
| 5.4359911699707535 |
|   9.41541289588513 |
+---------------------+
```

上記と同じ COMMISSION 値の四捨五入された平方根を返すには、次の例を使用します。

```
SELECT ROUND(SQRT(commission))
FROM sales WHERE salesid < 10 ORDER BY salesid;

+-------+
| round |
+-------+
|    10 |
|     3 |
|     7 |
|     5 |
|     5 |
```

```
|      8 |
|     11 |
|      5 |
|      9 |
+-------+
```

TAN 関数

TAN は、数値のタンジェントを返す三角関数です。入力引数は数値 (ラジアン単位) です。

構文

```
TAN(number)
```

引数

number

DOUBLE PRECISION 数。

戻り型

DOUBLE PRECISION

例

1 のタンジェント (正接) を返すには、次の例を使用します。

```
SELECT TAN(0);

+-----+
| tan |
+-----+
|   0 |
+-----+
```

TRUNC 関数

TRUNC 関数は、数値を前の整数または小数に切り捨てます。

TRUNC 関数にはオプションで、2 番目の引数として INTEGER を指定できます。この整数は、四捨五入後の小数点以下または小数点以上の桁数を指定します。2 番目の引数を指定しない場合、関数は

最も近い整数に四捨五入されます。2 番目の引数 integer が指定されている場合、この関数は小数点以下 integer 桁の精度で最も近い数値に四捨五入されます。

この関数は TIMESTAMP を切り捨て、DATE を返すこともできます。詳細については、「TRUNC 関数」を参照してください。

構文

```
TRUNC(number [ , integer ])
```

引数

number

数値、または数値に評価される式。DECIMAL、FLOAT8、または SUPER 型を使用できます。その他のデータ型は、暗黙的変換ルールに従って Amazon Redshift によって変換できます。

integer

(オプション) 小数点以上または小数点以下の精度の桁数を示す INTEGER。この integer が指定されなかった場合は、数値が切り捨てられて整数になります。この integer が指定された場合は、数値が切り捨てられて、指定された桁数になります。これは SUPER データ型ではサポートされていません。

戻り型

TRUNC は、最初の入力 number と同じデータ型を返します。

入力が SUPER 型の場合、出力は入力と同じ動的型を保持しますが、静的型は SUPER 型のままです。SUPER の動的型が数値でない場合、Amazon Redshift は NULL を返します。

例

次の一部の例では、TICKIT サンプルデータベースを使用しています。詳細については、「サンプルデータベース」を参照してください。

特定の取引において支払われたコミッションを切り捨てるには、次の例を使用します。

```
SELECT commission, TRUNC(commission)
FROM sales WHERE salesid=784;

+------------+-------+
```

```
| commission | trunc |
+------------+-------+
|     111.15 |   111 |
+------------+-------+
```

同じコミッション値を切り捨てて、小数点以下第 1 位までの数値にするには、次の例を使用します。

```
SELECT commission, TRUNC(commission,1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 | 111.1 |
+------------+-------+
```

2 番目の引数に対して負の値でコミッションを切り捨てるには、次の例を使用します。`111.15` は `110` に切り下げられることに注意してください。

```
SELECT commission, TRUNC(commission,-1)
FROM sales WHERE salesid=784;

+------------+-------+
| commission | trunc |
+------------+-------+
|     111.15 |   110 |
+------------+-------+
```

オブジェクト関数

次に、SUPER 型オブジェクトを作成するために Amazon Redshift がサポートする SQL オブジェクト関数を示します。

トピック

LOWER_ATTRIBUTE_NAMES 関数

LOWER 関数 と同じ大文字/小文字の変換ルーチンを使用して、SUPER 値のすべての該当する属性名を小文字に変換します。LOWER_ATTRIBUTE_NAMES は、UTF-8 マルチバイト文字をサポートします (1 文字につき最大 4 バイトまで)。

SUPER 属性名を大文字に変換するには、UPPER_ATTRIBUTE_NAMES 関数 を使用します。

構文

```
LOWER_ATTRIBUTE_NAMES(super_expression)
```

引数

super_expression

SUPER 式。

戻り型

SUPER

使用に関する注意事項

Amazon Redshift では、従来、列識別子の大文字と小文字を区別せずに小文字に変換します。JSON などの大文字と小文字を区別するデータ形式からデータを取り込むと、データには大文字と小文字が混在する属性名が含まれることがあります。

次の例を考えます。

```
CREATE TABLE t1 (s) AS SELECT JSON_PARSE('{"AttributeName": "Value"}');


SELECT s.AttributeName FROM t1;

attributename
-------------
NULL


SELECT s."AttributeName" FROM t1;

```

```
attributename
-------------
NULL
```

Amazon Redshift は、両方のクエリに対して NULL を返します。`AttributeName` をクエリするには、LOWER_ATTRIBUTE_NAMES を使用してデータの属性名を小文字に変換します。次の例を考えます。

```
CREATE TABLE t2 (s) AS SELECT LOWER_ATTRIBUTE_NAMES(s) FROM t1;


SELECT s.attributename FROM t2;

attributename
-------------
"Value"


SELECT s.AttributeName FROM t2;

attributename
-------------
"Value"


SELECT s."attributename" FROM t2;

attributename
-------------
"Value"


SELECT s."AttributeName" FROM t2;

attributename
-------------
"Value"
```

大文字と小文字が混在するオブジェクト属性名を使用するための関連オプションは、`enable_case_sensitive_super_attribute` 設定オプションです。これにより、Amazon Redshift は SUPER 属性名の大文字と小文字を認識できます。これは、LOWER_ATTRIBUTE_NAMES の代替ソリューションとして使用で

ます。enable_case_sensitive_super_attribute の詳細については、「enable_case_sensitive_super_attribute」を参照してください。

例

SUPER 属性名を小文字に変換する

次の例では、LOWER_ATTRIBUTE_NAMES を使用して、テーブル内のすべての SUPER 値の属性名を変換します。

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
  (1, NULL),
  (2, 'A'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "B"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"C": "C"},
        "Subarray": [{"D": "D"}, "E"]
      }]'));

-- Convert all attribute names to lowercase.
UPDATE t SET s = LOWER_ATTRIBUTE_NAMES(s);

SELECT i, s FROM t ORDER BY i;

 i |                         s
---+-----------------------------------------------------
 1 | NULL
 2 | "A"
 3 | {"attributename":"B"}
 4 | [{"subobject":{"c":"C"},"subarray":[{"d":"D"}, "E"]}]
```

LOWER_ATTRIBUTE_NAMES がどのように機能するかを確認します。

- NULL 値とスカラー SUPER 値 ("A" など) は変更されません。
- SUPER オブジェクトでは、すべての属性名は小文字に変更されますが、"B" などの属性値は変更されません。
- LOWER_ATTRIBUTE_NAMES は、SUPER 配列内または別のオブジェクト内にネストされているすべての SUPER オブジェクトに再帰的に適用されます。

重複する属性名を持つ SUPER オブジェクトでの LOWER_ATTRIBUTE_NAMES の使用

SUPER オブジェクトに大文字と小文字だけが異なる名前の属性が含まれている場合、LOWER_ATTRIBUTE_NAMES はエラーを発生させます。次の例を考えます。

```
SELECT LOWER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

OBJECT 関数

SUPER データ型のオブジェクトを作成します。

構文

```
OBJECT ( [ key1, value1 ], [ key2, value2 ...] )
```

引数

key1, key2

VARCHAR 型の文字列に評価される式。

value1, value2

Amazon Redshift では、日時の型は SUPER データ型にキャストされないため、すべての Amazon Redshift データ型の式では日時の型は除外されます。日時の型の詳細については、「[日時型](#)」を参照してください。

オブジェクト内の `value` 式は、同じデータ型である必要はありません。

戻り型

`SUPER`

例

```
-- Creates an empty object.
select object();

object
```

```
--------
{}
(1 row)

-- Creates objects with different keys and values.
select object('a', 1, 'b', true, 'c', 3.14);

object
---------------------------
{"a":1,"b":true,"c":3.14}
(1 row)

select object('a', object('aa', 1), 'b', array(2,3), 'c', json_parse('{}'));

object
---------------------------------
{"a":{"aa":1},"b":[2,3],"c":{}}
(1 row)

-- Creates objects using columns from a table.
create table bar (k varchar, v super);
insert into bar values ('k1', json_parse('[1]')), ('k2', json_parse('{}'));
select object(k, v) from bar;

object
------------
{"k1":[1]}
{"k2":{}}
(2 rows)

-- Errors out because DATE type values can't be converted to SUPER type.
select object('k', '2008-12-31'::date);

ERROR:  OBJECT could not convert type date to super
```

OBJECT_TRANSFORM 関数

SUPER オブジェクトを変換します。

構文

```
OBJECT_TRANSFORM(
  input
```

```
  [KEEP path1, ...]
  [SET
    path1, value1,
    ...,  ...
  ]
)
```

引数

input

SUPER 型オブジェクトに解決される式。

KEEP

この句で指定された path 値はすべて保持され、出力オブジェクトに引き継がれます。

この句はオプションです。

path1, path2, ...

二重引用符で囲まれたパスコンポーネントをピリオドで区切った形式の定数文字列リテラル。例えば、`'"a"."b"."c"'` は有効なパス値です。これは KEEP 句と SET 句の両方のパスパラメーターに当てはまります。

SET

path と value のペアを使用して既存のパスを変更するか、新しいパスを追加し、出力オブジェクトにそのパスの値を設定します。

この句はオプションです。

value1, value2, ...

SUPER 型の値に解決される式。数値型、テキスト型、ブール型は SUPER に解決可能であることに注意してください。

戻り型

`SUPER`

使用に関する注意事項

OBJECT_TRANSFORM は、KEEP で指定された input からのパス値と、SET で指定された path と value のペアを含む SUPER 型のオブジェクトを返します。

KEEP と SET の両方が空の場合、OBJECT_TRANSFORM は input を返します。

input が SUPER 型の object でない場合、OBJECT_TRANSFORM は KEEP 値や SET 値に関係なく input を返します。

例

次の例では、SUPER オブジェクトを別の SUPER オブジェクトに変換します。

```
CREATE TABLE employees (
    col_person SUPER
);

INSERT INTO employees
VALUES
    (
        json_parse('
            {
                "name": {
                    "first": "John",
                    "last": "Doe"
                },
                "age": 25,
                "ssn": "111-22-3333",
                "company": "Company Inc.",
                "country": "U.S."
            }
        ')
    ),
    (
        json_parse('
            {
                "name": {
                    "first": "Jane",
                    "last": "Appleseed"
                },
                "age": 34,
                "ssn": "444-55-7777",
                "company": "Organization Org.",
                "country": "Ukraine"
            }
        ')
    )
;
```

```
SELECT
    OBJECT_TRANSFORM(
        col_person
        KEEP
            '"name"."first"',
            '"age"',
            '"company"',
            '"country"'
        SET
            '"name"."first"', UPPER(col_person.name.first::TEXT),
            '"age"', col_person.age + 5,
            '"company"', 'Amazon'
    ) AS col_person_transformed
FROM employees;

--This result is formatted for ease of reading.
                  col_person_transformed
-------------------------------------------------------------
{
    "name": {
        "first": "JOHN"
    },
    "age": 30,
    "company": "Amazon",
    "country": "U.S."
}
{
    "name": {
        "first": "JANE"
    },
    "age": 39,
    "company": "Amazon",
    "country": "Ukraine"
}
```

UPPER_ATTRIBUTE_NAMES 関数

UPPER 関数 と同じ大文字/小文字の変換ルーチンを使用して、SUPER 値のすべての該当する属性名を大文字に変換します。UPPER_ATTRIBUTE_NAMES は、UTF-8 マルチバイト文字をサポートします (1 文字につき最大 4 バイトまで)。

SUPER 属性名を小文字に変換するには、LOWER_ATTRIBUTE_NAMES 関数 を使用します。

構文

```
UPPER_ATTRIBUTE_NAMES(super_expression)
```

引数

super_expression

SUPER 式。

戻り型

SUPER

例

SUPER 属性名を大文字に変換する

次の例では、UPPER_ATTRIBUTE_NAMES を使用して、テーブル内のすべての SUPER 値の属性名を変換します。

```
-- Create a table and insert several SUPER values.
CREATE TABLE t (i INT, s SUPER);

INSERT INTO t VALUES
  (1, NULL),
  (2, 'a'::SUPER),
  (3, JSON_PARSE('{"AttributeName": "b"}')),
  (4, JSON_PARSE(
     '[{"Subobject": {"c": "c"},
        "Subarray": [{"d": "d"}, "e"]
      }]'));

-- Convert all attribute names to uppercase.
UPDATE t SET s = UPPER_ATTRIBUTE_NAMES(s);

SELECT i, s FROM t ORDER BY i;

 i |                        s
---+-----------------------------------------------------
 1 | NULL
 2 | "a"
```

```
 3 | {"ATTRIBUTENAME":"B"}
 4 | [{"SUBOBJECT":{"C":"c"},"SUBARRAY":[{"D":"d"}, "e"]}]
```

UPPER_ATTRIBUTE_NAMES がどのように機能するかを確認します。

- NULL 値とスカラー SUPER 値 ("a" など) は変更されません。
- SUPER オブジェクトでは、すべての属性名が大文字に変更されますが、"b" などの属性値は変更されません。
- UPPER_ATTRIBUTE_NAMES は、SUPER 配列内または別のオブジェクト内にネストされているすべての SUPER オブジェクトに再帰的に適用されます。

重複する属性名を持つ SUPER オブジェクトでの UPPER_ATTRIBUTE_NAMES の使用

SUPER オブジェクトに大文字と小文字だけが異なる名前の属性が含まれている場合、UPPER_ATTRIBUTE_NAMES はエラーを発生させます。次の例を考えます。

```
SELECT UPPER_ATTRIBUTE_NAMES(JSON_PARSE('{"A": "A", "a": "a"}'));

error:   Invalid input
code:    8001
context: SUPER value has duplicate attributes after case conversion.
```

空間関数

ジオメトリオブジェクト間の関係は、DE-9IM (Dimensionally Extended nine-Intersection Model) に基づいています。このモデルは、EQUALS、CONTAINS、COVERS などの述語を定義します。空間的関係の定義についての詳細は、ウィキペディアの DE-9IM を参照してください。

Amazon Redshift で空間データを使用する方法の詳細については、「Amazon Redshift での空間データのクエリ」を参照してください。

Amazon Redshift には、`GEOMETRY` および `GEOGRAPHY` のデータ型で動作する空間関数が用意されています。以下に、`GEOGRAPHY` データ型をサポートする関数を一覧で示します。

- ST_Area
- ST_AsEWKT
- ST_AsGeoJSON
- ST_AsHexEWKB

- ST_AsHexWKB
- ST_AsText
- ST_Distance
- ST_GeogFromText
- ST_GeogFromWKB
- ST_Length
- ST_NPoints
- ST_Perimeter

Amazon Redshift でサポートされているすべての空間関数を以下に示します。

トピック

- ST_Buffer
- ST_Centroid
- ST_Collect
- ST_Contains
- ST_ContainsProperly
- ST_ConvexHull
- ST_CoveredBy
- ST_Covers
- ST_Crosses
- ST_Dimension
- ST_Disjoint
- ST_Distance
- ST_DistanceSphere
- ST_DWithin
- ST_EndPoint
- ST_Envelope
- ST_Equals
- ST_ExteriorRing
- ST_Force2D
- ST_Force3D
- ST_Force3DM
- ST_Force3DZ
- ST_Force4D
- ST_GeoHash
- ST_GeogFromText
- ST_GeogFromWKB
- ST_GeometryN
- ST_GeometryType
- ST_GeomFromEWKB
- ST_GeomFromEWKT

- ST_GeomFromGeoHash
- ST_GeomFromGeoJSON
- ST_GeomFromGeoSquare
- ST_GeomFromText
- ST_GeomFromWKB
- ST_GeoSquare
- ST_InteriorRingN
- ST_Intersects
- ST_Intersection
- ST_IsPolygonCCW
- ST_IsPolygonCW
- ST_IsClosed
- ST_IsCollection
- ST_IsEmpty
- ST_IsRing
- ST_IsSimple
- ST_IsValid
- ST_Length
- ST_LengthSphere
- ST_Length2D
- ST_LineFromMultiPoint
- ST_LineInterpolatePoint
- ST_M
- ST_MakeEnvelope
- ST_MakeLine
- ST_MakePoint
- ST_MakePolygon
- ST_MemSize
- ST_MMax
- ST_MMin

- ST_Multi
- ST_NDims
- ST_NPoints
- ST_NRings
- ST_NumGeometries
- ST_NumInteriorRings
- ST_NumPoints
- ST_Perimeter
- ST_Perimeter2D
- ST_Point
- ST_PointN
- ST_Points
- ST_Polygon
- ST_RemovePoint
- ST_Reverse
- ST_SetPoint
- ST_SetSRID
- ST_Simplify
- ST_SRID
- ST_StartPoint
- ST_Touches
- ST_Transform
- ST_Union
- ST_Within
- ST_X
- ST_XMax
- ST_XMin
- ST_Y
- ST_YMax
- ST_YMin

- ST_Z
- ST_ZMax
- ST_ZMin
- SupportsBBox

AddBBox

AddBBox は、事前に計算された境界ボックスによるエンコーディングをサポートする、入力ジオメトリのコピーを返します。境界ボックスのサポートの詳細については、「境界ボックス」を参照してください。

構文

```
AddBBox(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

geom が null の場合、null が返されます。

例

次の SQL は、境界ボックスによるエンコードをサポートする、入力ポリゴンジオメトリのコピーを返します。

```
SELECT ST_AsText(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

DropBBox

DropBBox は、事前計算された境界ボックスによるエンコーディングをサポートしていない、入力ジオメトリのコピーを返します。境界ボックスのサポートの詳細については、「境界ボックス」を参照してください。

構文

```
DropBBox(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

geom が null の場合、null が返されます。

例

次の SQL は、境界ボックスによるエンコードをサポートしない、入力ポリゴンジオメトリのコピーを返します。

```
SELECT ST_AsText(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
 st_astext
----------
 POLYGON((0 0,1 0,0 1,0 0))
```

GeometryType

GeometryType は、入力ジオメトリのサブタイプを文字列として返します。

構文

```
GeometryType(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

geom のサブタイプを表す VARCHAR。

geom が null の場合、null が返されます。

返される値は次のとおりです。

2D、3DZ、4D ジオメトリの場合に返される文字列値	3DM ジオメトリの場合に返される文字列値	ジオメトリのサブタイプ
POINT	POINTM	geom が POINT サブタイプの場合に返されます
LINESTRING	LINESTRINGM	geom が LINESTRING サブタイプの場合に返されます
POLYGON	POLYGONM	geom が POLYGON サブタイプの場合に返されます
MULTIPOINT	MULTIPOINTM	geom が MULTIPOINT サブタイプの場合に返されます
MULTILINESTRING	MULTILINESTRINGM	geom が MULTILINESTRING サブタイプの場合に返されます
MULTIPOLYGON	MULTIPOLYGONM	geom が MULTIPOLYGON サブタイプの場合に返されます
GEOMETRYCOLLECTION	GEOMETRYCOLLECTIONM	geom が GEOMETRYCOLLECTION サブタイプの場合に返されます

例

次の SQL は、ポリゴンの WKT (Well-Known Text) 表現を変換し、GEOMETRY サブタイプを文字列として返します。

```
SELECT GeometryType(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
geometrytype
-------------
 POLYGON
```

H3_FromLongLat

H3_FromLongLat は、入力された経度、緯度、および解像度から対応する H3 セル ID を返します。H3 インデックスの詳細については、「H3」を参照してください。

構文

```
H3_FromLongLat(longitude, lattitude, resolution)
```

引数

longitude

データ型 DOUBLE PRECISION の値または DOUBLE PRECISION 型と評価される式の値。

latitude

データ型 DOUBLE PRECISION の値または DOUBLE PRECISION 型と評価される式の値。

resolution

データ型 INTEGER の値または INTEGER 型と評価される式の値。この値は H3 グリッドシステムの解像度を表します。この値は 0～15 の整数である必要があります。0 が最も粗く、15 が最も細かいです。

戻り型

BIGINT — H3 セル ID を表します。

resolution が範囲外の場合、エラーが返されます。

例

次の SQL は、経度 0、緯度 0、解像度 10 から H3 セル ID を返します。

```
SELECT H3_FromLongLat(0, 0, 10);
```

```
 h3_fromlonglat
-------------------
 623560421467684863
```

H3_FromPoint

H3_FromPoint は、入力されたジオメトリポイントおよび解像度から対応する H3 セル ID を返します。H3 インデックスの詳細については、「H3」を参照してください。

構文

```
H3_FromPoint(geom, resolution)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。geom は POINT である必要があります。

resolution

データ型 INTEGER の値または INTEGER 型と評価される式の値。この値は H3 グリッドシステムの解像度を表します。この値は 0～15 の整数である必要があります。0 が最も粗く、15 が最も細かいです。

戻り型

BIGINT — H3 セル ID を表します。

geom が POINT でない場合、エラーが返されます。

resolution が範囲外の場合、エラーが返されます。

geom が空の場合、NULL が返されます。

例

次の SQL は、ポイント 0,0 と解像度 10 から H3 セル ID を返します。

```
SELECT H3_FromPoint(ST_GeomFromText('POINT(0 0)'), 10);
```

```


 h3_frompoint
--------------------
 623560421467684863
```

H3_Polyfill

H3_Polyfill は、指定された解像度の入力ポリゴンに含まれる六角形と五角形に対応する H3 セル ID を返します。H3 インデックスの詳細については、「H3」を参照してください。

構文

```
H3_Polyfill(geom, resolution)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。geom は POLYGON である必要があります。

resolution

データ型 INTEGER の値または INTEGER 型と評価される式の値。この値は H3 グリッドシステムの解像度を表します。この値は 0～15 の整数である必要があります。0 が最も粗く、15 が最も細かいです。

戻り型

SUPER — H3 セル ID のリストを表します。

geom が POLYGON でない場合、エラーが返されます。

resolution が範囲外の場合、エラーが返されます。

geom が空の場合、NULL が返されます。

例

次の SQL は、ポリゴンと解像度 4 から H3 セル ID の SUPER データ型の配列を返します。

```
SELECT H3_Polyfill(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'), 4);
```

```
 h3_polyfill
-------------------------------------------------------------------------------------------------------
 [596538848238895103,596538805289222143,596538856828829695,596538813879156735,59653792052595916
```

ST_AddPoint

ST_AddPoint は、入力ジオメトリと同じラインストリングジオメトリにポイントを追加して返します。インデックスが指定されている場合、ポイントはインデックスの位置に追加されます。インデックスが -1 または指定されていない場合、ポイントはラインストリングに付加されます。

このインデックスは 0 から始まります。結果の空間参照系識別子 (SRID) は、入力ジオメトリのものと同じです。

結果のジオメトリのディメンションは、geom1 値 のものと同じです。geom1 と geom2 でディメンションが異なる場合、geom2 が geom1 のディメンションに射影されます。

構文

```
ST_AddPoint(geom1, geom2)
```

```
ST_AddPoint(geom1, geom2, index)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` である必要があります。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT` である必要があります。ポイントは空のポイントにすることができます。

index

0 から始まるインデックスの位置を表すデータ型 INTEGER の値。

戻り型

GEOMETRY

geom1、geom2、または index が null の場合、null が返されます。

geom2 が空のポイントの場合、geom1 のコピーが返されます。

geom1 が LINESTRING でない場合、エラーが返されます。

geom2 が POINT でない場合、エラーが返されます。

index が範囲外の場合、エラーが返されます。インデックスの位置の有効な値は、-1 または 0 から ST_NumPoints(geom1) の間の値です。

例

次の SQL は、ラインストリングにポイントを追加して、閉じたラインストリングにします。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_StartPoint(g))) FROM tmp;
```

```

 st_asewkt
------------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5,0 0)
```

次の SQL は、ラインストリングの特定の位置にポイントを追加します。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_AddPoint(g, ST_SetSRID(ST_Point(5, 10), 4326), 3)) FROM tmp;
```

```

 st_asewkt
------------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 10,5 5,0 5)
```

ST_Angle

ST_Angle は、次のように時計回りに測定されたポイント間の角度をラジアン単位で返します。

- 3 つのポイントを入力した場合、戻り角度 P1-P2-P3 は、P1 から P3 まで P2 を中心に時計回りに回転して、獲得した角度として測定されます。
- 4 つのポイントを入力した場合、有向線 P1-P2 と P3-P4 がなす時計回りの角度が返されます。入力が縮退の場合 (つまり、P1 が P2 に等しい、または P3 が P4 に等しい場合)、null が返されます。

戻り値はラジアンで、範囲は [0, 2π) です。

ST_Angle は、入力ジオメトリの 2D 射影に対して動作します。

構文

```
ST_Angle(geom1, geom2, geom3)
```

```
ST_Angle(geom1, geom2, geom3, geom4)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT` である必要があります。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT` である必要があります。

geom3

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT` である必要があります。

geom4

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT` である必要があります。

戻り型

`DOUBLE PRECISION`.

geom1 が geom2 と等しい場合、または geom2 が geom3 と等しい場合は、null が返されます。

geom1、geom2、geom3、または geom4 が null の場合は、null が返されます。

geom1、geom2、geom3、または geom4 のいずれかが空のポイントである場合は、エラーが返されます。

geom1、geom2、geom3、geom4 の空間参照系識別子 (SRID) が同じ値ではない場合は、エラーが返されます。

例

次の SQL は、3 つの入力ポイントの角度に変換された角度を返します。

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0)) / Pi() * 180.0 AS angle;
```

```

 angle
---------------
    45
```

次の SQL は、4 つの入力ポイントの角度に変換された角度を返します。

```
SELECT ST_Angle(ST_Point(1,1), ST_Point(0,0), ST_Point(1,0), ST_Point(2,0)) / Pi() *
 180.0 AS angle;
```

```

 angle
---------------
   225
```

ST_Area

ST_Area は、入力ジオメトリの 2D 射影におけるデカルト座標系での面積を返します。面積の単位は、入力ジオメトリの座標を表す単位と同じです。ポイント、ラインストリング、マルチポイント、マルチラインストリングの場合、この関数は 0 を返します。ジオメトリコレクションの場合は、コレクションのジオメトリの合計の面積を返します。

入力がジオグラフィの場合、ST_Area では、SRID で決定される回転楕円体上で計算され与えられた面積ジオグラフィについて、その 2D 射影に関する測地座標系での面積を返します。面積の単位は平方メートルです。この関数は、ポイント、マルチポイント、および線形ジオグラフィに対して 0 を返します。入力がジオメトリコレクションの場合、この関数は、コレクション内の面積ジオグラフィの面積を合計して返します。

構文

```
ST_Area(geo)
```

引数

geo

: データ型 `GEOMETRY` または `GEOGRAPHY` の値、または `GEOMETRY` もしくは `GEOGRAPHY` 型として評価される式です。

戻り型

`DOUBLE PRECISION`

geo が null の場合、null が返されます。

例

次のSQLは、マルチポリゴンのデカルト座標系での面積を返します。

```
SELECT ST_Area(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20 10,10 0)))'));
```

```
 st_area
---------
     100
```

次の SQL は、ポリゴンの面積をジオグラフィで返します。

```
SELECT ST_Area(ST_GeogFromText('polygon((34 35, 28 30, 25 34, 34 35))'));
```

```
     st_area
------------------
 201824655743.383
```

次の SQL は、線形ジオグラフィに対して 0 を返します。

```
SELECT ST_Area(ST_GeogFromText('multipoint(0 0, 1 1, -21.32 121.2)'));
```

```
 st_area
---------
       0
```

ST_AsBinary

ST_AsBinary は、入力ジオメトリの 16 進数の well-known binary (WKB) 式を返します。3DZ、3DM、および 4D ジオメトリの場合、ST_AsBinary はジオメトリタイプとして、Open Geospatial Consortium (OGC) の標準値を使用します。

構文

```
ST_AsBinary(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`VARBYTE`

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンの 16 進数の WKB 表現を返します。

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asbinary
--------------------------------
01030000000100000005000000000000000000000000000000000000000000000000000000000000000000F03F00000
```

ST_AsEWKB

ST_AsEWKB は、入力ジオメトリの Extended Well-Known Binary (EWKB) 表現を返します。3DZ、3DM、および 4D ジオメトリの場合、ST_AsEWKB はジオメトリタイプとして、Open Geospatial Consortium (OGC) の標準値を使用します。

構文

```
ST_AsEWKB(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

VARBYTE

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンの 16 進数の EWKB 表現を返します。

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_asewkb
--------------------------------
0103000020E61000000100000005000000000000000000000000000000000000000000000000000000000000000000F
```

ST_AsEWKT

ST_AsEWKT は、入力されたジオメトリもしくはジオグラフィの、EWKT (Extended Well-Known Text) 表現を返します。ジオメトリが 3DZ、3DM、および 4D の場合、ST_AsEWKT は、ジオメトリタイプの WKT 値に Z、M、または ZM を追加します。

構文

```
ST_AsEWKT(geo)
```

```
ST_AsEWKT(geo, precision)
```

引数

geo

データ型 `GEOMETRY` または `GEOGRAPHY` の値、または `GEOMETRY` もしくは `GEOGRAPHY` 型として評価される式です。

precision

データ型 `INTEGER` の値。ジオメトリについての geo の座標は、1～20 で指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。ジオグラフィの場合、geo の座標は指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。

戻り型

`VARCHAR`

geo が null の場合、null が返されます。

precision が null の場合、null が返されます。

結果が 64-KB `VARCHAR` よりも大きい場合、エラーが返されます。

例

次の SQL は、LINESTRING の EWKT 表現を返します。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_asewkt
--------------------------------
 SRID=4326;LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905
 -1.41421356237309)
```

次の SQL は、LINESTRING の EWKT 表現を返します。ジオメトリの座標は 6 桁の精度で表示されます。

```
SELECT ST_AsEWKT(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_asewkt
--------------------------------
 SRID=4326;LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

次の SQL は、ジオブラフィの EWKT 表現を返します。

```
SELECT ST_AsEWKT(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                  st_asewkt
----------------------------------------------
 SRID=4326;LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_AsGeoJSON

ST_AsGeoJSON は、入力されたジオメトリもしくはジオグラフィの、GeoJSON 表現を返します。GeoJSON に関する詳細は、ウィキペディアの GeoJSON を参照してください。

3DZ および 4D ジオメトリの場合、出力ジオメトリは、入力の 3DZ または 4D ジオメトリの 3DZ 射影となります。すなわち、出力には x、y、および z 座標が含まれます。3DM ジオメトリの場合、出力ジオメトリは入力 3DM ジオメトリの 2D 射影となります。つまり、出力は x および y 座標のみを含みます。

ST_AsGeoJSON は、入力ジオグラフィの GeoJSON 表現を返します。このジオグラフィの座標は、指定された精度を使用して表示されます。

構文

```
ST_AsGeoJSON(geo)
```

```
ST_AsGeoJSON(geo, precision)
```

引数

geo

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。

precision

データ型 INTEGER の値。ジオメトリについての geo の座標は、1～20 で指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。ジオブラフィの場合、geo の座標は指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。

戻り型

VARCHAR

geo が null の場合、null が返されます。

precision が null の場合、null が返されます。

結果が 64-KB VARCHAR よりも大きい場合、エラーが返されます。

例

次の SQL は、LINESTRING の GeoJSON 表現を返します。

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'));
```

```


st_asgeojson
-------------------------------------------------------------------------------------------
```

```
 {"type":"LineString","coordinates":[[3.14159265358979,-6.28318530717959],
[2.71828182845905,-1.41421356237309]]}
```

次の SQL は、LINESTRING の GeoJSON 表現を返します。ジオメトリの座標は 6 桁の精度で表示されます。

```
SELECT ST_AsGeoJSON(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)'), 6);
```

```
st_asgeojson
-----------------------------------------------------------------------------
  {"type":"LineString","coordinates":[[3.14159,-6.28319],[2.71828,-1.41421]]}
```

次の SQL は、ジオグラフィの GeoJSON 表現を返します。

```
SELECT ST_AsGeoJSON(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
                             st_asgeojson
----------------------------------------------------------------------
 {"type":"LineString","coordinates":[[110,40],[2,3],[-10,80],[-7,9]]}
```

ST_AsHexWKB

ST_AsHexWKB は、ASCII 文字 (0～9、A～F) で記述した 16 進数を使用して、入力ジオメトリもしくは入力ジオグラフィの 16 進数による WKB (Well-known binary) 表現を返します。ジオメトリもしくはジオグラフィが 3DZ、3DM、および 4D の場合、ST_AsHexWKB は対象のジオメトリもしくはジオグラフィのタイプとして、Open Geospatial Consortium (OGC) 標準による値を使用します。

構文

```
ST_AsHexWKB(geo)
```

引数

geo

データ型 `GEOMETRY` または `GEOGRAPHY` の値、または `GEOMETRY` もしくは `GEOGRAPHY` 型として評価される式です。

戻り型

VARCHAR

geo が null の場合、null が返されます。

結果が 64-KB VARCHAR よりも大きい場合、エラーが返されます。

例

次の SQL は、ジオメトリ内ポリゴンの 16 進数 WKB 表現を返します。

```
SELECT ST_AsHexWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexwkb
--------------------------------
01030000000100000005000000000000000000000000000000000000000000000000000000000000000000F03F00000
```

次の SQL は、ジオグラフィ内ポリゴンの 16 進数 WKB 表現を返します。

```
SELECT ST_AsHexWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexwkb
--------------------------------
01030000000100000005000000000000000000000000000000000000000000000000000000000000000000F03F00000
```

ST_AsHexEWKB

ST_AsHexEWKB は、ASCII 文字 (0～9、A～F) で記述された 16 進数を使用して、入力ジオメトリもしくは入力ジオグラフィの EWKB (extended well-known binary) 表現を返します。ジオメトリあるいはジオグラフィが 3DZ、3DM、および 4D の場合、ST_AsHexEWKB は、そのジオメトリもしくはジオグラフィのタイプに PostGIS 拡張 WKB 値を使用します。

構文

```
ST_AsHexEWKB(geo)
```

引数

geo

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。

戻り型

VARCHAR

geo が null の場合、null が返されます。

結果が 64-KB VARCHAR よりも大きい場合、エラーが返されます。

例

次の SQL は、ジオメトリ内ポリゴンの 16 進数 EWKB 表現を返します。

```
SELECT ST_AsHexEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

```
st_ashexewkb
--------------------------------
0103000020E61000000010000000500000000000000000000000000000000000000000000000000000000000000000F
```

次の SQL は、ジオグラフィ内ポリゴンの 16 進数 EWKB 表現を返します。

```
SELECT ST_AsHexEWKB(ST_GeogFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'));
```

```
st_ashexewkb
--------------------------------
0103000020E61000000010000000500000000000000000000000000000000000000000000000000000000000000000F
```

ST_AsText

ST_AsText は、入力されたジオメトリもしくはジオグラフィの WKT (Well-Known Text) 表現を返します。ジオメトリもしくはジオグラフィが 3DZ、3DM、および 4D の場合、ST_AsEWKT は、そのジオメトリもしくはジオグラフィのタイプを表す WKT 値に、Z、M、または ZM を追加します。

構文

```
ST_AsText(geo)
```

```
ST_AsText(geo, precision)
```

引数

geo

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。

precision

データ型 INTEGER の値。ジオメトリについての geo の座標は、1～20 で指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。ジオグラフィの場合、geom の座標は指定された精度を使用して表示されます。precision が指定されていない場合は、デフォルトは 15 です。

戻り型

VARCHAR

geo が null の場合、null が返されます。

precision が null の場合、null が返されます。

結果が 64-KB VARCHAR よりも大きい場合、エラーが返されます。

例

次の SQL は、LINESTRING の WKT 表現を返します。

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326));
```

```
st_astext
---------------------------------
LINESTRING(3.14159265358979 -6.28318530717959,2.71828182845905 -1.41421356237309)
```

次の SQL は、LINESTRING の WKT 表現を返します。ジオメトリの座標は 6 桁の精度で表示されます。

```
SELECT ST_AsText(ST_GeomFromText('LINESTRING(3.141592653589793
 -6.283185307179586,2.718281828459045 -1.414213562373095)', 4326), 6);
```

```
st_astext
---------------------------------------------
 LINESTRING(3.14159 -6.28319,2.71828 -1.41421)
```

次の SQL は、ジオグラフィの WKT 表現を返します。

```
SELECT ST_AsText(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```
             st_astext
------------------------------------
 LINESTRING(110 40,2 3,-10 80,-7 9)
```

ST_Azimuth

ST_Azimuth は、2 つの入力ポイントの 2D 射影を使用して、北を基準としたデカルト方位角を返します。

構文

```
ST_Azimuth(point1, point2)
```

引数

point1

データ型 `GEOMETRY` の `POINT` 値。point1 の空間リファレンスシステム識別子 (SRID) は point2 の SRID と一致する必要があります。

point2

データ型 `GEOMETRY` の `POINT` 値。point2 の SRID は、point1 の SRID と一致する必要があります。

戻り型

DOUBLE PRECISION データ型のラジアンの角度である数字。値の範囲は 0 (包括的) から 2 pi (排他的) です。

point1 または point2 が空のポイントの場合は、エラーが返されます。

point1 または point2 のいずれかが null の場合、null が返されます。

point1 および point2 が等しい場合、null が返されます。

point1 または point2 がポイントでない場合、エラーが返されます。

geom1 および geom2 に空間リファレンスシステム識別子 (SRID) の値がない場合、エラーが返されます。

例

次の SQL は、入力ポイントの方位角を返します。

```
SELECT ST_Azimuth(ST_Point(1,2), ST_Point(5,6));
```

```


st_azimuth
-------------------
 0.7853981633974483
```

ST_Boundary

ST_Boundary は、次のように入力ジオメトリの境界を返します。

- 入力ジオメトリが空の場合 (つまり、ポイントが含まれていない場合)、そのまま返されます。
- 入力ジオメトリがポイントまたは空ではないマルチポイントの場合、空のジオメトリコレクションが返されます。
- 入力がライン文字列またはマルチライン文字列の場合、境界上のすべてのポイントを含むマルチポイントが返されます。マルチポイントが空である可能性があります)。
- 入力が内部リングのないポリゴンである場合、その境界を表す閉じたライン文字列が返されます。
- 入力が内部リングのあるポリゴンであるか、マルチポリゴンである場合、マルチライン文字列が返されます。マルチライン文字列には、面積ジオメトリ内のすべてのリングのすべての境界が閉じたライン文字列として含まれます。

ポイントの等価性を判断するために、ST_Boundary は入力ジオメトリの 2D 射影に対して作用します。入力ジオメトリが空の場合、入力のコピーが同じディメンションで返されます。空でない 3DM および 4D ジオメトリの場合、m 座標はドロップされます。3DZ および 4D マルチライン文字列の特殊なケースでは、マルチラインストリングの境界ポイントにおける z 座標は、同じ 2D 射影法を使用するライン文字列の境界ポイントにおける、個別の Z 値の平均として計算されます。

構文

```
ST_Boundary(geom)
```

引数

geom

　データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

geom が null の場合、null が返されます。

geom が GEOMETRYCOLLECTION である場合、エラーが返されます。

例

次の SQL は、入力ポリゴンの境界をマルチライン文字列として返します。

```
SELECT ST_AsEWKT(ST_Boundary(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1
 2,2 1,1 1))')));
```

```
st_asewkt
---------------------
 MULTILINESTRING((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1))
```

ST_Buffer

ST_Buffer は、xy-デカルト平面に投影された入力ジオメトリからの距離が入力距離以下であるすべてのポイントを表す 2D ジオメトリを返します。

構文

```
ST_Buffer(geom, distance)
```

```
ST_Buffer(geom, distance, number_of_segments_per_quarter_circle)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

distance

バッファの距離 (半径) を表すデータ型 DOUBLE PRECISION の値。

number_of_segments_per_quarter_circle

データ型 INTEGER の値。この値は、入力ジオメトリの各頂点の周りの四分円を近似するためのポイントの数を決定します。負の値はデフォルトでゼロになります。デフォルトは 8 です。

戻り型

GEOMETRY

ST_Buffer 関数は、xy-デカルト平面内の 2 次元 (2D) ジオメトリを返します。

geom が GEOMETRYCOLLECTION である場合、エラーが返されます。

例

以下の SQL は、入力ラインストリングのバッファを返します。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('LINESTRING(1 2,5 2,5 8)'), 2));
```

```
               st_asewkt
POLYGON((-1 2,-0.96157056080646 2.39018064403226,-0.847759065022573
 2.76536686473018,-0.662939224605089 3.11114046603921,-0.414213562373093
 3.4142135623731,-0.111140466039201 3.66293922460509,0.234633135269824
 3.84775906502257,0.609819355967748 3.96157056080646,1 4,3 4,3 8,3.03842943919354
 8.39018064403226,3.15224093497743 8.76536686473018,3.33706077539491
```

```
 9.11114046603921,3.58578643762691 9.4142135623731,3.8888595339608
 9.66293922460509,4.23463313526982 9.84775906502257,4.60981935596775
 9.96157056080646,5 10,5.39018064403226 9.96157056080646,5.76536686473018
 9.84775906502257,6.11114046603921 9.66293922460509,6.4142135623731
 9.41421356237309,6.66293922460509 9.1111404660392,6.84775906502258
 8.76536686473017,6.96157056080646 8.39018064403225,7 8,7 2,6.96157056080646
 1.60981935596774,6.84775906502257 1.23463313526982,6.66293922460509
 0.888859533960796,6.41421356237309 0.585786437626905,6.1111404660392
 0.33706077539491,5.76536686473018 0.152240934977427,5.39018064403226
 0.0384294391935391,5 0,1 0,0.609819355967744 0.0384294391935391,0.234633135269821
 0.152240934977427,-0.111140466039204 0.337060775394909,-0.414213562373095
 0.585786437626905,-0.662939224605091 0.888859533960796,-0.847759065022574
 1.23463313526982,-0.961570560806461 1.60981935596774,-1 2))
```

以下の SQL は、円を近似する入力ポイントジオメトリのバッファを返します。このコマンドは四分円あたりのセグメント数を指定しないため、関数はデフォルト値の 8 セグメントを使用して四分円を近似します。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2));
```

```
               st_asewkt
POLYGON((1 4,1.03842943919354 4.39018064403226,1.15224093497743
 4.76536686473018,1.33706077539491 5.11114046603921,1.58578643762691
 5.4142135623731,1.8888595339608 5.66293922460509,2.23463313526982
 5.84775906502257,2.60981935596775 5.96157056080646,3 6,3.39018064403226
 5.96157056080646,3.76536686473019 5.84775906502257,4.11114046603921
 5.66293922460509,4.4142135623731 5.41421356237309,4.66293922460509
 5.1111404660392,4.84775906502258 4.76536686473017,4.96157056080646 4.39018064403225,5
 4,4.96157056080646 3.60981935596774,4.84775906502257 3.23463313526982,4.66293922460509
 2.8888595339608,4.41421356237309 2.58578643762691,4.1111404660392
 2.33706077539491,3.76536686473018 2.15224093497743,3.39018064403226 2.03842943919354,3
 2,2.60981935596774 2.03842943919354,2.23463313526982 2.15224093497743,1.8888595339608
 2.33706077539491,1.58578643762691 2.58578643762691,1.33706077539491
 2.8888595339608,1.15224093497743 3.23463313526982,1.03842943919354 3.60981935596774,1
 4))
```

以下の SQL は、円を近似する入力ポイントジオメトリのバッファを返します。このコマンドは四分円あたりのセグメント数として 3 を指定するため、関数は 3 つのセグメントを使用して四分円を近似します。

```
SELECT ST_AsEwkt(ST_Buffer(ST_GeomFromText('POINT(3 4)'), 2, 3));
```

```
                st_asewkt
POLYGON((1 4,1.26794919243112 5,2 5.73205080756888,3 6,4
 5.73205080756888,4.73205080756888 5,5 4,4.73205080756888 3,4 2.26794919243112,3 2,2
 2.26794919243112,1.26794919243112 3,1 4))
```

ST_Centroid

ST_CentroID は、ジオメトリの重心を表すポイントを次のように返します。

- POINT ジオメトリの場合、ジオメトリ内にあるポイントの座標の平均を座標とするポイントを返します。
- LINESTRING ジオメトリの場合、ジオメトリのセグメントの中間点の加重平均を座標とするポイントを返します。この際、加重はジオメトリのセグメントの長さとなります。
- POLYGON ジオメトリの場合、面積ジオメトリの三角形分割における重心の加重平均を座標とするポイントを返します。ここで加重は、三角形分割を構成する三角形の面積が使用されます。
- ジオメトリコレクションの場合、ジオメトリコレクション内の最大トポロジディメンションの、ジオメトリにおける重心の加重平均を返します。

構文

```
ST_Centroid(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

geom が null の場合、null が返されます。

geom が空の場合、null が返されます。

例

次の SQL は、入力ラインストリングの中心点を返します。

```
SELECT ST_AsEWKT(ST_Centroid(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9, -22
 -33)', 4326)))
```

```
                     st_asewkt
----------------------------------------------------
 SRID=4326;POINT(15.6965103455214 27.0206782881905)
```

ST_Collect

ST_Collect には 2 つの変形があります。1 つは 2 つのジオメトリを受け入れ、もう 1 つは集約式を受け入れます。

ST_Collect の 1 番目の変形は、入力ジオメトリからジオメトリを作成します。入力ジオメトリの順序は保持されます。このバリアントは次のように動作します。

- 両方の入力ジオメトリがポイントの場合、2 つのポイントを持つ `MULTIPOINT` が返されます。
- 両方の入力ジオメトリがライン文字列である場合、2 つのライン文字列を持つ `MULTILINESTRING` が返されます。
- 両方の入力ジオメトリがポリゴンの場合、2 つのポリゴンを持つ `MULTIPOLYGON` が返されます。
- それ以外の場合は、2 つの入力ジオメトリを持つ `GEOMETRYCOLLECTION` が返されます。

ST_Collect の 2 番目の変形は、ジオメトリ列のジオメトリからジオメトリを作成します。ジオメトリの戻り順序が決定されていません。WITHIN GROUP (ORDER BY...) 句を指定して、返されるジオメトリの順序を指定します。このバリアントは次のように動作します。

- 入力集計式の NULL 以外の行がすべてポイントである場合、集計式のすべてのポイントを含むマルチポイントが返されます。
- 集約式の NULL 以外の行がすべてライン文字列の場合、集約式内のすべてのライン文字列を含むマルチライン文字列が返されます。
- 集約式内の NULL 以外の行がすべてポリゴンの場合、その結果は、集約式内のすべてのポリゴンを含むマルチポリゴンが返されます。
- それ以外の場合は、集計式のすべてのジオメトリを含む `GEOMETRYCOLLECTION` が返されます。

ST_Collect は、入力ジオメトリと同じディメンションのジオメトリを返します。入力ジオメトリはすべて、同じディメンションにする必要があります。

構文

```
ST_Collect(geom1, geom2)
```

```
ST_Collect(aggregate_expression)  [WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC]
 [, sort_expression2 [ASC | DESC] ...])]
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

aggregate_expression

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の列。

[WITHIN GROUP (ORDER BY sort_expression1 [ASC | DESC] [, sort_expression2 [ASC | DESC] ...])]

オプションの集計値のソート順を指定する句。ORDER BY 句には、ソート式のリストが含まれています。ソート式は、列名など、クエリ選択リストの有効なソート式に似た式です。昇順 (ASC) または降順 (DESC) の順を指定できます。デフォルト: ASC。

戻り型

サブタイプ MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、または GEOMETRYCOLLECTION の GEOMETRY。

返されたジオメトリの空間リファレンス識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom1 または geom2 が両方とも null の場合、null が返されます。

aggregate_expression のすべての行が null の場合、null が返されます。

geom1 が null の場合、geom2 のコピーが返されます。同様に、geom2 が null の場合、geom1 のコピーが返されます。

geom1 および geom2 の SRID 値が異なる場合、エラーが返されます。

aggregate_expression の 2 つのジオメトリの SRID 値が異なる場合、エラーが返されます。

返されるジオメトリが GEOMETRY の最大サイズよりも大きい場合、エラーが返されます。

geom1 と geom2 が異なるディメンションの場合は、エラーが返されます。

aggregate_expression 内の 2 つのジオメトリの ディメンションが異なる場合、エラーが返されます。

例

次の SQL は、2 つの入力ジオメトリを含むジオメトリコレクションを返します。

```
SELECT ST_AsText(ST_Collect(ST_GeomFromText('LINESTRING(0 0,1 1)'),
 ST_GeomFromText('POLYGON((10 10,20 10,10 20,10 10))')));
```

```


st_astext
-----------
 GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),POLYGON((10 10,20 10,10 20,10 10)))
```

次の SQL は、テーブルからジオメトリコレクションにすべてのジオメトリを収集します。

```
WITH tbl(g) AS (SELECT ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT NULL::geometry UNION ALL
SELECT ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326))
SELECT ST_AsEWKT(ST_Collect(g)) FROM tbl;
```

```


st_astext
-----------
 SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(0 0,10 0),MULTIPOINT((13 4),(8 5),
(4 4)),POLYGON((0 0,10 0,0 10,0 0)))
```

次の SQL は、id 列でグループ化され、この ID で並べ替えられたテーブル内のすべてのジオメトリを収集します。この例では、結果のジオメトリは ID によって次のようにグループ化されます。

- id 1 – マルチポイント内のポイント。
- id 2 – マルチライン文字列内のライン文字列。

- id 3 – ジオメトリコレクションに含まれる混在サブタイプ。
- id 4 – マルチポリゴンのポリゴン。
- id 5 – null であり、結果はnull です。

```
WITH tbl(id, g) AS (SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT id, ST_AsEWKT(ST_Collect(g)) FROM tbl GROUP BY id ORDER BY id;
```

```
 id |                                                 st_asewkt

----
+--------------------------------------------------------------------------------------------------
  1 | SRID=4326;MULTIPOINT((1 2),(4 5))
  2 | SRID=4326;MULTILINESTRING((0 0,10 0),(10 0,20 -5))
  3 | SRID=4326;GEOMETRYCOLLECTION(MULTIPOINT((13 4),(8 5),(4 4)),MULTILINESTRING((-1
 -1,-2 -2),(-3 -3,-5 -5)))
  4 | SRID=4326;MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((20 20,20 30,30 20,20 20)))
  5 |
```

次の SQL は、ジオメトリコレクションのテーブルからすべてのジオメトリを収集します。結果は id の降順で並べられ、次に最小および最大の x 座標に基づいて辞書式の順で並べられます。

```
WITH tbl(id, g) AS (
SELECT 1, ST_GeomFromText('POINT(4 5)', 4326) UNION ALL
SELECT 1, ST_GeomFromText('POINT(1 2)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(10 0,20 -5)', 4326) UNION ALL
SELECT 2, ST_GeomFromText('LINESTRING(0 0,10 0)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTIPOINT(13 4,8 5,4 4)', 4326) UNION ALL
SELECT 3, ST_GeomFromText('MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5))', 4326) UNION
 ALL
```

```
SELECT 4, ST_GeomFromText('POLYGON((20 20,20 30,30 20,20 20))', 4326) UNION ALL
SELECT 4, ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))', 4326) UNION ALL
SELECT 1, NULL::geometry UNION ALL SELECT 2, NULL::geometry UNION ALL
SELECT 5, NULL::geometry UNION ALL SELECT 5, NULL::geometry)
SELECT ST_AsEWKT(ST_Collect(g) WITHIN GROUP (ORDER BY id DESC, ST_XMin(g), ST_XMax(g)))
 FROM tbl;
```

```
                           st_asewkt

-----------------------------------------------------------------------------------------------------
 SRID=4326;GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0 0)),POLYGON((20 20,20 30,30
 20,20 20)),MULTILINESTRING((-1 -1,-2 -2),(-3 -3,-5 -5)),MULTIPOINT((13 4),(8 5),(4
 4)),LINESTRING(0 0,10 0),LINESTRING(10 0,20 -5),POINT(1 2),POINT(4 5)
```

ST_Contains

ST_Contains は、最初の入力ジオメトリの 2D 投影に、2 番目の入力ジオメトリの 2D 投影が含まれている場合に true を返します。A のすべてのポイントが B のポイントであり、それらの内部の交差が空ではない場合に、ジオメトリ B はジオメトリ A を含みます。

ST_Contains(A、B) は ST_Within(B、A) と同等です。

構文

```
ST_Contains(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。この値を geom1 と比較して、geom1 に含まれているかどうかを判断します。

戻り型

`BOOLEAN`

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴンを含むかどうかを確認します。

```
SELECT ST_Contains(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```


st_contains
-----------
 false
```

ST_ContainsProperly

両方の入力ジオメトリが空ではなく、2 番目のジオメトリにおける 2D 射影のすべてのポイントが、1 番目のジオメトリにおける 2D 射影の内部ポイントである場合、ST_ContainsProperly は true を返します。

構文

```
ST_ContainsProperly(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `GEOMETRYCOLLECTION` にすることはできません。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `GEOMETRYCOLLECTION` にすることはできません。この値を geom1 と比較して、そのすべてのポイントが geom1 の内部ポイントであるかどうかを判断します。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、ST_Contains および ST_ContainsProperly の値を返します。ここで、入力ライン文字列は、入力ポリゴンの内部と境界で交差しますが、外部では交差しません。ポリゴンにはライン文字列が含まれていますが、含まれているライン文字列が正しくありません。

```
WITH tmp(g1, g2)
AS (SELECT ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(5 5,10 5,10 6,5 5)')) SELECT ST_Contains(g1, g2),
 ST_ContainsProperly(g1, g2)
FROM tmp;
```

```
 st_contains | st_containsproperly
-------------+---------------------
 t           | f
```

ST_ConvexHull

ST_ConvexHull は、入力ジオメトリに含まれる、空でないポイントの凸包を表すジオメトリを返します。

入力が空の場合は、結果のジオメトリは入力ジオメトリと同じになります。空でないすべての入力に対して、この関数は入力ジオメトリの 2D 射影を処理します。ただし、出力ジオメトリのディメンションは、入力ジオメトリのディメンションによって異なります。具体的には、入力が空でない 3DM または 3D ジオメトリの場合、m 座標がドロップされます。つまり、返されるジオメトリのディメンションは、それぞれ 2D または 3DZ となります。入力が空でない 2D または 3DZ ジオメトリの場合、結果のジオメトリのディメンションは同じになります。

構文

```
ST_ConvexHull(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

返される値は次のとおりです。

凸包上のポイント数	ジオメトリのサブタイプ
0	geom のコピーが返されます。
1	POINT サブタイプが返されます。
2	LINESTRING サブタイプが返されます。返されるライン文字列の 2 つのポイントは、辞書式の順序になっています。
3 以上	内部リングのない POLYGON サブタイプが返されます。ポリゴンは時計回りに向けられており、外部リングの最初のポイントは、辞書式順序でリングの最小のポイントです。

例

次の SQL は、LINESTRING の拡張された well-known text (EWKT) 表現を返します。この場合、返される凸包はポリゴンです。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 0,0 1,1 1,0.5 0.5)')))
 as output;
```

```
output
-------------
POLYGON((0 0,0 1,1 1,1 0,0 0))
```

次の SQL は、LINESTRING の EWKT 表現を返します。この場合、返される凸包はライン文字列です。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('LINESTRING(0 0,1 1,0.2 0.2,0.6 0.6,0.5
 0.5)'))) as output;
```

```
output
-------------
LINESTRING(0 0,1 1)
```

次の SQL は、マルチポイントの EWKT 表現を返します。この場合、返される凸包はポイントです。

```
SELECT ST_AsEWKT(ST_ConvexHull(ST_GeomFromText('MULTIPOINT(0 0,0 0,0 0)'))) as output;
```

```
output
-------------
POINT(0 0)
```

ST_CoveredBy

ST_CoveredBy は、最初の入力ジオメトリの 2D 射影が、 2 番目の入力ジオメトリの 2D 射影に包含される場合 true を返します。どちらも空ではなく、A のすべてのポイントが B のポイントである場合、ジオメトリ A はジオメトリ B で覆われます。

ST_CoveredBy(A、B) は ST_Covers(B、A) と同等です。

構文

```
ST_CoveredBy(geom1, geom2)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。この値を geom2 と比較して、geom2 でカバーされているかどうかを判断します。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴンで覆われるかどうかを確認します。

```
SELECT ST_CoveredBy(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```

st_coveredby
-----------
 true
```

ST_Covers

ST_Covers は、最初の入力ジオメトリの 2D 射影が、2 番目の入力ジオメトリの 2D 射影を包含する場合、true を返します。両方が空でなく、A のすべてのポイントが B のポイントである場合に、ジオメトリ B は、ジオメトリ A を覆います。

ST_Covers(A、B) は ST_CoveredBy(B、A) と同等です。

構文

```
ST_Covers(geom1, geom2)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。この値を geom1 と比較して、geom1 をカバーしているかどうかを判断します。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴンを覆うかどうかを確認します。

```
SELECT ST_Covers(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```


st_covers
-----------
 false
```

ST_Crosses

ST_Crosses は、2 つの入力ジオメトリの 2D 射影が互いに交差する場合に true を返します。

構文

```
ST_Crosses(geom1, geom2)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のマルチポイントと交差するかどうかを確認します。この例では、マルチポイントがポリゴンの内部と外部の両方と交差しているため、ST_Crosses は true を返します。

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(5 5,0 0,-1 -1)'));
```

```


st_crosses
-------------
 true
```

次の SQL は、最初のポリゴンが 2 番目のマルチポイントと交差するかどうかを確認します。この例では、マルチポイントはポリゴンの内部ではなく外部と交差します。そのため、ST_Crosses は false を返します。

```
SELECT ST_Crosses (ST_GeomFromText('polygon((0 0,10 0,10 10,0 10,0 0))'),
 ST_GeomFromText('multipoint(0 0,-1 -1)'));
```

```
st_crosses
-------------
 false
```

ST_Dimension

ST_Dimension は、入力ジオメトリの固有のディメンションを返します。inherent dimension は、ジオメトリで定義されているサブタイプのディメンション値です。

ST_Dimension は、3DM、3DZ、4D ジオメトリが入力された場合、2D ジオメトリが入力された場合と同じ結果を返します。

構文

```
ST_Dimension(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`INTEGER` は geom 固有のディメンションを表します。

geom が null の場合、null が返されます。

返される値は次のとおりです。

戻り値	ジオメトリのサブタイプ
0	geom が `POINT` または `MULTIPOINT` サブタイプの場合に返されます

戻り値	ジオメトリのサブタイプ
1	geom が LINESTRING または MULTILINESTRING サブタイプの場合に返されます
2	geom が POLYGON または MULTIPOLYGON サブタイプの場合に返されます
0	geom が空の GEOMETRYCOLLECTION サブタイプの場合に返されます
コレクションのコンポーネントの最大ディメンション	geom が GEOMETRYCOLLECTION サブタイプの場合に返されます

例

次の SQL は、4 ポイントの LINESTRING の WKT (Well-Known Text) 表現を GEOMETRY オブジェクトに変換し、LINESTRING のディメンションを返します。

```
SELECT ST_Dimension(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_dimension
-------------
1
```

ST_Disjoint

ST_Disjoint は、2 つの入力ジオメトリの 2D 射影に共通するポイントがない場合に true を返します。

構文

```
ST_Disjoint(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴンから切り離されているかどうかを確認します。

```
SELECT ST_Disjoint(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_Point(4, 4));
```

```


st_disjoint
-----------
 true
```

ST_Distance

入力がジオメトリの場合、ST_Distance は 2 つのジオメトリの 2D 射影による値の間で、最短のユークリッド距離を返します。

ジオメトリが 3DM、3DZ、4D の場合、ST_Distance は、2 つの入力ジオメトリ値の 2D 間のユークリッド距離を返します。

入力ジオグラフィの場合、ST_Distance は 2 つの 2D ポイントの測地的距離を返します。距離の単位はメートルです。ポイントと空ポイント以外のジオグラフィの場合、エラーが返されます。

構文

```
ST_Distance(geo1, geo2)
```

引数

geo1

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。geo1 のデータ型は geo2 と同じである必要があります。

geo2

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。geo2 のデータ型は geo1 と同じである必要があります。

戻り型

入力のジオメトリもしくはジオグラフィと同じ単位の DOUBLE PRECISION です。

geo1 または geo2 が null または空の場合、null が返されます。

geo1 および geo2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geo1 または geo2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、2 つのポリゴン間の距離を返します。

```
SELECT ST_Distance(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 -3,-2 -1,0 -3,-1 -3))'));
```

```

  st_distance
-----------
1.4142135623731
```

次の SQL は、GEOGRAPHY データ型を使用して、ベルリンのブランデンブルク門と国会議事堂の間の距離 (メートル単位) を返します。

```
SELECT ST_Distance(ST_GeogFromText('POINT(13.37761826722198 52.516411678282445)'),
 ST_GeogFromText('POINT(13.377950831464005 52.51705102546893)'));
```

```
   st_distance
------------------
 74.64129172609631
```

ST_DistanceSphere

ST_DistanceSphere は、球面の 2 つのポイントのジオメトリ間の距離を返します。

構文

```
ST_DistanceSphere(geom1, geom2)
```

```
ST_DistanceSphere(geom1, geom2, radius)
```

引数

geom1

球面のデータ型 GEOMETRY の角度のポイント値。ポイントの最初の座標は経度の値です。ポイントの 2 番目の座標は緯度の値です。3DZ、3DM、または 4D ジオメトリの場合、最初の 2 つの座標のみが使用されます。

geom2

球面のデータ型 GEOMETRY の角度のポイント値。ポイントの最初の座標は経度の値です。ポイントの 2 番目の座標は緯度の値です。3DZ、3DM、または 4D ジオメトリの場合、最初の 2 つの座標のみが使用されます。

radius

データ型 DOUBLE PRECISION の球の半径。radius が指定されていない場合、球体はデフォルトで地球に設定され、半径は楕円体の WGS (World Geodetic System) 84 表現から計算されます。

戻り型

半径と同じユニットの DOUBLE PRECISION。半径が指定されていない場合、距離はメートル単位です。

geom1 または geom2 が null または空の場合、null が返されます。

radius が指定されていない場合、地球の表面に沿った結果がメートルで表示されます。

radius が負数である場合、エラーが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がポイントでない場合、エラーが返されます。

例

次の SQL の例では、地球上の 2 点間の距離をキロメートル単位で計算します。

```
SELECT ROUND(ST_DistanceSphere(ST_Point(-122, 47), ST_Point(-122.1, 47.1))/ 1000, 0);
```

```
  round
-----------
 13
```

次の SQL の例では、ドイツのベルリンテーゲル空港 (TXL)、ミュンヘン空港 (MUC)、フランクフルト空港 (FRA) の 3 つのポイントの場所の間の距離をキロメートルで計算します。

```
WITH airports_raw(code,lon,lat) AS (
(SELECT 'MUC', 11.786111, 48.353889) UNION
(SELECT 'FRA', 8.570556, 50.033333) UNION
(SELECT 'TXL', 13.287778, 52.559722)),
airports1(code,location) AS (SELECT code, ST_Point(lon, lat) FROM airports_raw),
airports2(code,location) AS (SELECT * from airports1)
SELECT (airports1.code || ' <-> ' || airports2.code) AS airports,
round(ST_DistanceSphere(airports1.location, airports2.location) / 1000, 0) AS
 distance_in_km
FROM airports1, airports2 WHERE airports1.code < airports2.code ORDER BY 1;
```

```
  airports   | distance_in_km
-------------+----------------
 FRA <-> MUC |            299
 FRA <-> TXL |            432
```

```
MUC <-> TXL |            480
```

ST_DWithin

ST_DWithin は、2 つの入力ジオメトリの 2D 射影値の間のユークリッド距離が、しきい値以下の場合に true を返します。

構文

```
ST_DWithin(geom1, geom2, threshold)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

threshold

データ型 DOUBLE PRECISION の値。この値は入力引数のユニットです。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

threshold が負数の場合、エラーが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、2 つのポリゴン間の距離が 5 ユニットいないであるかどうかを確認します。

```
SELECT ST_DWithin(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'),5);
```

```
st_dwithin
-----------
 true
```

ST_EndPoint

ST_EndPoint は、入力ラインストリングの最後のポイントを返します。結果の空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_EndPoint(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` である必要があります。

戻り型

`GEOMETRY`

geom が null の場合、null が返されます。

geom が空の場合、null が返されます。

geom が `LINESTRING` でない場合、null が返されます。

例

次の SQL は、`GEOMETRY` オブジェクトの 4 つのポイントを持つ `LINESTRING` の EWKT (Extended well-known text) 表現を返し、そのラインストリングの最後のポイントを返します。

```
SELECT ST_AsEWKT(ST_EndPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```
st_asewkt
-------------
 SRID=4326;POINT(0 5)
```

ST_Envelope

ST_Envelope は、次のように、入力ジオメトリの最小の境界ボックスを返します。

- 入力ジオメトリが空の場合、返されるジオメトリは入力ジオメトリのコピーです。
- 入力ジオメトリの最小の境界ボックスに含まれるのが 1 つのポイントのみの場合、返されるジオメトリは 1 つのポイントです。
- 入力ジオメトリの最小の境界ボックスが 1 次元である場合、2 つのポイントのラインストリングが返されます。
- 上記のいずれにも該当しない場合、この関数は、その頂点を最小の境界ボックスの角とする時計回りのポリゴンを返します。

返されるジオメトリの空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。

空でないすべての入力に対して、この関数は入力ジオメトリの 2D 射影を処理します。

構文

```
ST_Envelope(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`GEOMETRY`

geom が null の場合、null が返されます。

例

次の SQL は、GEOMETRY オブジェクトの 4 つのポイントを持つ LINESTRING の WKT (Well-known text) 表現を変換し、その頂点を最小の境界ボックスの角とするポリゴンを返します。

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0
 10,0 0)),LINESTRING(20 10,20 0,10 0))')));
```

```


    st_astext
------------------------------------
  POLYGON((0 0,0 10,20 10,20 0,0 0))
```

ST_Equals

ST_Equals は、それぞれの入力ジオメトリの 2D 射影が、幾何学的に等しい場合に true を返します。ジオメトリは、ポイントセットが等しく、内部の交差が空でない場合に幾何学的に等しいと考えられます。

構文

```
ST_Equals(geom1, geom2)
```

引数

geom1

　データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

　データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。この値を geom1 と比較して、geom1 と等しいかどうかを判断します。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、エラーが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、2 つのポリゴンが幾何学的に等しいかどうかを確認します。

```
SELECT ST_Equals(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```
st_equals
-----------
 false
```

次の SQL は、2 つの LINESTRING が幾何学的に等しいかどうかを確認します。

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(1 0,10 0)'), ST_GeomFromText('LINESTRING(1
 0,5 0,10 0)'));
```

```
st_equals
-----------
 true
```

ST_ExteriorRing

ST_ExteriorRing は、入力ポリゴンの外部リングを表す、閉じたライン文字列を返します。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_ExteriorRing(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`GEOMETRY`サブタイプ の `LINESTRING`。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

geom がポリゴンでない場合、null が返されます。

geom が空の場合、空のポリゴンが返されます。

例

次の SQL は、ポリゴンの外部リングを閉じたライン文字列として返します。

```
SELECT ST_AsEWKT(ST_ExteriorRing(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))')));
```

```
st_asewkt
-----------
 LINESTRING(7 9,8 7,11 6,15 8,16 6,17 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9)
```

ST_Force2D

ST_Force2D は、入力ジオメトリの 2D ジオメトリを返します。2D ジオメトリの場合、入力のコピーが返されます。ジオメトリが 3DZ、3DM、4D の場合、ST_Force2D はジオメトリを XY デカルト平面に射影します。入力ジオメトリの空のポイントは、出力ジオメトリでも空のポイントのままで維持されます。

構文

```
ST_Force2D(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

GEOMETRY.

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

geom が空の場合、空のジオメトリが返されます。

例

次の SQL は、3DZ ジオメトリから 2D ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Force2D(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```


st_asewkt
-----------
  MULTIPOINT((0 1),EMPTY,(2 3),(5 6))
```

ST_Force3D

ST_Force3D は、ST_Force3DZ のエイリアスです。詳細については、「ST_Force3DZ」を参照してください。

ST_Force3DM

ST_Force3DM は、入力ジオメトリの 3DM ジオメトリを返します。2D ジオメトリの場合、出力ジオメトリ内の空でないポイントのための `m` 座標は、すべて `0` に設定されます。3DM ジオメトリの場合、入力ジオメトリのコピーが返されます。3DZ ジオメトリの場合、ジオメトリは XY デカルト平面に射影され、出力ジオメトリ内の空でないポイントの `m` 座標は、すべて `0` に設定されます。4D ジオメトリの場合、ジオメトリは XYM デカルト空間に射影されます。入力ジオメトリの空のポイントは、出力ジオメトリでも空のポイントのままで維持されます。

構文

```
ST_Force3DM(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY.

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

geom が空の場合、空のジオメトリが返されます。

例

次の SQL は、3DZ ジオメトリから 3DM ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Force3DM(ST_GeomFromText('MULTIPOINT Z(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```


st_asewkt
-----------
  MULTIPOINT M ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force3DZ

ST_Force3DZ は、入力ジオメトリから 3DZ ジオメトリを返します。2D ジオメトリの場合、出力ジオメトリ内の空でないポイントのための z 座標は、すべて 0 に設定されます。ジオメトリが 3DM の場合、ジオメトリは XY デカルト平面上に射影され、出力ジオメトリ内の空でないポイントの z 座標は、すべて 0 に設定されます。ジオメトリが 3DZ の場合には、入力ジオメトリのコピーが返されます。4D ジオメトリの場合は、ジオメトリは XYZ デカルト空間に射影されます。入力ジオメトリの空のポイントは、出力ジオメトリでも空のポイントのままで維持されます。

構文

```
ST_Force3DZ(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY.

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

geom が空の場合、空のジオメトリが返されます。

例

次の SQL は、3DM ジオメトリから 3DZ ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Force3DZ(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```


st_asewkt
-----------
  MULTIPOINT Z ((0 1 0),EMPTY,(2 3 0),(5 6 0))
```

ST_Force4D

ST_Force4D は、入力ジオメトリの 4D ジオメトリを返します。2D ジオメトリの場合、出力ジオメトリ内の空でないポイントにおける z および m 座標は、すべて 0 に設定されます。ジオメトリが 3DM の場合は、出力ジオメトリ内の空でないポイントの z 座標は、すべて 0 設定されます。ジオメトリが 3DZ の場合は、出力ジオメトリ内の空でないポイントの m 座標は、すべて 0 に設定されます。4D ジオメトリの場合には、入力ジオメトリのコピーが返されます。入力ジオメトリの空のポイントは、出力ジオメトリでも空のポイントのままで維持されます。

構文

```
ST_Force4D(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY.

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom が null の場合、null が返されます。

geom が空の場合、空のジオメトリが返されます。

例

次の SQL は、3DM ジオメトリから 4D ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Force4D(ST_GeomFromText('MULTIPOINT M(0 1 2, EMPTY, 2 3 4, 5 6
 7)')));
```

```


st_asewkt
-----------
  MULTIPOINT ZM ((0 1 0 2),EMPTY,(2 3 0 4),(5 6 0 7))
```

ST_GeoHash

ST_GeoHash は、指定された精度で入力ポイントの geohash 表現を返します。デフォルトの精度値は 20 です。geohash の定義についての詳細は、Wikipedia の「ジオハッシュ」を参照してください。

構文

```
ST_GeoHash(geom)
```

```
ST_GeoHash(geom, precision)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

precision

データ型 INTEGER の値。デフォルトは 20 です。

戻り型

GEOMETRY

この関数は、入力ポイントの geohash 表現を返します。

入力ポイントが空の場合、関数は null を返します。

入力ジオメトリがポイントではない場合、関数はエラーを返します。

例

以下の SQL は、入力ポイントの geohash 表現を返します。

```
SELECT ST_GeoHash(ST_GeomFromText('POINT(45 -45)'), 25) AS geohash;
```

```
          geohash
---------------------------
 m00000000000000000000gzz
```

以下の SQL は、入力ポイントが空であることから null を返します。

```
SELECT ST_GeoHash(ST_GeomFromText('POINT EMPTY'), 10) IS NULL AS result;
```

```
 result
---------
 true
```

ST_GeogFromText

ST_GeogFromText は、入力ジオメトリの WKT (Well-Known Text) もしくは EWKT (拡張 Well-Known Text) 表現を基に、ジオメトリオブジェクトを構築します。

構文

```
ST_GeogFromText(wkt_string)
```

引数

wkt_string

ジオグラフィの WKT もしくは EWKT 表現であるデータ型 VARCHAR の値です。

戻り型

GEOGRAPHY

入力で指定された値に SRID 値が設定されている場合。SRID が指定されていない場合は、4326 がセットされます。

wkt_string が null の場合、null が返されます。

wkt_string が有効でない場合、エラーが返されます。

例

次の SQL は、SRID 値を持つジオグラフィオブジェクトからポリゴンを作成します。

```
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
 st_asewkt
-------------------------------------------------
 SRID=4324;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

次の SQL は、Geograpy オブジェクトからポリゴンを作成します。SRID の値は 4326 にセットされます。

```
SELECT ST_AsEWKT(ST_GeogFromText('POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))'));
```

```
 st_asewkt
------------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,10 10,1 0,0 0))
```

ST_GeogFromWKB

ST_GeogFromWKB は、入力ジオグラフィの 16 進数による WKB (Well-Known Binary) 表現からジオグラフィオブジェクトを作成します。

構文

```
ST_GeogFromWKB(wkb_string)
```

引数

wkb_string

ジオグラフィの 16 進数 WKB 表現であるデータ型 `VARCHAR` の値。

戻り型

`GEOGRAPHY`

SRID 値が指定されている場合、指定された値に設定されます。SRID が指定されていない場合、`4326` に設定されます。

wkb_string が null の場合、null が返されます。

wkb_string が有効でない場合、エラーが返されます。

例

次の SQL は、16 進数 WKB 値からジオグラフィを構築します。

```
SELECT
 ST_AsEWKT(ST_GeogFromWKB('01030000000100000005000000000000000000000000000000000000000000000
```

```
 st_asewkt
-------------------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeometryN

ST_GeometryN は、次のように、入力ジオメトリの入力インデックスで指定されているジオメトリを返します。

- 入力がポイント、ラインストリング、またはポリゴンの場合、インデックスが 1 の場合はジオメトリがそのまま返され、インデックスが 1 以外の場合は null が返されます。
- 入力がマルチポイント、マルチラインストリング、マルチポリゴン、またはジオメトリコレクションの場合、ポイント、ラインストリング、ポリゴン、またはジオメトリコレクションは入力インデックスでの指定に従って返されます。

このインデックスは 1 から始まります。結果の空間参照系識別子 (SRID) は、入力ジオメトリのものと同じです。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_GeometryN(geom, index)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

index

1 から始まるインデックスの位置を表すデータ型 `INTEGER` の値。

戻り型

`GEOMETRY`

geom または index が null の場合、null が返されます。

index が範囲外の場合、エラーが返されます。

例

次の SQL は、ジオメトリコレクションに含まれているジオメトリを返します。

```
WITH tmp1(idx) AS (SELECT 1 UNION SELECT 2),
tmp2(g) AS (SELECT ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))'))
SELECT idx, ST_AsEWKT(ST_GeometryN(g, idx)) FROM tmp1, tmp2 ORDER BY idx;
```

```

 idx |          st_asewkt
-----+------------------------------
   1 | POLYGON((0 0,10 0,0 10,0 0))
   2 | LINESTRING(20 10,20 0,10 0)
```

ST_GeometryType

ST_GeometryType は、入力ジオメトリのサブタイプを文字列として返します。

入力のジオメトリが 3DM、3DZ、4D の場合、ST_GeometryType は、2D ジオメトリ入力の場合と同じ結果を返します。

構文

```
ST_GeometryType(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

geom のサブタイプを表す `VARCHAR`。

geom が null の場合、null が返されます。

返される値は次のとおりです。

返される文字列値	ジオメトリのサブタイプ
ST_Point	geom が POINT サブタイプの場合に返されます
ST_LineString	geom が LINESTRING サブタイプの場合に返されます
ST_Polygon	geom が POLYGON サブタイプの場合に返されます
ST_MultiPoint	geom が MULTIPOINT サブタイプの場合に返されます
ST_MultiLineString	geom が MULTILINESTRING サブタイプの場合に返されます
ST_MultiPolygon	geom が MULTIPOLYGON サブタイプの場合に返されます
ST_GeometryCollection	geom が GEOMETRYCOLLECTION サブタイプの場合に返されます

例

次の SQL は、入力 LINESTRING ジオメトリのサブタイプを返します。

```
SELECT ST_GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```


st_geometrytype
-------------
 ST_LineString
```

ST_GeomFromEWKB

ST_GeomFromEWKB は、入力ジオメトリの EWKB (Extended Well-Known Binary) 表現からジオメトリオブジェクトを作成します。

ST_GeomFromEWKB は、WKB および EWKB の 16 進形式で記述された 3DZ、3DM、および 4D ジオメトリを受け入れます。

構文

```
ST_GeomFromEWKB(ewkb_string)
```

引数

ewkb_string

ジオメトリ の 16 進数の EWKB 表現であるデータ型 VARCHAR の値。

戻り型

GEOMETRY

ewkb_string が null の場合、null が返されます。

ewkb_string が有効でない場合、エラーが返されます。

例

次の SQL は、EWKB 値からポリゴンを作成し、ポリゴンの EWKT 表現を返します。

```
SELECT
 ST_AsEWKT(ST_GeomFromEWKB('0103000020E61000000100000005000000000000000000000000000000000000000
```

```

 st_asewkt
--------------------------------
 SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeomFromEWKT

ST_GeomFromEWKT は、入力ジオメトリの拡張 well-known text (EWKT) 表現からジオメトリオブジェクトを構築します。

ST_GeomFromEWKT は、3DZ、3DM、および 4D を受け入れます。この場合、各ジオメトリタイプの先頭には Z、M、または ZM がプレフィックスされます。

構文

```
ST_GeomFromEWKT(ewkt_string)
```

引数

ewkt_string

データ型 VARCHAR の値、または VARCHAR 型に評価される式、つまりジオメトリの EWKT 表現。

WKT キーワード EMPTY を使用すると、空のポイント、空のポイントを持つマルチポイント、あるいは空のポイントを持つジオメトリコレクションを指定することができます。次の例では、空のポイントを 1 つ作成します。

```
ST_GeomFromEWKT('SRID=4326;POINT EMPTY');
```

戻り型

GEOMETRY

ewkt_string が null の場合、null が返されます。

ewkt_string が有効でない場合、エラーが返されます。

例

次の SQL は、EWKT 値からマルチライン文字列を作成し、ジオメトリを返します。また、ジオメトリの ST_AsEWKT 結果を返します。

```
SELECT ST_GeomFromEWKT('SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))')
 as geom, ST_AsEWKT(geom);
```

```
                                                              geom
                                              |                      st_asewkt
-----------------------------------------------------------------------------------------------------
+-------------------------------------------------------------
```

```
0105000020E61000000300000001020000000200000000000000000000000000F03F0000000000000000000000000000F03F00
| SRID=4326;MULTILINESTRING((1 0,1 0),(2 0,3 0),(4 0,5 0,6 0))
```

ST_GeomFromGeoHash

ST_GeomFromGeoHash は、入力ジオメトリの geohash 表現からジオメトリオブジェクトを作成します。ST_GeomFromGeoHash は、空間参照識別子 (SRID) がゼロ (0) の 2 次元 (2D) ジオメトリを返します。geohash フォーマットに関する詳細は、ウィキペディアの「Geohash」を参照してください。

構文

```
ST_GeomFromGeoHash(geohash_string)
```

```
ST_GeomFromGeoHash(geohash_string, precision)
```

引数

geohash_string

データ型 VARCHAR の値、または VARCHAR 型に評価される式、つまりジオメトリの geohash 表現。

precision

geohash の精度を表すデータ型 INTEGER の値。値は、精度として使用される geohash の文字数です。値が指定されていない場合、ゼロより小さい場合、または geohash_string の長さを越えている場合、geohash_string の長さが使用されます。

戻り型

GEOMETRY

geohash_string が null の場合、null が返されます。

geohash_string が有効でない場合、エラーが返されます。

例

次の SQL は、高精度のポリゴンを返します。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816
 36.114646,-115.172816 36.114646))
```

次の SQL は、高精度のポイントを返します。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz00'));
```

```
 st_asewkt
-----------------------
 POINT(-115.172816 36.114646)
```

次の SQL は、低精度のポリゴンを返します。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qq'));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

次の SQL は、精度 3 のポリゴンを返します。

```
SELECT ST_AsText(ST_GeomFromGeoHash('9qqj7nmxncgyy4d0dbxqz0', 3));
```

```
 st_asewkt
-----------------------
 POLYGON((-115.3125 35.15625,-115.3125 36.5625,-113.90625 36.5625,-113.90625
 35.15625,-115.3125 35.15625))
```

ST_GeomFromGeoJSON

ST_GeomFromGeoJSON は、入力ジオメトリの GeoJSON 表現からジオメトリオブジェクトを作成します。GeoJSON フォーマットに関する詳細は、ウィキペディアの GeoJSON を参照してください。

3 つ以上の座標を持つ点が少なくとも 1 つ存在する場合、結果のジオメトリは 3DZ になります。ここで、座標が 2 つしかない点の Z コンポーネントはゼロになります。入力 GeoJSON のすべてのポイントに 2 つの座標が含まれているか、空の場合、ST_GeomFromGeoJSON は 2D ジオメトリを返します。返されたジオメトリには、必ず空間リファレンスシステム識別子 (SRID) 値 4326 が含まれます。

構文

```
ST_GeomFromGeoJSON(geojson_string)
```

引数

geojson_string

データ型 VARCHAR の値、または VARCHAR 型に評価される式、つまりジオメトリの GeoJSON 表現。

戻り型

GEOMETRY

geojson_string が null の場合、null が返されます。

geojson_string が有効でない場合、エラーが返されます。

例

次の SQL は、入力 GeoJSON で表される 2D ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Point","coordinates":[1,2]}'));
```

```
 st_asewkt
-----------------------
 SRID=4326;POINT(1 2)
```

次の SQL は、入力 GeoJSON で表される 3DZ ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"LineString","coordinates":[[1,2,3],
[4,5,6],[7,8,9]]}'));
```

```
 st_asewkt
------------------------------------------
 SRID=4326;LINESTRING Z (1 2 3,4 5 6,7 8 9)
```

次の SQL は、入力 GeoJSON で 1 つの点だけが 3 つの座標を持ち、他のすべてのポイントが 2 つの座標を持つ場合に 3DZ ジオメトリを返します。

```
SELECT ST_AsEWKT(ST_GeomFromGeoJSON('{"type":"Polygon","coordinates":[[[0, 0],[0, 1,
 8],[1, 0],[0, 0]]]}'));
```

```
 st_asewkt
------------------------------------------------
 SRID=4326;POLYGON Z ((0 0 0,0 1 8,1 0 0,0 0 0))
```

ST_GeomFromGeoSquare

ST_GeomFromGeoSquare は、入力された geosquare 値で表されるエリアをカバーするジオメトリを返します。返されるジオメトリは常に 2 次元です。geosquare 値を計算するには、「ST_GeoSquare」を参照してください。

構文

```
ST_GeomFromGeoSquare(geosquare)
```

```
ST_GeomFromGeoSquare(geosquare, max_depth)
```

引数

geosquare

データ型の値 BIGINT または、目的の正方形になるように最初のドメインで作成される区分の順序を表す geosquare 値である BIGINT 型に評価される式。この値は ST_GeoSquare で計算されます。

max_depth

初期ドメインで行われたドメインの分割の最大数を表すデータ型 INTEGER の値。この値は、1 以上にする必要があります。

戻り型

GEOMETRY

geosquare が有効でない場合、関数はエラーを返します。

入力 max_depth が範囲内にない場合、関数はエラーを返します。

例

次の SQL は、geosquare 値からジオメトリを返します。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852));
```

```
 st_astext
--------------------------------------------------------------------------------------------------------------------------------------
 POLYGON((13.359375 52.3828125,13.359375 52.734375,13.7109375 52.734375,13.7109375 
 52.3828125,13.359375 52.3828125))
```

次の SQL は、geosquare 値と 3 の最大深度からジオメトリを返します。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(797852, 3));
```

```
 st_astext
--------------------------------------
 POLYGON((0 45,0 90,45 90,45 45,0 45))
```

次の SQL は、最初に x 座標を経度、y 座標を緯度 (-122.3、47.6) として指定して、シアトルの geosquare 値を計算します。次に、geosquare のポリゴンを返します。出力は 2 次元のジオメトリですが、これを使用して経度と緯度の空間データを計算できます。

```
SELECT ST_AsText(ST_GeomFromGeoSquare(ST_GeoSquare(ST_Point(-122.3, 47.6))));
```

```
 st_astext
-------------------------------------------------------------------------------------------------
POLYGON((-122.335167014971 47.6080129947513,-122.335167014971
 47.6080130785704,-122.335166931152 47.6080130785704,-122.335166931152
 47.6080129947513,-122.335167014971 47.6080129947513))
```

ST_GeomFromText

ST_GeomFromText は、入力ジオメトリの WKT (Well-Known Text) 表現からジオメトリオブジェクトを作成します。

ST_GeomFromText は、(ジオメトリタイプに Z、M、または ZM がプレフィックスされた) 3DZ、3DM、および 4D を受け入れます。

構文

```
ST_GeomFromText(wkt_string)
```

```
ST_GeomFromText(wkt_string, srid)
```

引数

wkt_string

ジオメトリの WKT 表現であるデータ型 `VARCHAR` の値。

WKT キーワード `EMPTY` を使用すると、空のポイント、空のポイントを持つマルチポイント、あるいは空のポイントを持つジオメトリコレクションを指定することができます。次の例では、空のポイントと空でないポイントを、それぞれ1 つ持つマルチポイントを作成します。

```
ST_GeomFromEWKT('MULTIPOINT(1 0,EMPTY)');
```

srid

空間リファレンス識別子 (SRID) であるデータ型 `INTEGER` の値。SRID 値が指定されている場合、返されたジオメトリはこの SRID 値を持ちます。指定されていない場合は、返されるジオメトリの SRID 値は 0 に設定されます。

戻り型

`GEOMETRY`

wkt_string または srid が null の場合、null が返されます。

srid が負数の場合、null が返されます。

wkt_string が有効でない場合、エラーが返されます。

srid が有効でない場合、エラーが返されます。

例

次の SQL は、WKT 表現と SRID 値からジオメトリオブジェクトを作成します。

```
SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326);
```

```
st_geomfromtext
--------------------------------
0103000020E610000001000000050000000000000000000000000000000000000000000000000000000000000000000F
```

ST_GeomFromWKB

ST_GeomFromWKB は、入力ジオメトリの 16 進数の WKB (Well-Known Binary) 表現からジオメトリオブジェクトを作成します。

ST_GeomFromWKB は、WKB の 16 進形式で記述された 3DZ、3DM、および 4D ジオメトリを受け入れます。

構文

```
ST_GeomFromWKB(wkb_string)
```

```
ST_GeomFromWKB(wkb_string, srid)
```

引数

wkb_string

ジオメトリの 16 進数の WKB 表現であるデータ型 `VARCHAR` の値。

srid

空間リファレンス識別子 (SRID) であるデータ型 `INTEGER` の値。SRID 値が指定されている場合、返されたジオメトリはこの SRID 値を持ちます。そうでない場合は、返されたジオメトリの SRID 値は 0 に設定されます。

戻り型

`GEOMETRY`

wkb_string または srid が null の場合、null が返されます。

srid が負数の場合、null が返されます。

wkb_string が有効でない場合、エラーが返されます。

srid が有効でない場合、エラーが返されます。

例

次の SQL は、WKB 値からポリゴンを作成し、ポリゴンの WKT 表現を返します。

```
SELECT
 ST_AsText(ST_GeomFromWKB('010300000001000000050000000000000000000000000000000000000000000000000
```

```
 st_astext
-------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
```

ST_GeoSquare

ST_GeoSquare は、ドメイン ([-180, 180]、[-90, 90]) を、指定された深さまで geosquare と呼ばれる等しい正方形の領域に再帰的に分割します。分割は、指定されたポイントの位置に基づいて行われます。ポイントを含む geosquare の 1 つが、最大深度に達するまで各ステップで分割されます。この geosquare の選択は安定しています。つまり、関数の結果は入力引数のみに依存します。この関数は、ポイントが位置する最終的な geosquare を識別する固有の値を返します。

ST_GeoSquare は、x 座標が経度を表し、y 座標が緯度を表すポイントを受け入れます。経度と緯度はそれぞれ [-180, 180] と [-90, 90] に制限されます。ST_GeoSquare の出力は、ST_GeomFromGeoSquare 関数への入力として使用できます。

地球の赤道円弧の周囲は 360° あり、2 つの半球 (東と西) に分かれており、それぞれの半球には 0° の経線 (子午線) から 180° の子午線があります。慣例により、デカルト平面の x 軸に投影すると、東経は「+」(正) 座標になり、西経はデカルト平面の x 軸に投影すると「-」(負) 座標になります。地球の赤道円周 0° の北と南には 90° の緯度線があり、それぞれが地球の赤道円周 0° と平行です。慣例により、北緯線はデカルト平面に投影すると「+」(正) の y 軸と交差し、南緯線はデカルト平面に投影すると「-」(負) の y 軸と交差します。経線と緯線が交わることによって形成される球状グリッドは、標準の正と負の x 座標、およびデカルト平面上の正と負の y 座標により、デカルト平面に投影されたグリッドに変換されます。

ST_GeoSquare の目的は、近いポイントに同じコード値でタグ付けする、またはマークを付けることです。同じ geosquare にあるポイントには、同じコード値が割り当てられます。geosquare は、地理座標 (緯度と経度) を整数にエンコードするために使用されます。大きな領域をグリッドに分割して、さまざまな解像度のマップ上のエリアを表します。geosquare は、空間インデックス作成、空間ビニング、近接検索、位置検索、および固有の場所識別子の作成に使用できます。ST_GeoHash 関数は、領域をグリッドに分割するプロセスと似たプロセスに従いますが、エンコーディングが異なります。

構文

```
ST_GeoSquare(geom)
```

```
ST_GeoSquare(geom, max_depth)
```

引数

geom

データ型 GEOMETRY のポイント値またはポイントのサブタイプに評価される式。ポイントの x 座標 (経度) は -180～180 の範囲内である必要があります。ポイントの y 座標 (緯度) は -90～90 の範囲内である必要があります。

max_depth

データ型 INTEGER の値。ポイントを含むドメインが再帰的に分割される最大回数。値は 1～32 の整数にする必要があります。デフォルトは 32 です。分割の実際の最終数は、指定した max_depth 以下です。

戻り型

BIGINT

この関数は、入力ポイントが位置する最終的な geosquare を識別する固有の値を返します。

入力 geom がポイントではない場合、関数はエラーを返します。

入力ポイントが空の場合、戻り値は ST_GeomFromGeoSquare 関数への有効な入力ではありません。ST_IsEmpty 関数を使用すると、ポイントが空の ST_GeoSquare を呼び出すのを防ぐことができます。

入力ポイントが範囲内にない場合、関数はエラーを返します。

入力 max_depth が範囲外の場合、関数はエラーを返します。

例

次の SQL は、入力ポイントから geosquare を返します。

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5));
```

```
  st_geosquare
-----------------------
 -4410772491521635895
```

次の SQL は、最大深度が 10 の入力ポイントから geosquare を返します。

```
SELECT ST_GeoSquare(ST_Point(13.5, 52.5), 10);
```

```
 st_geosquare
--------------
 797852
```

ST_InteriorRingN

ST_InteriorRingN は、インデックス位置にある入力ポリゴンの内部リングに対応する閉じたライン文字列を返します。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_InteriorRingN(geom, index)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

index

1 から始まるインデックスのリング位置を表すデータ型 `INTEGER` の値。

戻り型

`GEOMETRY`サブタイプ の `LINESTRING`。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom または index が null の場合、null が返されます。

index が範囲外の場合、null が返されます。

geom がポリゴンでない場合、null が返されます。

geom が空のポリゴンである場合、null が返されます。

例

次の SQL は、ポリゴンの 2 番目のリングを閉じたライン文字列として返します。

```
SELECT ST_AsEWKT(ST_InteriorRingN(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17
 7,17 10,18 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12
 14,15 14,13 11,12 14))'),2));
```

```


st_asewkt
-----------
 LINESTRING(12 14,15 14,13 11,12 14)
```

ST_Intersects

ST_Intersects は、2 つの入力ジオメトリの 2D 射影が少なくとも 1 つの共通したポイントを持つ場合、true を返します。

構文

```
ST_Intersects(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴンと交差するかどうかを確認します。

```
SELECT ST_Intersects(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 5,5 5,5
 2,2 2))'), ST_GeomFromText('MULTIPOINT((4 4),(6 6))'));
```

```
st_intersects
-------------
 true
```

ST_Intersection

ST_Intersection は、2 つのジオメトリのポイントセットの交差か所を表すジオメトリを返します。つまり、2 つの入力ジオメトリ間で共有されている部分を返します。

構文

```
ST_Intersection(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`GEOMETRY`

geom1 と geom2 が空間を共有しない (互いに離れている) 場合、空のジオメトリが返されます。

geom1 もしくは geom2 が空の場合、空のジオメトリが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

geom1 もしくは geom2 が 二次元の (2D) ジオメトリでない場合、エラーが返されます。

例

次の SQL は、2 つの入力ジオメトリの交差を表す、空でないジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('polygon((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('polygon((0 0,10 0,0 10,0 0))')));
```

```


        st_asewkt
-------------------------
 POLYGON((0 0,0 10,5 5,0 0))
```

次の SQL は、離れている (交差しない) ジオメトリを入力に渡した場合、空のジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Intersection(ST_GeomFromText('linestring(0 100,0 0)'),
 ST_GeomFromText('polygon((1 0,10 0,1 10,1 0))')));
```

```


    st_asewkt
------------------
 LINESTRING EMPTY
```

ST_IsPolygonCCW

ST_IsPolygonCCW は、入力ポリゴンまたはマルチポリゴンの 2D 射影が、反時計回りの場合に true を返します。入力ジオメトリがポイント、ライン文字列、マルチポイント、またはマルチライン文字列の場合、true が返されます。ジオメトリコレクションの場合、ST_IsPolygonCCW は、コレクション内のすべてのジオメトリが反時計回りの場合、true を返します。

構文

```
ST_IsPolygonCCW(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンが反時計回りであるかどうかを確認します。

```
SELECT ST_IsPolygonCCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```
 st_ispolygonccw
----------
 true
```

ST_IsPolygonCW

ST_IsPolygonCW は、入力ポリゴンまたはマルチポリゴンの 2D 射影が、時計回りの場合に true を返します。入力ジオメトリがポイント、ライン文字列、マルチポイント、またはマルチライン文字列の場合、true が返されます。ジオメトリコレクションの場合、ST_IsPolygonCW は、コレクション内のすべてのジオメトリが時計回りの場合、true を返します。

構文

```
ST_IsPolygonCW(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

BOOLEAN

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンが時計回りであるかどうかを確認します。

```
SELECT ST_IsPolygonCW(ST_GeomFromText('POLYGON((7 9,8 7,11 6,15 8,16 6,17 7,17 10,18
 12,17 14,15 15,11 15,10 13,9 12,7 9),(9 9,10 10,11 11,11 10,10 8,9 9),(12 14,15 14,13
 11,12 14))'));
```

```

 st_ispolygonccw
----------
 true
```

ST_IsClosed

ST_IsClosed は、入力ジオメトリの 2D 射影がクローズされている場合に true を返します。次のルールによって、クローズされたジオメトリが定義されます。

- 入力ジオメトリがポイントまたはマルチポイントである。
- 入力ジオメトリが LINESTRING で、LINESTRING のスタートポイントとエンドポイントが一致する。
- 入力ジオメトリが空ではない MULTILINESTRING で、すべての LINESTRING がクローズされている。
- 入力ジオメトリが空でないポリゴンで、すべてのポリゴンのリングが空でなく、すべてのリングのスタートポイントとエンドポイントが一致する。
- 入力ジオメトリが空でないマルチポリゴンで、すべてのポリゴンがクローズされている。
- 入力ジオメトリが空でないジオメトリコレクションで、すべてのコンポーネントがクローズされている。

構文

```
ST_IsClosed(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom が空のポイントである場合、false が返されます。

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンがクローズされているかどうかを確認します。

```
SELECT ST_IsClosed(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```


st_isclosed
-----------
 true
```

ST_IsCollection

ST_IsCollection は、入力ジオメトリが `GEOMETRYCOLLECTION`、`MULTIPOINT`、`MULTILINESTRING`、`MULTIPOLYGON` のいずれかのサブタイプである場合、true を返します。

構文

```
ST_IsCollection(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンがコレクションであるかどうかを確認します。

```
SELECT ST_IsCollection(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```

st_iscollection
-----------
 false
```

ST_IsEmpty

ST_IsEmpty は、入力ジオメトリが空の場合、true を返します。ジオメトリに空でない点が少なくとも 1 つ含まれていれば、そのジオメトリは空ではありません。

ST_IsEmpty は、入力ジオメトリに空でないポイントが少なくとも 1 つ含まれる場合、true を返します。

構文

```
ST_IsEmpty(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom が null の場合、null が返されます。

例

次の SQL は、ポリゴンが空であるかどうかを確認します。

```
SELECT ST_IsEmpty(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'));
```

```
st_isempty
-----------
 false
```

ST_IsRing

ST_IsRing は、入力ライン文字列がリングの場合、true を返します。閉じており、かつ単純なライン文字列はリングです。

構文

```
ST_IsRing(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。ジオメトリは `LINESTRING` である必要があります。

戻り型

`BOOLEAN`

geom が `LINESTRING` でない場合、エラーが返されます。

例

次の SQL は、指定されたライン文字列がリングであるかどうかを確認します。

```
SELECT ST_IsRing(ST_GeomFromText('linestring(0 0, 1 1, 1 2, 0 0)'));
```

```
st_isring
-----------
 true
```

ST_IsSimple

ST_IsSimple は、入力ジオメトリの 2D 射影が単純な場合、true を返します。単純なジオメトリ定義の詳細については、幾何学的な単純度 を参照してください。

構文

```
ST_IsSimple(geom)
```

引数

geom

　データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom が null の場合、null が返されます。

例

次の SQL は、指定されたライン文字列がシンプルであるかどうかを確認します。この例では、自己交差があるため、単純ではありません。

```
SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(0 0,10 0,5 5,5 -5)'));
```

```
 st_issimple
-----------
 false
```

ST_IsValid

ST_IsValid は、入力ジオメトリ 2D 射影が有効である場合、true を返します。有効なジオメトリ定義の詳細については、幾何学的妥当性 を参照してください。

構文

```
ST_IsValid(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom が null の場合、null が返されます。

例

次の SQL は、指定されたポリゴンが有効であるかどうかを確認します。この例では、ポリゴンの内部が単純に接続されていないため、ポリゴンは無効です。

```
SELECT ST_IsValid(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 0,10 5,5 10,0
 5,5 0))'));
```

```

 st_isvalid
-----------
 false
```

ST_Length

線形ジオメトリの場合、ST_Length は 2D 射影のデカルト座標系での長さを返します。長さの単位は、入力ジオメトリの座標を表す単位と同じです。この関数は、ポイント、マルチポイント、および面積ジオメトリに対して 0 を返します。入力がジオメトリコレクションの場合、この関数はコレクションのジオメトリの長さの合計を返します。

ジオグラフィに対して ST_Length は、SRID で決定される回転楕円体上で計算され与えられた、線形ジオグラフィの 2D 射影について測地座標系での長さが返します。長さの単位はメートルです。この関数は、ポイント、マルチポイント、および面積ジオグラフィに対して 0 を返します。入力がジ

オメトリコレクションの場合、この関数はコレクション内にあるジオグラフィの長さの合計を返します。

構文

```
ST_Length(geo)
```

引数

geo

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。

戻り型

DOUBLE PRECISION

geo が null の場合、null が返されます。

SRID 値が見つからない場合、エラーが返されます。

例

次のSQLは、マルチラインストリングのデカルト座標系での長さを返します。

```
SELECT ST_Length(ST_GeomFromText('MULTILINESTRING((0 0,10 0,0 10),(10 0,20 0,20
 10))'));
```

```
st_length
--------------------------------
  44.142135623731
```

次の SQL は、ジオグラフィ内にあるライン文字列の長さを返します。

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;LINESTRING(5 0,6 0,4 0)'));
```

```
 st_length
------------------
```

```
 333958.472379804
```

次の SQL は、ジオグラフィ内にあるポイントの長さを返します。

```
SELECT ST_Length(ST_GeogFromText('SRID=4326;POINT(4 5)'));
```

```
 st_length
-----------
 0
```

ST_LengthSphere

ST_LengthSphere は、線形ジオメトリの長さをメートル単位で返します。ポイントジオメトリ、マルチポイントジオメトリ、および面積ジオメトリの場合、ST_LengthSphere は 0 を返します。ジオメトリコレクションの場合、ST_LengthSphere は、コレクション内の線形ジオメトリの全長をメートル単位で返します。

ST_LengthSphere は、入力ジオメトリの各ポイントの座標を経度および緯度 (度単位) として解釈します。3DZ、3DM、または 4D ジオメトリの場合、最初の 2 つの座標のみが使用されます。

構文

```
ST_LengthSphere(geom)
```

引数

geom

  データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`DOUBLE PRECISION` の長さ (メートル単位)。長さの計算は、地球の世界測地系 (WGS) 84 楕円体モデルの地球平均半径を半径とする地球の球形モデルに基づいています。

geom が null の場合、null が返されます。

例

次の SQL 例では、ライン文字列の長さをメートル単位で計算します。

```
SELECT ST_LengthSphere(ST_GeomFromText('LINESTRING(10 10,45 45)'));
```

```
 st_lengthsphere
------------------
 5127736.08292556
```

ST_Length2D

ST_Length2D は、ST_Length のエイリアスです。詳細については、「ST_Length」を参照してください。

ST_LineFromMultiPoint

ST_LineFromMultiPoint は、入力マルチポイントジオメトリからラインストリングを返します。ポイントの順序は保持されます。返されるジオメトリの空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_LineFromMultiPoint(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `MULTIPOINT` である必要があります。

戻り型

`GEOMETRY`

geom が null の場合、null が返されます。

geom が空の場合、空の `LINESTRING` が返されます。

geom に空のポイントが含まれている場合、これらの空のポイントは無視されます。

geom が `MULTIPOINT` でない場合、エラーが返されます。

例

次の SQL は、マルチポイントからラインストリングを作成します。

```
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromText('MULTIPOINT(0 0,10 0,10 10,5 5,0 5)',4326)));
```

```
 st_asewkt
--------------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5,0 5)
```

ST_LineInterpolatePoint

ST_LineInterpolatePoint は、ラインの開始点からの分数距離にあるラインに沿ったポイントを返します。

ポイントの等価性を判断するために、ST_LineInterpolatePoint は、入力ジオメトリの 2D 射影を処理します。入力ジオメトリが空の場合、入力のコピーが同じディメンションで返されます。3DZ、3DM、4D ジオメトリの場合、`z` または `m` 座標は、ポイントが置かれているセグメントの `z` または `m` 座標の平均になります。

構文

```
ST_LineInterpolatePoint(geom, fraction)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` です。

分数

ラインのライン文字列に沿った点の位置を表すデータ型 `DOUBLE PRECISION` の値。値は、0～1 の範囲の小数部です。

戻り型

`GEOMETRY`サブタイプ の `POINT`。

geom または fraction が null の場合、null が返されます。

geom が空の場合、空のポイントが返されます。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

fraction が範囲外の場合、エラーが返されます。

geom が LINESTRING でない場合、エラーが返されます。

例

次の SQL は、ライン文字列に沿ってポイントを返します。

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.50));
```

```


st_asewkt
-----------
 POINT(5 5)
```

次の SQL は、ライン文字列に沿って 90% のポイントを返します。

```
SELECT ST_AsEWKT(ST_LineInterpolatePoint(ST_GeomFromText('LINESTRING(0 0, 5 5, 7 7, 10
 10)'), 0.90));
```

```


st_asewkt
-----------
 POINT(9 9)
```

ST_M

ST_M は、入力ポイントの m 座標を返します。

構文

```
ST_M(point)
```

引数

point

データ型 GEOMETRY の POINT 値。

戻り型

m 座標系の DOUBLE PRECISION 値。

point が null の場合、null が返されます。

point が 2D または 3DZ ポイントの場合、null が返されます。

point が空のポイントの場合、null が返されます。

point が POINT でない場合、エラーが返されます。

例

次の SQL は、ポイントの m 座標を、3DM ジオメトリで返します。

```
SELECT ST_M(ST_GeomFromEWKT('POINT M (1 2 3)'));
```

```


st_m
-----------
 3
```

次の SQL は、1 つのポイントの m 座標を、4D ジオメトリで返します。

```
SELECT ST_M(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```


st_m
-----------
 4
```

ST_MakeEnvelope

ST_MakeEnvelope は以下のようにジオメトリを返します。

- 入力座標でポイントを指定すると、返されるジオメトリはポイントになります。
- 入力座標がラインを指定している場合、返されるジオメトリはライン文字列です。
- それ以外の場合、返されるジオメトリはポリゴンであり、入力座標はボックスの左下隅と右上隅を指定します。

指定されている場合、返されたジオメトリの空間リファレンス識別子 (SRID) 値が、入力 SRID 値に設定されます。

構文

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax)
```

```
ST_MakeEnvelope(xmin, ymin, xmax, ymax, srid)
```

引数

xmin

データ型 `DOUBLE PRECISION` の値。この値は、ボックスの左下隅の最初の座標です。

ymin

データ型 `DOUBLE PRECISION` の値。この値は、ボックスの左下隅の 2 番目の座標です。

xmax

データ型 `DOUBLE PRECISION` の値。この値は、ボックスの右上隅の最初の座標です。

ymax

データ型 `DOUBLE PRECISION` の値。この値は、ボックスの右上隅の 2 番目の座標です。

srid

空間リファレンス識別子 (SRID) を表すデータ型 `INTEGER` の値。SRID 値が指定されていない場合、0 に設定されます。

戻り型

サブタイプ `POINT`、`LINESTRING`、または `POLYGON` の `GEOMETRY`。

返されたジオメトリの SRID は、`srid` に設定され、`srid` が設定されていない場合は 0 に設定されます。

xmin、ymin、xmax、ymax、または srid が null の場合は、null が返されます。

srid が負数の場合、エラーが返されます。

例

次の SQL は、4 つの入力座標値によって定義されたエンベロープを表すポリゴンを返します。

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7));
```

```
 st_astext
---------------
 POLYGON((2 4,2 7,5 7,5 4,2 4))
```

次の SQL は、4 つの入力座標値と SRID 値によって定義されたエンベロープを表すポリゴンを返します。

```
SELECT ST_AsEWKT(ST_MakeEnvelope(2,4,5,7,4326));
```

```
 st_astext
----------------------------------
 SRID=4326;POLYGON((2 4,2 7,5 7,5 4,2 4))
```

ST_MakeLine

ST_MakeLine は、入力ジオメトリから LINESTRING を作成します。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。両方の入力ジオメトリは、同じディメンションである必要があります。

構文

```
ST_MakeLine(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT`、`LINESTRING`、または `MULTIPOINT` である必要があります。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `POINT`、`LINESTRING`、または `MULTIPOINT` である必要があります。

戻り型

`GEOMETRY`サブタイプ の `LINESTRING`。

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 が空のポイントであるか、空のポイントが含まれている場合には、これらの空のポイントは無視されます。

geom1 および geom2 が空の場合、空の `LINESTRING` が返されます。

返されたジオメトリの空間リファレンス識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom1 および geom2 の SRID 値が異なる場合、エラーが返されます。

geom1 または geom2 が `POINT`、`LINESTRING`、または `MULTIPOINT` ではない場合は、エラーが返されます。

geom1 と geom2 のディメンションが異なる場合、エラーが返されます。

例

次の SQL は、2 つの入力 LINESTRING から LINESTRING を作成します。

```
SELECT ST_MakeLine(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'), ST_GeomFromText('LINESTRING(88.29 39.07,88.42 39.26,88.27
 39.31,88.29 39.07)'));
```

```
st_makeline
-----------
```

```
010200000008000000C3F5285C8F52534052B81E85EB113D407B14AE47E15A5340C3F5285C8F423D40E17A14AE4751
```

ST_MakePoint

ST_MakePoint は、座標値が入力値であるポイントジオメトリを返します。

構文

```
ST_MakePoint(x, y)
```

```
ST_MakePoint(x, y, z)
```

```
ST_MakePoint(x, y, z, m)
```

引数

x

最初の座標を表すデータ型 `DOUBLE PRECISION` の値。

y

2 番目の座標を表すデータ型 `DOUBLE PRECISION` の値。

z

3 番目の座標を表すデータ型 `DOUBLE PRECISION` の値。

m

4 番目の座標を表すデータ型 `DOUBLE PRECISION` の値。

戻り型

`GEOMETRY`サブタイプ の `POINT`。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が 0 に設定されます。

x、y、z、または m が null の場合は、null が返されます。

例

次の SQL は、提供された座標を持つサブタイプ `POINT` の `GEOMETRY` タイプを返します。

```
SELECT ST_AsText(ST_MakePoint(1,3));
```

```
st_astext
-----------
 POINT(1 3)
```

次の SQL は、提供された座標を持つサブタイプ `POINT` の `GEOMETRY` タイプを返します。

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3));
```

```
st_asewkt
----------------
 POINT Z (1 2 3)
```

次の SQL は、提供された座標を持つサブタイプ `GEOMETRY` の `POINT` タイプを返します。

```
SELECT ST_AsEWKT(ST_MakePoint(1, 2, 3, 4));
```

```
st_asewkt
-------------------
 POINT ZM (1 2 3 4)
```

ST_MakePolygon

ST_MakePolygon には、ポリゴンを返す 2 つの変形があります。1 つは単一のジオメトリを取り、もう 1 つは 2 つのジオメトリを取ります。

- 最初の変形の入力は、出力ポリゴンの外リングを定義するライン文字列です。
- 2 番目の変形の入力は、ライン文字列とマルチライン文字列です。どちらも空であるか、閉じられています。

  出力ポリゴンの外部リングの境界は入力ライン文字列であり、ポリゴンの内部リングの境界は入力マルチライン文字列のライン文字列です。入力ライン文字列が空の場合、空のポリゴンが返されます。マルチライン文字列内の空のライン文字列は無視されます。結果ジオメトリの空間リファレンス識別子 (SRID) は、2 つの入力ジオメトリの共通 SRID です。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。外部リングと内部リングは同じディメンションでなければなりません。

構文

```
ST_MakePolygon(geom1)
```

```
ST_MakePolygon(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` である必要があります。linestring 値はクローズされているか、空である必要があります。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `MULTILINESTRING` である必要があります。

戻り型

`GEOMETRY`サブタイプ の `POLYGON`。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) は、入力の SRID と等しくなります。

geom1 または geom2 が null の場合、null が返されます。

geom1 が linestring でない場合、エラーが返されます。

geom2 がマルチライン文字列でない場合、エラーが返されます。

geom1 がクローズされていない場合、エラーが返されます。

geom1 が単一のポイントであるか、クローズされていない場合、エラーが返されます。

geom2 に、単一のポイントを持つ、または閉じられていないライン文字列が少なくとも 1 つ含まれている場合、エラーが返されます。

geom1 および geom2 の SRID 値が異なる場合、エラーが返されます。

geom1 と geom2 のディメンションが異なる場合、エラーが返されます。

例

次の SQL は、入力 LINESTRING からポリゴンを返します。

```
SELECT ST_AsText(ST_MakePolygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42
 29.26,77.27 29.31,77.29 29.07)')));
```

```

 st_astext
---------------
POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

次の SQL では、閉じたライン文字列と閉じたマルチライン文字列からポリゴンを作成します。ライン文字列は、ポリゴンの外部リングに使用されます。マルチライン文字列のライン文字列は、ポリゴンの内部リングに使用されます。

```
SELECT ST_AsEWKT(ST_MakePolygon(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,0 10,0 0)'),
 ST_GeomFromText('MULTILINESTRING((1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))')));
```

```

 st_astext
----------------------------------
POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 1,1 1),(3 3,3 4,4 3,3 3))
```

ST_MemSize

ST_MemSize は、入力ジオメトリによって使用されるメモリ容量 (バイト) を返します。このサイズは、Amazon Redshift でのジオメトリの内部的な表現に依存するため、内部的な表現が変更されると変わる場合があります。このサイズは、Amazon Redshift でのジオメトリオブジェクトの相対的なサイズを示すために使用できます。

構文

```
ST_MemSize(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

INTEGER は geom 固有のディメンションを表します。

geom が null の場合、null が返されます。

例

次の SQL は、ジオメトリコレクションのメモリサイズを返します。

```
SELECT ST_MemSize(ST_GeomFromText('GEOMETRYCOLLECTION(POLYGON((0 0,10 0,0 10,0
 0)),LINESTRING(20 10,20 0,10 0))'))::varchar + ' bytes';
```

```

 ?column?
-----------
 172 bytes
```

ST_MMax

ST_MMax は、入力ジオメトリの最大の m 座標を返します。

構文

```
ST_MMax(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

最大の m 座標の DOUBLE PRECISION 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

geom が 2D または 3DZ ジオメトリの場合、null が返されます。

例

次の SQL は、ライン文字列の最大の m 座標を、3DM ジオメトリで返します。

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```

st_mmax
-----------
  8
```

次の SQL は、ライン文字列の最大の m 座標を、4D ジオメトリで返します。

```
SELECT ST_MMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```

st_mmax
-----------
  11
```

ST_MMin

ST_MMin は、入力ジオメトリの最小の m 座標を返します。

構文

```
ST_MMin(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

最小の m 座標の `DOUBLE PRECISION` 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

geom が 2D または 3DZ ジオメトリの場合、null が返されます。

例

次の SQL は、ライン文字列の最小の m 座標を、3DM ジオメトリで返します。

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING M (0 1 2, 3 4 5, 6 7 8)'));
```

```


st_mmin
-----------
  2
```

次の SQL は、ライン文字列の最小の m 座標を、4D ジオメトリで返します。

```
SELECT ST_MMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```


st_mmin
-----------
  3
```

ST_Multi

ST_Multi は、ジオメトリを対応するマルチタイプに変換します。入力ジオメトリがすでにマルチタイプまたはジオメトリコレクションである場合は、そのコピーが返されます。入力ジオメトリがポイント、ライン文字列、またはポリゴンである場合、入力ジオメトリを含むマルチポイント、マルチライン文字列、またはマルチポリゴンがそれぞれ返されます。

構文

```
ST_Multi(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

サブタイプ MULTIPOINT、MULTILINESTRING、MULTIPOLYGON、または GEOMETRYCOLLECTION の GEOMETRY。

返されるジオメトリの空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。

geom が null の場合、null が返されます。

例

次の SQL は、入力マルチポイントからマルチポイントを返します。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('MULTIPOINT((1 2),(3 4))', 4326)));
```

```


    st_asewkt
------------------------------------
  SRID=4326;MULTIPOINT((1 2),(3 4))
```

次の SQL は、入力ポイントからマルチポイントを返します。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('POINT(1 2)', 4326)));
```

```


    st_asewkt
------------------------------------
  SRID=4326;MULTIPOINT((1 2))
```

次の SQL は、入力ジオメトリからジオメトリコレクションを返します。

```
SELECT ST_AsEWKT(ST_Multi(ST_GeomFromText('GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1
 2),(3 4)))', 4326)));
```

```
    st_asewkt
------------------------------------
  SRID=4326;GEOMETRYCOLLECTION(POINT(1 2),MULTIPOINT((1 2),(3 4)))
```

ST_NDims

ST_NDims は、ジオメトリの座標ディメンションを返します。ST_NDims では、ジオメトリのトポロジディメンションは考慮されません。代わりに、ジオメトリのディメンションに応じた定数値を返します。

構文

```
ST_NDims(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`INTEGER` は geom 固有のディメンションを表します。

geom が null の場合、null が返されます。

返される値は次のとおりです。

戻り値	入力ジオメトリのディメンション
2	2D
3	3DZ もしくは 3DM
4	4D

例

次の SQL は、2D のライン文字列における、ディメンションの数を返します。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING(0 0,1 1,2 2,0 0)'));
```

```
st_ndims
-------------
 2
```

次の SQL は、3DZ のライン文字列における、ディメンションの数を返します。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING Z(0 0 3,1 1 3,2 2 3,0 0 3)'));
```

```
st_ndims
-------------
 3
```

次の SQL は、3DM のライン文字列における、ディメンションの数を返します。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING M(0 0 4,1 1 4,2 2 4,0 0 4)'));
```

```
st_ndims
-------------
 3
```

次の SQL は、4D のライン文字列における、ディメンションの数を返します。

```
SELECT ST_NDims(ST_GeomFromText('LINESTRING ZM(0 0 3 4,1 1 3 4,2 2 3 4,0 0 3 4)'));
```

```
st_ndims
-------------
 4
```

ST_NPoints

ST_NPoints は、入力されたジオメトリもしくはジオグラフィ内の、空ではないポイントの数を返します。

構文

```
ST_NPoints(geo)
```

引数

geo

データ型 GEOMETRY または GEOGRAPHY の値、または GEOMETRY もしくは GEOGRAPHY 型として評価される式です。

戻り型

INTEGER

geo が空のポイントの場合、0 が返されます。

geo が null の場合、null が返されます。

例

次の SQL は、LINESTRING のポイントの数を返します。

```
SELECT ST_NPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```


st_npoints
-------------
 4
```

次の SQL は、ジオグラフィ内のライン文字列のポイント数を返します。

```
SELECT ST_NPoints(ST_GeogFromText('LINESTRING(110 40, 2 3, -10 80, -7 9)'));
```

```


st_npoints
-------------
 4
```

ST_NRings

ST_NRings は、入力ジオメトリでリングの数を返します。

構文

```
ST_NRings(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

INTEGER

geom が null の場合、null が返されます。

返される値は次のとおりです。

戻り値	ジオメトリのサブタイプ
0	geom が POINT、LINESTRING 、MULTIPOINT 、MULTILINESTRING サブタイプの場合に返されます。
リングの数。	geom が POLYGON または MULTIPOLYGON サブタイプの場合に返されます
すべてのコンポーネントのリングの数	geom が GEOMETRYCOLLECTION サブタイプの場合に返されます

例

次の SQL は、マルチポリゴンのリングの数を返します。

```
SELECT ST_NRings(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((0 0,-10 0,0
 -10,0 0)))'));
```

```
 st_nrings
-------------
 2
```

ST_NumGeometries

ST_NumGeometries は、入力ジオメトリのジオメトリの数を返します。

構文

```
ST_NumGeometries(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

INTEGER は geom 内のジオメトリの数を表します。

geom が null の場合、null が返されます。

geom が単一の空のジオメトリの場合、0 が返されます。

geom が、空ではない単一のジオメトリの場合、1 が返されます。

geom が GEOMETRYCOLLECTION または MULTI のサブタイプの場合、ジオメトリの数が返されます。

例

次の SQL は、入力 MULTILINESTRING のジオメトリの数を返します。

```
SELECT ST_NumGeometries(ST_GeomFromText('MULTILINESTRING((0 0,1 0,0 5),(3 4,13 26))'));
```

```
st_numgeometries
```

```
-------------
 2
```

ST_NumInteriorRings

ST_NumInteriorRings は、入力ポリゴンジオメトリのリングの数を返します。

構文

```
ST_NumInteriorRings(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`INTEGER`

geom が null の場合、null が返されます。

geom がポリゴンでない場合、null が返されます。

例

次の SQL は、入力ポリゴンの内部リングの数を返します。

```
SELECT ST_NumInteriorRings(ST_GeomFromText('POLYGON((0 0,100 0,100 100,0 100,0 0),(1
 1,1 5,5 1,1 1),(7 7,7 8,8 7,7 7))'));
```

```

 st_numinteriorrings
-------------
 2
```

ST_NumPoints

ST_NumPoints は、入力ジオメトリのポイントの数を返します。

構文

```
ST_NumPoints(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

INTEGER

geom が null の場合、null が返されます。

geom がサブタイプ LINESTRING でない場合は、null が返されます。

例

次の SQL は、入力 LINESTRING のポイントの数を返します。

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'));
```

```
st_numpoints
-------------
4
```

次の SQL は、入力 geom がサブタイプ LINESTRING ではないため、null を返します。

```
SELECT ST_NumPoints(ST_GeomFromText('MULTIPOINT(1 2,3 4)'));
```

```
st_numpoints
-------------
```

ST_Perimeter

ST_Perimeter は入力された面積ジオメトリに対して、その 2D 射影の周長 (境界の長さ) をデカルト座標系で返します。周長の単位は、入力ジオメトリの座標を表す単位と同じです。この関数は、ポイント、マルチポイント、および線形ジオメトリに対して 0 を返します。入力がジオメトリコレクションの場合、この関数はコレクションのジオメトリの周長の合計を返します。

入力がジオグラフィの場合、ST_Perimeter では、SRID で決定される回転楕円体上で計算され与えられた面積ジオグラフィについて、その 2D 射影に関する測地座標系での周長 (境界の長さ) を返します。周長の単位はメートルです。この関数は、ポイント、マルチポイント、および線形地形に対して 0 を返します。入力がジオメトリコレクションの場合、この関数はコレクション内にあるジオグラフィの周長の合計を返します。

構文

```
ST_Perimeter(geo)
```

引数

geo

データ型 `GEOMETRY` または `GEOGRAPHY` の値、または `GEOMETRY` もしくは `GEOGRAPHY` 型として評価される式です。

戻り型

`DOUBLE PRECISION`

geo が null の場合、null が返されます。

SRID 値が見つからない場合、エラーが返されます。

例

次の SQL は、マルチポリゴンのデカルト座標系での周長を返します。

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20 10,10 0)))'));
```

```
 st_perimeter
```

```
--------------------------------
    68.2842712474619
```

次の SQL は、マルチポリゴンのデカルト座標系での周長を返します。

```
SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((0 0,10 0,0 10,0 0)),((10 0,20 0,20
 10,10 0)))'));
```

```


 st_perimeter
--------------------------------
    68.2842712474619
```

次の SQL は、ジオグラフィ内にあるポリゴンの周長を返します。

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;POLYGON((0 0,1 0,0 1,0 0))'));
```

```


 st_perimeter
------------------
 378790.428393693
```

次の SQL は、ジオグラフィ内にあるライン文字列の周長を返します。

```
SELECT ST_Perimeter(ST_GeogFromText('SRID=4326;LINESTRING(5 0,10 0)'));
```

```


 st_perimeter
--------------
 0
```

ST_Perimeter2D

ST_Perimeter2D は、ST_Perimeter のエイリアスです。詳細については、「ST_Perimeter」を参照してください。

ST_Point

ST_Point は、入力座標値からポイントジオメトリを返します。

構文

```
ST_Point(x, y)
```

引数

x

最初の座標を表すデータ型 DOUBLE PRECISION の値。

y

2 番目の座標を表すデータ型 DOUBLE PRECISION の値。

戻り型

GEOMETRYサブタイプ の POINT。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が 0 に設定されます。

x または y が null の場合、null が返されます。

例

次の SQL は、入力座標からポイントジオメトリを作成します。

```
SELECT ST_AsText(ST_Point(5.0, 7.0));
```

```


st_astext
-------------
POINT(5 7)
```

ST_PointN

ST_PointN は、インデックス値で指定されたラインストリングのポイントを返します。負のインデックス値は、ラインストリングの末尾から逆方向にカウントされるため、-1 は最後のポイントになります。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_PointN(geom, index)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。サブタイプは LINESTRING である必要があります。

index

ラインストリングのポイントのインデックスを表すデータ型 INTEGER の値。

戻り型

GEOMETRYサブタイプ の POINT。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が 0 に設定されます。

geom または index が null の場合、null が返されます。

index が範囲外の場合、null が返されます。

geom が空の場合、null が返されます。

geom が LINESTRING でない場合、null が返されます。

例

次の SQL は、GEOMETRY オブジェクトの 6 つのポイントを持つ LINESTRING の EWKT (Extended well-known text) 表現を返し、そのラインストリングのインデックス 5 のポイントを返します。

```
SELECT ST_AsEWKT(ST_PointN(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5,0
 0)',4326), 5));
```

```
st_asewkt
-------------
```

```
SRID=4326;POINT(0 5)
```

ST_Points

ST_Points は、入力ジオメトリ内の空でないポイントをすべて含むマルチポイントジオメトリを返します。ST_Points は、リングジオメトリの開始点と終了点を含め、入力で重複しているポイントを削除しません。

構文

```
ST_Points(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRYサブタイプ の MULTIPOINT。

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値は、geom と同じです。

geom が null の場合、null が返されます。

geom が空の場合、空のマルチポイントが返されます。

例

次の SQL 例では、入力ジオメトリからマルチポイントジオメトリを作成します。その結果、入力ジオメトリ内の空でないポイントを含むマルチポイントジオメトリが作成されます。

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('LINESTRING(1 0,2 0,3 0)'),
 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((1 0),(2 0),(3 0))
```

```
SELECT ST_AsEWKT(ST_Points(ST_SetSRID(ST_GeomFromText('MULTIPOLYGON(((0 0,1 0,0 1,0
 0)))'), 4326)));
```

```
st_asewkt
-------------
SRID=4326;MULTIPOINT((0 0),(1 0),(0 1),(0 0))
```

ST_Polygon

ST_Polygon は、アウターリングが空間リファレンスシステム識別子 (SRID) に入力された値を持つ入力 LINESTRING であるポリゴンジオメトリを返します。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_Polygon(linestring, srid)
```

引数

linestring

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは、LINESTRING を表す `LINESTRING` である必要があります。linestring 値はクローズされている必要があります。

srid

SRID を表すデータ型 `INTEGER` の値。

戻り型

`GEOMETRY`サブタイプ の `POLYGON`。

返されたジオメトリの SRID 値は、srid に設定されます。

linestring または srid が null の場合、null が返されます。

linestring が LINESTRING でない場合、エラーが返されます。

linestring がクローズされていない場合、エラーが返されます。

srid が負数の場合、エラーが返されます。

例

次の SQL は SRID 値を使用してポリゴンを作成します。

```
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),4356));
```

```
st_asewkt
-------------
 SRID=4356;POLYGON((77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07))
```

ST_RemovePoint

ST_RemovePoint は、入力ジオメトリのインデックスの位置にあるポイントを削除したラインストリングジオメトリを返します。

このインデックスは 0 から始まります。結果の空間参照系識別子 (SRID) は、入力ジオメトリのものと同じです。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_RemovePoint(geom, index)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。サブタイプは LINESTRING である必要があります。

index

0 から始まるインデックスの位置を表すデータ型 INTEGER の値。

戻り型

GEOMETRY

geom または index が null の場合、null が返されます。

geom がサブタイプ LINESTRING でない場合、エラーが返されます。

index が範囲外の場合、エラーが返されます。インデックスの位置の有効な値は、0 から ST_NumPoints(geom) の間の値と -1 です。

例

次の SQL は、ラインストリングの最後のポイントを削除します。

```
WITH tmp(g) AS (SELECT ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0 5)',4326))
SELECT ST_AsEWKT(ST_RemovePoint(g, ST_NumPoints(g) - 1)) FROM tmp;
```

```


   st_asewkt
-----------------------------------------
 SRID=4326;LINESTRING(0 0,10 0,10 10,5 5)
```

ST_Reverse

ST_Reverse は、線形ジオメトリと面積ジオメトリの頂点の順序を逆にします。ポイントジオメトリまたはマルチポイントジオメトリの場合、元のジオメトリのコピーが返されます。ジオメトリコレクションの場合、ST_Reverse は、コレクション内の各ジオメトリの頂点の順序を逆にします。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_Reverse(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

GEOMETRY

返されるジオメトリの空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。

geom が null の場合、null が返されます。

例

次の SQL は、LINESTRING のポイントの順序を逆にします。

```
SELECT ST_AsEWKT(ST_Reverse(ST_GeomFromText('LINESTRING(1 0,2 0,3 0,4 0)', 4326)));
```

```
    st_asewkt
------------------------------------
  SRID=4326;LINESTRING(4 0,3 0,2 0,1 0)
```

ST_SetPoint

ST_SetPoint は、インデックスで指定された入力ライン文字列の位置に関して更新された座標を持つライン文字列を返します。新しい座標は、入力ポイントの座標です。

結果のジオメトリのディメンションは、geom1 値 のものと同じです。geom1 と geom2 でディメンションが異なる場合、geom2 が geom1 のディメンションに射影されます。

構文

```
ST_SetPoint(geom1, index, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` である必要があります。

index

インデックスの位置を表すデータ型 `INTEGER` の値。`0` はライン文字列の左からの最初のポイントを指し、`1` は 2 番目のポイントを指します。インデックスには負の値を指定できます。`-1` はライン文字列の右から最初のポイントを指し、`-2` はライン文字列の右から 2 番目のポイントを指します。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。サブタイプは POINT である必要があります。

戻り型

GEOMETRY

geom2 が空のポイントである場合、geom1 が返されます。

geom1、geom2、または index が null の場合、null が返されます。

geom1 が linestring でない場合、エラーが返されます。

インデックスが有効なインデックス範囲内にない場合、エラーが返されます。

geom2 が point でない場合、エラーが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

例

次の SQL は、入力ライン文字列の 2 番目のポイントを指定されたポイントに設定する新しいライン文字列を返します。

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), 2,
 ST_GeomFromText('POINT(7 9)')));
```

```


st_astext
-------------
 LINESTRING(1 2,3 2,7 9,1 2)
```

次の SQL 例では、新しいライン文字列を返します。ここでは、指定したポイントを持つライン文字列の右から 3 番目のポイント (インデックスは負の値) を設定します。

```
SELECT ST_AsText(ST_SetPoint(ST_GeomFromText('LINESTRING(1 2, 3 2, 5 2, 1 2)'), -3,
 ST_GeomFromText('POINT(7 9)')));
```

```
st_astext
-------------
 LINESTRING(1 2,7 9,5 2,1 2)
```

ST_SetSRID

ST_SetSRID は、入力ジオメトリと同じジオメトリを返します。空間リファレンスシステム識別子 (SRID) に入力された値で更新された場合を除きます。

構文

```
ST_SetSRID(geom, srid)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

srid

SRID を表すデータ型 `INTEGER` の値。

戻り型

`GEOMETRY`

返されたジオメトリの SRID 値は、srid に設定されます。

geom または srid が null の場合、null が返されます。

srid が負数の場合、エラーが返されます。

例

次の SQL は、LINESTRING の SRID 値を設定します。

```
SELECT ST_AsEWKT(ST_SetSRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27
 29.31,77.29 29.07)'),50));
```

```
st_asewkt
-------------
 SRID=50;LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)
```

ST_Simplify

ST_Simplify は、指定された許容値で Ramer-Douglas-Peucker アルゴリズムを使用して、入力ジオメトリの簡略化されたコピーを返します。入力ジオメトリのトポロジが保持されない場合があります。アルゴリズムの詳細については、ウィキペディアで Ramer—Douglas—Peucker アルゴリズムを参照してください。

ST_Simplify が距離を計算してジオメトリを単純化する際は、その計算で入力ジオメトリの 2D 射影が処理されます。

構文

```
ST_Simplify(geom, tolerance)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

許容値

Ramer-Douglas-Peucker アルゴリズムの許容レベルを表すデータ型 `DOUBLE PRECISION` の値。許容値が負の数の場合、0 が使用されます。

戻り型

`GEOMETRY`.

返されたジオメトリの空間リファレンスシステム識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

geom が null の場合、null が返されます。

例

次の SQL では、Ramer-Douglas-Peucker アルゴリズムを使用したユークリッド距離の許容値 1 を使用して、入力ライン文字列を簡略化しています。距離の単位は、ジオメトリ座標の単位と同じです。

```
SELECT ST_AsEWKT(ST_Simplify(ST_GeomFromText('LINESTRING(0 0,1 2,1 1,2 2,2 1)'), 1));
```

```
 st_asewkt
-----------
LINESTRING(0 0,1 2,2 1)
```

ST_SRID

ST_SRID は、入力ジオメトリの空間リファレンスシステム識別子 (SRID) を返します。SRID の詳細については、「Amazon Redshift での空間データのクエリ」を参照してください。

構文

```
ST_SRID(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`INTEGER` は geom の SRID 値を表します。

geom が null の場合、null が返されます。

例

次の SQL は、SRID 4326 に設定された linestring の SRID 値を返します。

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)',4326));
```

```
st_srid
-------------
 4326
```

次の SQL は、作成時に設定されない linestring の SRID 値を返します。この結果、SRID 値が `0` になります。

```
SELECT ST_SRID(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_srid
-------------
 0
```

ST_StartPoint

ST_StartPoint は、入力ラインストリングの最初のポイントを返します。結果の空間参照系識別子 (SRID) の値は、入力ジオメトリのものと同じです。結果のジオメトリのディメンションは、入力ジオメトリのものと同じです。

構文

```
ST_StartPoint(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。サブタイプは `LINESTRING` である必要があります。

戻り型

`GEOMETRY`

geom が null の場合、null が返されます。

geom が空の場合、null が返されます。

geom が LINESTRING でない場合、null が返されます。

例

次の SQL は、GEOMETRY オブジェクトの 4 つのポイントを持つ LINESTRING の EWKT (Extended well-known text) 表現を返し、そのラインストリングの最初のポイントを返します。

```
SELECT ST_AsEWKT(ST_StartPoint(ST_GeomFromText('LINESTRING(0 0,10 0,10 10,5 5,0
 5)',4326)));
```

```

st_asewkt
-------------
 SRID=4326;POINT(0 0)
```

ST_Touches

ST_Touches は、2 つの入力ジオメトリの 2D 射影が接している場合に true を返します。2 つのジオメトリは、空ではなく、交わっており、共通の内部ポイントがない場合に接します。

構文

```
ST_Touches(geom1, geom2)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、ポリゴンがラインストリングに接しているかどうかをチェックします。

```
SELECT ST_Touches(ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))'),
 ST_GeomFromText('LINESTRING(20 10,20 0,10 0)'));
```

```
 st_touches
-------------
 t
```

ST_Transform

ST_Transform は、入力された空間参照系識別子 (SRID) によって定義される空間参照系に変換された座標を使用する、新しいジオメトリを返します。

構文

```
ST_Transform(geom, srid)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

srid

SRID を表すデータ型 `INTEGER` の値。

戻り型

`GEOMETRY`.

返されたジオメトリの SRID 値は、srid に設定されます。

geom または srid が null の場合、null が返されます。

入力の geom に関連付けられている SRID 値が存在しない場合は、エラーが返されます。

srid が存在しない場合、エラーが返されます。

例

次の SQL は、空のジオメトリコレクションの SRID を変換します。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('GEOMETRYCOLLECTION EMPTY', 3857),
 4326));
```

```
             st_asewkt
------------------------------------
 SRID=4326;GEOMETRYCOLLECTION EMPTY
```

次の SQL は、ライン文字列の SRID 値を設定します。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('LINESTRING(110 40, 2 3, -10 80, -7 9,
 -22 -33)', 4326), 26918));
```

```
     st_asewkt
-------------------------------------------------------------------------------------------------------
 SRID=26918;LINESTRING(73106.6977300955 15556182.9688576,14347201.5059964
 1545178.32934967,1515090.41262989 9522193.25115316,10491250.83295
 2575457.28410878,5672303.72135968 -5233682.61176205)
```

次の SQL は、ポリゴンの SRID を変換します。

```
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromText('POLYGON Z ((-10 10 -7, -65 10 -6, -10 64
 -5, -10 10 -7), (-11 11 5, -11 12 6, -12 11 7, -11 11 5))', 6989), 6317));
```

```
                                          st_asewkt
-------------------------------------------------------------------------------------------------------
 SRID=6317;POLYGON Z ((6186430.2771091 -1090834.57212608
 1100247.33216237,2654831.67853801 -5693304.90741276 1100247.50581055,2760987.41750022
 -486836.575101877 5709710.44137268,6186430.2771091 -1090834.57212608
 1100247.33216237),(6146675.25029258 -1194792.63532103 1209007.1115113,6125027.87562215
```

```
-1190584.81194058 1317403.77865723,6124888.99555252 -1301885.3455052
1209007.49312929,6146675.25029258 -1194792.63532103 1209007.1115113))
```

ST_Union

ST_Union は、2 つのジオメトリの和集合を表すジオメトリを返します。つまり、入力ジオメトリをマージして、オーバーラップのない結果のジオメトリを生成します。

構文

```
ST_Union(geom1, geom2)
```

引数

geom1

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

geom2

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`GEOMETRY`

返されたジオメトリの空間リファレンス識別子 (SRID) 値が、入力ジオメトリの SRID 値です。

geom1 または geom2 が null の場合、null が返されます。

geom1 もしくは geom2 が空の場合、空のジオメトリが返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) が同じ値でない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクション、linestring、または multilinestring である場合、エラーが返されます。

geom1 もしくは geom2 が 二次元の (2D) ジオメトリでない場合、エラーが返されます。

例

次の SQL は、2 つの入力ジオメトリの和集合を表す、空でないジオメトリを返します。

```
SELECT ST_AsEWKT(ST_Union(ST_GeomFromText('POLYGON((0 0,100 100,0 200,0 0))'),
 ST_GeomFromText('POLYGON((0 0,10 0,0 10,0 0))')));
```

```
        st_asewkt
-------------------------
 POLYGON((0 0,0 200,100 100,5 5,10 0,0 0))
```

ST_Within

ST_Within は、最初の入力ジオメトリの 2D 射影が 2 番目の入力ジオメトリの 2D 射影内にある場合、true を返します。

たとえば、A のすべてのポイントが B のポイントであり、内部の交差が空でない場合にジオメトリ A はジオメトリ B 内にあります。

ST_Within(A、B) は ST_Contains(B、A) と等しいものです。

構文

```
ST_Within(geom1, geom2)
```

引数

geom1

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。この値を geom2 と比較して、geom2 内にあるかどうかを判断します。

geom2

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

BOOLEAN

geom1 または geom2 が null の場合、null が返されます。

geom1 および geom2 の空間リファレンスシステム識別子 (SRID) 値が同じでない場合、エラーが返されます。

geom1 または geom2 がジオメトリコレクションである場合、エラーが返されます。

例

次の SQL は、最初のポリゴンが 2 番目のポリゴン内にあるかどうかを確認します。

```
SELECT ST_Within(ST_GeomFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeomFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'));
```

```


st_within
-----------
 true
```

ST_X

ST_X は、入力ポイントの最初の座標を返します。

構文

```
ST_X(point)
```

引数

point

    データ型 `GEOMETRY` の `POINT` 値。

戻り型

最初の座標の `DOUBLE PRECISION` 値。

point が null の場合、null が返されます。

point が空のポイントの場合、null が返されます。

point が `POINT` でない場合、エラーが返されます。

例

次の SQL は、ポイントの最初の座標を返します。

```
SELECT ST_X(ST_Point(1,2));
```

```
st_x
-----------
 1.0
```

ST_XMax

ST_XMax は、入力ジオメトリの最大の最初の座標を返します。

構文

```
ST_XMax(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

最大の最初の座標の `DOUBLE PRECISION` 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

例

次の SQL は、LINESTRING の最大の最初の座標を返します。

```
SELECT ST_XMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmax
-----------
```

```
77.42
```

ST_XMin

ST_XMin は、入力ジオメトリの最小の最初の座標を返します。

構文

```
ST_XMin(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

最小の最初の座標の `DOUBLE PRECISION` 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

例

次の SQL は、LINESTRING の最小の最初の座標を返します。

```
SELECT ST_XMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```
st_xmin
-----------
 77.27
```

ST_Y

ST_Y は、入力ポイントの 2 番目の座標を返します。

構文

```
ST_Y(point)
```

引数

point

データ型 GEOMETRY の POINT 値。

戻り型

2 番目の座標の DOUBLE PRECISION 値。

point が null の場合、null が返されます。

point が空のポイントの場合、null が返されます。

point が POINT でない場合、エラーが返されます。

例

次の SQL は、ポイントの 2 番目の座標を返します。

```
SELECT ST_Y(ST_Point(1,2));
```

```
st_y
-----------
 2.0
```

ST_YMax

ST_YMax は、入力ジオメトリの最大の 2 番目の座標を返します。

構文

```
ST_YMax(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

最大の 2 番目の座標 の DOUBLE PRECISION 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

例

次の SQL は、LINESTRING の 最大の 2 番目の座標を返します。

```
SELECT ST_YMax(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```


st_ymax
-----------
 29.31
```

ST_YMin

ST_YMin 入力ジオメトリの最小の 2 番目の座標を返します。

構文

```
ST_YMin(geom)
```

引数

geom

データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

最小の 2 番目の座標の DOUBLE PRECISION 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

例

次の SQL は、LINESTRING の 最小の 2 番目の座標を返します。

```
SELECT ST_YMin(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29
 29.07)'));
```

```

st_ymin
-----------
 29.07
```

ST_Z

ST_Z は、入力ポイントの z 座標を返します。

構文

```
ST_Z(point)
```

引数

point

データ型 GEOMETRY の POINT 値。

戻り型

m 座標系の DOUBLE PRECISION 値。

point が null の場合、null が返されます。

point が 2D または 3DM のポイントの場合、null が返されます。

point が空のポイントの場合、null が返されます。

point が POINT でない場合、エラーが返されます。

例

次の SQL は、1 つのポイントの z 座標を、3DZ ジオメトリで返します。

```
SELECT ST_Z(ST_GeomFromEWKT('POINT Z (1 2 3)'));
```

```

st_z
-----------
 3
```

次の SQL は、1 つのポイントの z 座標を、4D ジオメトリで返します。

```
SELECT ST_Z(ST_GeomFromEWKT('POINT ZM (1 2 3 4)'));
```

```

st_z
-----------
 3
```

ST_ZMax

ST_ZMax は、入力ジオメトリの最大の z 座標を返します。

構文

```
ST_ZMax(geom)
```

引数

geom

　データ型 GEOMETRY の値または GEOMETRY 型と評価される式の値。

戻り型

最大の z 座標の DOUBLE PRECISION 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

geom が、2D または 3DM ジオメトリの場合、null が返されます。

例

次の SQL は、ライン文字列の最大の z 座標を、3DZ ジオメトリで返します。

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmax
-----------
  8
```

次の SQL は、ライン文字列の最大の z 座標を、4D ジオメトリで返します。

```
SELECT ST_ZMax(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmax
-----------
  10
```

ST_ZMin

ST_ZMin は、入力ジオメトリの最小の z 座標を返します。

構文

```
ST_ZMin(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

最小の z 座標の `DOUBLE PRECISION` 値。

geom が空の場合、null が返されます。

geom が null の場合、null が返されます。

geom が、2D または 3DM ジオメトリの場合、null が返されます。

例

次の SQL は、ライン文字列の最小の z 座標を、3DZ ジオメトリで返します。

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING Z (0 1 2, 3 4 5, 6 7 8)'));
```

```
st_zmin
-----------
  2
```

次の SQL は、ライン文字列の最小の z 座標を、4D ジオメトリで返します。

```
SELECT ST_ZMin(ST_GeomFromEWKT('LINESTRING ZM (0 1 2 3, 4 5 6 7, 8 9 10 11)'));
```

```
st_zmin
-----------
  2
```

SupportsBBox

SupportsBBox は、入力ジオメトリが、事前に計算された境界ボックスによるエンコーディングをサポートしている場合に true を返します。バウンディングボックスのサポートの詳細については、「境界ボックス」を参照してください。

構文

```
SupportsBBox(geom)
```

引数

geom

データ型 `GEOMETRY` の値または `GEOMETRY` 型と評価される式の値。

戻り型

`BOOLEAN`

geom が null の場合、null が返されます。

例

次の SQL は、入力ポイントジオメトリが境界ボックスによるエンコードをサポートしているため、true を返します。

```
SELECT SupportsBBox(AddBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
t
```

次の SQL は、入力ポイントジオメトリが境界ボックスによるエンコードをサポートしていないため、false を返します。

```
SELECT SupportsBBox(DropBBox(ST_GeomFromText('POLYGON((0 0,1 0,0 1,0 0))')));
```

```
supportsbbox
--------------
f
```

文字列関数

トピック

- [|| (連結) 演算子]

- ASCII 関数
- BPCHARCMP 関数
- BTRIM 関数
- BTTEXT_PATTERN_CMP 関数
- CHAR_LENGTH 関数
- CHARACTER_LENGTH 関数
- CHARINDEX 関数
- CHR 関数
- COLLATE 関数
- CONCAT 関数
- CRC32 関数
- DIFFERENCE 関数
- INITCAP 関数
- LEFT 関数および RIGHT 関数
- LEN 関数
- LENGTH 関数
- LOWER 関数
- LPAD 関数および RPAD 関数
- LTRIM 関数
- OCTETINDEX 関数
- OCTET_LENGTH 関数
- POSITION 関数
- QUOTE_IDENT 関数
- QUOTE_LITERAL 関数
- REGEXP_COUNT 関数
- REGEXP_INSTR 関数
- REGEXP_REPLACE 関数
- REGEXP_SUBSTR 関数
- REPEAT 関数

- REPLACE 関数
- REPLICATE 関数
- REVERSE 関数
- RTRIM 関数
- SOUNDEX 関数
- SPLIT_PART 関数
- STRPOS 関数
- STRTOL 関数
- SUBSTRING 関数
- TEXTLEN 関数
- TRANSLATE 関数
- TRIM 関数
- UPPER 関数

文字列関数は、文字列、または文字列に評価される式を処理します。これらの関数の string 引数がリテラル値である場合は、その値を一重引用符で囲む必要があります。サポートされるデータ型には、CHAR や VARCHAR があります。

次のセクションでは、サポートされる関数の関数名と構文を示し、それらについて説明します。文字列のオフセットはすべて 1 から始まります。

廃止されたリーダーノード専用の関数

次の文字列関数は、リーダーノードのみで実行されるため、非推奨となりました。詳細については、「リーダーノード専用関数」を参照してください。

- GET_BYTE
- SET_BIT
- SET_BYTE
- TO_ASCII

|| (連結) 演算子

|| 記号の両側にある 2 つの式を連結し、連結した式を返します。

CONCAT 関数 と同様です。

> **Note**
>
> 一方または両方の式が null の場合、連結の結果は NULL になります。

構文

```
expression1 || expression2
```

引数

expression1

CHAR 文字列、VARCHAR 文字列、バイナリ式、またはこれらの型のいずれかに評価される式。

expression2

CHAR 文字列、VARCHAR 文字列、バイナリ式、またはこれらの型のいずれかに評価される式。

戻り型

文字列の戻り型は、入力引数の型と同じです。例えば、VARCHAR 型の 2 つの文字列を連結すると、VARCHAR 型の文字列が返されます。

例

以下の例では、TICKIT サンプルデータベースの USERS テーブルと VENUE テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

サンプルデータベースで USERS テーブルの FIRSTNAME フィールドと LASTNAME フィールドを連結するには、次の例を使用します。

```
SELECT (firstname || ' ' || lastname) as fullname
FROM users
ORDER BY 1
LIMIT 10;

+-----------------+
```

```
|    fullname     |
+-----------------+
| Aaron Banks     |
| Aaron Booth     |
| Aaron Browning  |
| Aaron Burnett   |
| Aaron Casey     |
| Aaron Cash      |
| Aaron Castro    |
| Aaron Dickerson |
| Aaron Dixon     |
| Aaron Dotson    |
+-----------------+
```

Null を含む可能性がある列を連結するには、NVL および COALESCE 関数式を使用します。次の例は、NVL を使用して、NULL が発生するたびに `0` を返します。

```
SELECT (venuename || ' seats ' || NVL(venueseats, 0)) as seating
FROM venue
WHERE venuestate = 'NV' or venuestate = 'NC'
ORDER BY 1
LIMIT 10;

+-------------------------------------+
|               seating               |
+-------------------------------------+
| Ballys Hotel seats 0                |
| Bank of America Stadium seats 73298 |
| Bellagio Hotel seats 0              |
| Caesars Palace seats 0              |
| Harrahs Hotel seats 0               |
| Hilton Hotel seats 0                |
| Luxor Hotel seats 0                 |
| Mandalay Bay Hotel seats 0          |
| Mirage Hotel seats 0                |
| New York New York seats 0           |
+-------------------------------------+
```

ASCII 関数

ASCII 関数は、指定した文字列の最初の文字の ASCII コード、または Unicode コードポイントを返します。文字列が空の場合、関数は `0` を返します。文字列が null の場合は、NULL を返します。

構文

```
ASCII('string')
```

引数

string

CHAR 文字列または VARCHAR 文字列。

戻り型

INTEGER

例

NULL を返すには、次の例を使用します。2 つの引数が同じである場合、NULLIF 関数は NULL を返し、ASCII 関数の入力引数は NULL になります。詳細については、「NULLIF 関数」を参照してください。

```
SELECT ASCII(NULLIF('',''));

+-------+
| ascii |
+-------+
|  NULL |
+-------+
```

ASCII コード 0 を返すには、次の例を使用します。

```
SELECT ASCII('');

+-------+
| ascii |
+-------+
|     0 |
+-------+
```

amazon という単語の 1 文字目の ASCII コード 97 を返すには、次の例を使用します。

```
SELECT ASCII('amazon');
```

```
+--------+
| ascii |
+--------+
|    97 |
+--------+
```

Amazon という単語の 1 文字目の ASCII コード 65 を返すには、次の例を使用します。

```
SELECT ASCII('Amazon');

+--------+
| ascii |
+--------+
|    65 |
+--------+
```

BPCHARCMP 関数

2 つの文字列の値を比較し、整数を返します。文字列が同じである場合、関数は 0 を返します。1 番目の文字列がアルファベット順でより大きい場合、関数は 1 を返します。2 番目の文字列がより大きい場合、関数は -1 を返します。

マルチバイトの文字の場合は、バイトエンコーディングを基に比較が行われます。

BTTEXT_PATTERN_CMP 関数 のシノニム。

構文

```
BPCHARCMP(string1, string2)
```

引数

string1

CHAR 文字列または VARCHAR 文字列。

string2

CHAR 文字列または VARCHAR 文字列。

戻り型

INTEGER

例

以下の例では、TICKIT サンプルデータベースの USERS テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

USERS テーブル内の最初の 10 個のエントリについて、ユーザーの名と姓をアルファベット順で比較し、名が姓よりアルファベット順で先になるかどうかを判別するには、次の例を使用します。FIRSTNAME の文字列が LASTNAME の文字列よりアルファベット順で後になるエントリについては、関数は 1 を返します。LASTNAME が FIRSTNAME よりアルファベット順で後になる場合、関数は -1 を返します。

```
SELECT userid, firstname, lastname, BPCHARCMP(firstname, lastname)
FROM users
ORDER BY 1, 2, 3, 4
LIMIT 10;

+--------+-----------+-----------+-----------+
| userid | firstname | lastname  | bpcharcmp |
+--------+-----------+-----------+-----------+
|      1 | Rafael    | Taylor    |        -1 |
|      2 | Vladimir  | Humphrey  |         1 |
|      3 | Lars      | Ratliff   |        -1 |
|      4 | Barry     | Roy       |        -1 |
|      5 | Reagan    | Hodge     |         1 |
|      6 | Victor    | Hernandez |         1 |
|      7 | Tamekah   | Juarez    |         1 |
|      8 | Colton    | Roy       |        -1 |
|      9 | Mufutau   | Watkins   |        -1 |
|     10 | Naida     | Calderon  |         1 |
+--------+-----------+-----------+-----------+
```

この関数が `0` を返す、USERS テーブル内のすべてのエントリを返すには、次の例を使用します。FIRSTNAME と LASTNAME が同じである場合、この関数は `0` を返します。

```
SELECT userid, firstname, lastname,
BPCHARCMP(firstname, lastname)
FROM users
WHERE BPCHARCMP(firstname, lastname)=0
ORDER BY 1, 2, 3, 4;

+--------+-----------+----------+-----------+
| userid | firstname | lastname | bpcharcmp |
```

```
+--------+-----------+----------+-----------+
|     62 | Chase     | Chase    |         0 |
|   4008 | Whitney   | Whitney  |         0 |
|  12516 | Graham    | Graham   |         0 |
|  13570 | Harper    | Harper   |         0 |
|  16712 | Cooper    | Cooper   |         0 |
|  18359 | Chase     | Chase    |         0 |
|  27530 | Bradley   | Bradley  |         0 |
|  31204 | Harding   | Harding  |         0 |
+--------+-----------+----------+-----------+
```

BTRIM 関数

BTRIM 関数は、先頭および末尾の空白を削除するか、またはオプションで指定された文字列と一致する先頭および末尾の文字を削除することによって、文字列を切り捨てます。

構文

```
BTRIM(string [, trim_chars ] )
```

引数

string

切り捨てる入力 VARCHAR 文字列。

trim_chars

イッチする文字を含む VARCHAR 文字列。

戻り型

BTRIM 関数は、VARCHAR 型の文字列を返します。

例

次の例では、文字列 ' abc ' の先頭および末尾の空白を切り捨てます。

```
select '     abc    ' as untrim, btrim('     abc    ') as trim;

untrim    | trim
```

```
----------+------
   abc     | abc
```

次の例では、文字列 'xyzaxyzbxyzcxyz' から先頭および末尾の文字列 'xyz' を削除します。先頭および末尾にある 'xyz' は削除されますが、文字列内部にあるその文字列は削除されません。

```
select 'xyzaxyzbxyzcxyz' as untrim,
btrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | axyzbxyzc
```

次の例では、trim_chars リスト 'tes' のいずれかの文字と一致する文字列 'setuphistorycassettes' の先頭と末尾の部分を削除します。入力文字列の先頭または末尾にある trim_chars リストに含まれていない別の文字の前に出現する t、e または s が削除されます。

```
SELECT btrim('setuphistorycassettes', 'tes');

     btrim
-----------------
 uphistoryca
```

BTTEXT_PATTERN_CMP 関数

BPCHARCMP 関数のシノニム。

詳細については、「BPCHARCMP 関数」を参照してください。

CHAR_LENGTH 関数

LEN 関数のシノニム。

「LEN 関数」を参照してください。

CHARACTER_LENGTH 関数

LEN 関数のシノニム。

「LEN 関数」を参照してください。

CHARINDEX 関数

文字列内の指定されたサブ文字列の位置を返します。

同様の関数については、「POSITION 関数」および「STRPOS 関数」を参照してください。

構文

```
CHARINDEX( substring, string )
```

引数

substring

string 内を検索するサブ文字列。

string

検索する文字列または列。

戻り型

INTEGER

CHARINDEX 関数は、サブ文字列の位置 (0 ではなく 1 から始まる) に対応する `INTEGER` を返します。position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。文字列内でサブ文字列が見つからなかった場合、CHARINDEX は `0` を返します。

例

dog という語の中での文字列 `fish` の位置を返すには、次の例を使用します。

```
SELECT CHARINDEX('fish', 'dog');

+-----------+
| charindex |
+-----------+
|         0 |
+-----------+
```

dogfish という語の中での文字列 `fish` の位置を返すには、次の例を使用します。

```
SELECT CHARINDEX('fish', 'dogfish');

+-----------+
| charindex |
+-----------+
|         4 |
+-----------+
```

以下の例は、TICKIT サンプルデータベースの SALES テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブル内でコミッションが 999.00 を上回る販売取引の数を返すには、次の例を使用します。このコマンドは、999.00 を超えるコミッションをカウントします。これは、小数点以下の桁数がコミッション値の先頭から 4 桁より大きいかどうかを調べることでカウントされます。

```
SELECT DISTINCT CHARINDEX('.', commission), COUNT (CHARINDEX('.', commission))
FROM sales
WHERE CHARINDEX('.', commission) > 4
GROUP BY CHARINDEX('.', commission)
ORDER BY 1,2;

+-----------+-------+
| charindex | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

CHR 関数

CHR 関数は、入力パラメータに指定された ASCII コードポイント値と一致する文字を返します。

構文

```
CHR(number)
```

引数

number

入力パラメータは、ASCII コードポイント値を表す INTEGER です。

戻り型

CHAR

ASCII 文字が入力値と一致した場合、CHR 関数は CHAR 型の文字列を返します。入力された数値に一致する ASCII が存在しない場合、この関数は NULL を返します。

例

ASCII コードポイント 0 に対応する文字を返すには、次の例を使用します。CHR 関数は入力 0 に対して NULL を返すことに注意してください。

```
SELECT CHR(0);

+------+
| chr  |
+------+
|      |
+------+
```

ASCII コードポイント 65 に対応する文字を返すには、次の例を使用します。

```
SELECT CHR(65);

+------+
| chr  |
+------+
| A    |
+------+
```

大文字 A (ASCII コードポイント 65) で始まる固有のイベント名を返すには、次の例を使用します。次の例では、TICKIT サンプルデータベースの EVENT テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

```
SELECT DISTINCT eventname FROM event
WHERE SUBSTRING(eventname, 1, 1)=CHR(65) LIMIT 5;

+-----------------------+
|       eventname       |
+-----------------------+
```

```
| A Catered Affair       |
| As You Like It         |
| A Man For All Seasons  |
| Alan Jackson           |
| Armando Manzanero      |
+------------------------+
```

COLLATE 関数

COLLATE 関数は、文字列の列または表現に関する照合をオーバーライドします。

データベースの照合を使用してテーブルを作成する方法については、「CREATE TABLE」を参照してください。

データベース照合を使用してデータベースを作成する方法については、「CREATE DATABASE」を参照してください。

構文

```
COLLATE( string, 'case_sensitive' | 'case_insensitive');
```

引数

string

上書きする先の文字列の列または式です。

'case_sensitive' | 'case_insensitive'

照合名を示す文字列定数です。Amazon Redshift では、case_sensitive もしくは case_insensitive のみがサポートされます。

戻り型

COLLATE 関数は、最初の入力式の型に応じて `VARCHAR` または `CHAR` を返します。この関数は、最初の入力引数についての照合を変更するだけで、出力値は変更されません。

例

テーブル T を作成し、テーブル T の col1 を`case_sensitive` と定義するには、次の例を使用します。

```
CREATE TABLE T ( col1 Varchar(20) COLLATE case_sensitive );

INSERT INTO T VALUES ('john'),('JOHN');
```

最初のクエリを実行すると、Amazon Redshift は john だけを返します。col1 上で COLLATE 関数が実行された後、照合は case_insensitive になります。2 番目のクエリは john と JOHN の両方を返します。

```
SELECT * FROM T WHERE col1 = 'john';

+------+
| col1 |
+------+
| john |
+------+

SELECT * FROM T WHERE COLLATE(col1, 'case_insensitive') = 'john';

+------+
| col1 |
+------+
| john |
| JOHN |
+------+
```

テーブル A を作成し、テーブル A の col1 を case_insensitive と定義するには、次の例を使用します。

```
CREATE TABLE A ( col1 Varchar(20) COLLATE case_insensitive );

INSERT INTO A VALUES ('john'),('JOHN');
```

最初のクエリを実行すると、Amazon Redshift は john と JOHN の両方を返します。col1 上で COLLATE 関数が実行された後、照合は case_sensitive になります。2 番目のクエリは john のみを返します。

```
SELECT * FROM A WHERE col1 = 'john';

+------+
| col1 |
```

```
+------+
| john |
| JOHN |
+------+

SELECT * FROM A WHERE COLLATE(col1, 'case_sensitive') = 'john';

+------+
| col1 |
+------+
| john |
+------+
```

CONCAT 関数

CONCAT 関数は、2 つの式を連結し、結果の表現を返します。3 つ以上の式を連結するには、CONCAT 関数をネストして使用します。2 つの式の間に連結演算子 (||) を指定した場合も、CONCAT 関数と同じ結果が返されます。

構文

```
CONCAT ( expression1, expression2 )
```

引数

expression1, expression2

どちらの引数にも、固定長文字列、可変長文字列、バイナリ式、またはこれらの入力のいずれかとして評価される式を指定できます。

戻り型

CONCAT は式を返します。式のデータ型は、入力引数の型と同じです。

入力式の型が異なる場合、Amazon Redshift はそれらの式の 1 つを暗黙的に型キャストしようとします。値を型キャストできない場合は、エラーが返されます。

使用に関する注意事項

- CONCAT 関数ならびに連結演算子のどちらにおいても、一方または両方の表現が null である場合は、連結の結果も null になります。

例

次の例では、2 つの文字リテラルを連結します。

```
SELECT CONCAT('December 25, ', '2008');

concat
-------------------
December 25, 2008
(1 row)
```

次のクエリでは、CONCAT ではなく || 演算子を使用しており、同じ結果が返されます。

```
SELECT 'December 25, '||'2008';

?column?
-------------------
December 25, 2008
(1 row)
```

次の例では、ネストされた CONCAT 関数を別の CONCAT 関数の中で使用して 3 つの文字列を連結します。

```
SELECT CONCAT('Thursday, ', CONCAT('December 25, ', '2008'));

concat
-----------------------------
Thursday, December 25, 2008
(1 row)
```

NULL を含む可能性のある列を連結するには、[NVL および COALESCE 関数](#) を使用し、NULL に遭遇したときに特定の値を返します。次の例は、NVL を使用して、NULL が発生するたびに 0 を返します。

```
SELECT CONCAT(venuename, CONCAT(' seats ', NVL(venueseats, 0))) AS seating
FROM venue WHERE venuestate = 'NV' OR venuestate = 'NC'
ORDER BY 1
LIMIT 5;

seating
-----------------------------------
Ballys Hotel seats 0
```

```
Bank of America Stadium seats 73298
Bellagio Hotel seats 0
Caesars Palace seats 0
Harrahs Hotel seats 0
(5 rows)
```

次のクエリでは、VENUE テーブル内の CITY 値と STATE 値を連結します。

```
SELECT CONCAT(venuecity, venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
-------------------
DenverCO
Kansas CityMO
East RutherfordNJ
LandoverMD
(4 rows)
```

次のクエリでは、ネストされた CONCAT 関数を使用しています。このクエリは、VENUE テーブル内の CITY 値と STATE 値を連結しますが、結果の文字列をカンマおよびスペースで区切ります。

```
SELECT CONCAT(CONCAT(venuecity,', '),venuestate)
FROM venue
WHERE venueseats > 75000
ORDER BY venueseats;

concat
---------------------
Denver, CO
Kansas City, MO
East Rutherford, NJ
Landover, MD
(4 rows)
```

次の例では、2 つのバイナリ式を連結します。ここで、abcは 616263 の 16 進数表現を含む 2 進数値 であり、defは 646566 の 16 進数表現を含む 2 進数値です。結果は、バイナリ値の 16 進数表現として自動的に出力されます。

```
SELECT CONCAT('abc'::VARBYTE, 'def'::VARBYTE);
```

```
concat
--------------------
616263646566
```

CRC32 関数

CRC32 はエラー検出に使用される機能です。この関数は、CRC32 アルゴリズムを使用して、ソースデータとターゲットデータの間の変更を検出します。CRC32 関数は、可変長文字列を 8 文字の文字列に変換します。この 8 文字の文字列は、32 ビットバイナリシーケンスの 16 進値をテキストで表記したものです。ソースデータとターゲットデータ間の変更を検出するには、ソースデータに CRC32 関数を使用して出力を保存します。次に、ターゲットデータに CRC32 関数を使用して、その出力をソースデータからの出力と比較します。データが変更されていない場合は出力が同じになり、データが変更されている場合は出力が異なります。

構文

```
CRC32(string)
```

引数

string

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

戻り型

CRC32 関数は、8 文字の文字列を返します。この 8 文字の文字列は、32 ビットバイナリシーケンスの 16 進値をテキストで表記したものです。Amazon Redshift の CRC32 関数は CRC-32C の多項式に基づいています。

例

文字列 `Amazon Redshift` の 8 ビット値を表示するには。

```
SELECT CRC32('Amazon Redshift');

+----------+
|  crc32   |
+----------+
| f2726906 |
```

```
+----------+
```

DIFFERENCE 関数

DIFFERENCE 関数は 2 つの文字列の American Soundex コードを比較します。この関数は INTEGER を返し、Soundex コード間で一致する文字の数を示します。

Soundex コードは 4 文字の長さの文字列です。Soundex コードは、単語のスペルではなく発音方法を表します。例えば、Smith と Smyth は同じ Soundex コードを持ちます。

構文

```
DIFFERENCE(string1, string2)
```

引数

string1

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

string2

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

戻り型

INTEGER

DIFFERENCE 関数は、American Soundex コードで 2 つの文字列内の一致する文字の数をカウントする INTEGER 値 0～4 を返します。Soundex コードは 4 文字であるため、文字列の American Soundex コード値の 4 文字すべてが同じ場合、DIFFERENCE 関数は 4 を返します。2 つの文字列のうちの 1 つが空の場合、DIFFERENCE は 0 を返します。この関数は、どちらの文字列にも有効な文字が含まれていない場合、1 を返します。DIFFERENCE 関数は、a～z および A～Z を含む、英字のアルファベットで小文字または大文字の ASCII 文字のみを変換します。DIFFERENCE 関数は、他の文字を無視します。

例

文字列 % および @ の Soundex 値を比較するには、次の例を使用します。この関数は、どちらの文字列にも有効な文字が含まれていないため、1 を返します。

```
SELECT DIFFERENCE('%', '@');

+------------+
| difference |
+------------+
|          1 |
+------------+
```

Amazon および空の文字列の Soundex 値を比較するには、次の例を使用します。2 つの文字列のうちの 1 つが空であるため、この関数は 0 を返します。

```
SELECT DIFFERENCE('Amazon', '');

+------------+
| difference |
+------------+
|          0 |
+------------+
```

文字列 Amazon および Ama の Soundex 値を比較するには、次の例を使用します。文字列の Soundex 値の 2 文字が同じであるため、この関数は 2 を返します。

```
SELECT DIFFERENCE('Amazon', 'Ama');

+------------+
| difference |
+------------+
|          2 |
+------------+
```

文字列 Amazon および +-*/%Amazon の Soundex 値を比較するには、次の例を使用します。文字列の Soundex 値の 4 文字すべてが同じであるため、この関数は 4 を返します。この関数は、2 番目の文字列の無効な文字 +-*/% を無視することに注目してください。

```
SELECT DIFFERENCE('Amazon', '+-*/%Amazon');

+------------+
| difference |
+------------+
|          4 |
```

```
+------------+
```

文字列 AC/DC および `Ay See Dee See` の Soundex 値を比較するには、次の例を使用します。文字列の Soundex 値の 4 文字すべてが同じであるため、この関数は 4 を返します。

```
SELECT DIFFERENCE('AC/DC', 'Ay See Dee See');

+------------+
| difference |
+------------+
|          4 |
+------------+
```

INITCAP 関数

指定された文字列内の各単語の先頭文字を大文字にします。INITCAP は、UTF-8 マルチバイト文字に対応しています (1 文字につき最大で 4 バイトまで)。

構文

```
INITCAP(string)
```

引数

string

`CHAR` 文字列、`VARCHAR` 文字列、あるいは `CHAR` または `VARCHAR` 型に暗黙的に評価される式。

戻り型

VARCHAR

使用に関する注意事項

INITCAP 関数は、文字列内の各単語の先頭文字を大文字にして、2 文字目以降を小文字 (のまま) にします。そのため、単語区切り文字として機能する文字 (スペース文字以外) を理解することが重要です。単語区切り文字は、任意の非英数字 (句読点、記号、および制御文字を含む) です。以下の文字はすべて、単語区切り文字です。

```
! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
```

タブ、改行文字、フォームフィード、ラインフィード、およびキャリッジリターンも単語区切り文字です。

例

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルと USERS テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATDESC 列内の各単語の先頭文字を大文字にするには、次の例を使用します。

```
SELECT catid, catdesc, INITCAP(catdesc)
FROM category
ORDER BY 1, 2, 3;

+-------+--------------------------------------------
+--------------------------------------------+
| catid |                  catdesc                   |                  initcap
          |
+-------+--------------------------------------------
+--------------------------------------------+
|     1 | Major League Baseball                      | Major League Baseball
          |
|     2 | National Hockey League                     | National Hockey League
          |
|     3 | National Football League                   | National Football League
          |
|     4 | National Basketball Association            | National Basketball Association
          |
|     5 | Major League Soccer                        | Major League Soccer
          |
|     6 | Musical theatre                            | Musical Theatre
          |
|     7 | All non-musical theatre                    | All Non-Musical Theatre
          |
|     8 | All opera and light opera                  | All Opera And Light Opera
          |
|     9 | All rock and pop music concerts            | All Rock And Pop Music Concerts
          |
|    10 | All jazz singers and bands                 | All Jazz Singers And Bands
          |
|    11 | All symphony, concerto, and choir concerts | All Symphony, Concerto, And
 Choir Concerts |
```

```
+-------+--------------------------------------------
+------------------------------------------+
```

INITCAP 関数が先頭文字以外の大文字を大文字として保存しないことを示すには、次の例を使用します。例えば、文字列 `MLB` は `Mlb` になります。

```
SELECT INITCAP(catname)
FROM category
ORDER BY catname;

+-----------+
|  initcap  |
+-----------+
| Classical |
| Jazz      |
| Mlb       |
| Mls       |
| Musicals  |
| Nba       |
| Nfl       |
| Nhl       |
| Opera     |
| Plays     |
| Pop       |
+-----------+
```

スペース関数以外の英数字以外の文字を単語の区切り文字として使用するには、次の例を使用します。各文字列のいくつかの文字が大文字になります。

```
SELECT email, INITCAP(email)
FROM users
ORDER BY userid DESC LIMIT 5;

+------------------------------------+------------------------------------+
|               email                |              initcap               |
+------------------------------------+------------------------------------+
| urna.Ut@egetdictumplacerat.edu     | Urna.Ut@Egetdictumplacerat.Edu     |
| nibh.enim@egestas.ca               | Nibh.Enim@Egestas.Ca               |
| in@Donecat.ca                      | In@Donecat.Ca                      |
| sodales@blanditviverraDonec.ca     | Sodales@Blanditviverradonec.Ca     |
| sociis.natoque.penatibus@vitae.org | Sociis.Natoque.Penatibus@Vitae.Org |
+------------------------------------+------------------------------------+
```

LEFT 関数および RIGHT 関数

これらの関数は、文字列の左端または右端にある指定数の文字を返します。

number はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。

構文

```
LEFT( string,  integer )

RIGHT( string,  integer )
```

引数

string

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 文字列に評価される式。

integer

正の整数。

戻り型

VARCHAR

例

次の例では、TICKIT サンプルデータベースの EVENT テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

イベント ID が 1000～1005 であるイベント名の左端 5 文字および右端 5 文字を返すには、次の例を使用します。

```
SELECT eventid, eventname,
LEFT(eventname,5) AS left_5,
RIGHT(eventname,5) AS right_5
FROM event
WHERE eventid BETWEEN 1000 AND 1005
ORDER BY 1;
```

```
+---------+---------------+--------+---------+
| eventid |   eventname   | left_5 | right_5 |
+---------+---------------+--------+---------+
|    1000 | Gypsy         | Gypsy  | Gypsy   |
|    1001 | Chicago       | Chica  | icago   |
|    1002 | The King and I | The K  | and I   |
|    1003 | Pal Joey      | Pal J  | Joey    |
|    1004 | Grease        | Greas  | rease   |
|    1005 | Chicago       | Chica  | icago   |
+---------+---------------+--------+---------+
```

LEN 関数

指定された文字列の長さを文字列数として返します。

構文

LEN は LENGTH 関数、CHAR_LENGTH 関数、CHARACTER_LENGTH 関数、および TEXTLEN 関数 のシノニムです。

```
LEN(expression)
```

引数

expression

CHAR 文字列、VARCHAR 文字列、VARBYTE 式、あるいは CHAR、VARCHAR、または VARBYTE 型に暗黙的に評価される式。

戻り型

INTEGER

LEN 関数は、入力文字列の文字数を示す整数を返します。

入力が文字列の場合、LEN 関数は、マルチバイト文字列のバイト数ではなく、この文字列の実際の文字数を返します。例えば、4 バイトの中国語文字を 3 つ保存するためには、VARCHAR(12) 列が必要です。LEN 関数は、その文字列に対して 3 を返します。文字列の長さをバイト単位で取得するには、OCTET_LENGTH関数を使用します。

使用に関する注意事項

式が CHAR 文字列である場合、末尾のスペースはカウントされません。

式が VARCHAR 文字列である場合、末尾のスペースはカウントされます。

例

文字列 français のバイト数および文字数を返すには、次の例を使用します。

```
SELECT OCTET_LENGTH('français'),
LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
+--------------+-----+
```

OCTET_LENGTH 関数を使用せずに、文字列 français のバイト数および文字数を返すには、次の例を使用します。詳細については、「CAST 関数」を参照してください。

```
SELECT LEN(CAST('français' AS VARBYTE)) as bytes, LEN('français');

+-------+-----+
| bytes | len |
+-------+-----+
|     9 |   8 |
+-------+-----+
```

末尾にスペースのない文字列 cat、末尾にスペースが 3 つある cat 、末尾にスペースが 3 つあり、長さ 6 の CHAR にキャストされる cat 、末尾にスペースが 3 つあり、長さ 6 の VARCHAR にキャストされる cat の文字数を返すには、次の例を使用します。この関数は CHAR 文字列の末尾のスペースをカウントしませんが、VARCHAR 文字列の末尾のスペースはカウントすることに注意してください。

```
SELECT LEN('cat'), LEN('cat   '), LEN(CAST('cat   ' AS CHAR(6))) AS len_char,
 LEN(CAST('cat   ' AS VARCHAR(6))) AS len_varchar;

+-----+-----+----------+-------------+
| len | len | len_char | len_varchar |
+-----+-----+----------+-------------+
```

```
|   3 |   6 |         3 |            6 |
+-----+-----+----------+-------------+
```

次の例では、TICKIT サンプルデータベースの VENUE テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

VENUE テーブル内で最長の venue 名を 10 個返すには、次の例を使用します。

```
SELECT venuename, LEN(venuename)
FROM venue
ORDER BY 2 DESC, 1
LIMIT 10;

+-----------------------------------------+-----+
|                venuename                | len |
+-----------------------------------------+-----+
| Saratoga Springs Performing Arts Center |  39 |
| Lincoln Center for the Performing Arts  |  38 |
| Nassau Veterans Memorial Coliseum       |  33 |
| Jacksonville Municipal Stadium          |  30 |
| Rangers BallPark in Arlington           |  29 |
| University of Phoenix Stadium           |  29 |
| Circle in the Square Theatre            |  28 |
| Hubert H. Humphrey Metrodome            |  28 |
| Oriole Park at Camden Yards             |  27 |
| Dick's Sporting Goods Park              |  26 |
+-----------------------------------------+-----+
```

LENGTH 関数

LEN 関数のシノニム。

「LEN 関数」を参照してください。

LOWER 関数

文字列を小文字に変換します。LOWER は、UTF-8 マルチバイト文字に対応しています (1 文字につき最大で 4 バイトまで)。

構文

```
LOWER(string)
```

引数

string

VARCHAR 型に評価される VARCHAR 文字列または式。

戻り型

string

LOWER 関数は、入力文字列と同じデータ型の文字列を返します。例えば、入力が CHAR 文字列の場合、この関数は CHAR 文字列を返します。

例

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATGROUP 列の文字列 VARCHAR を小文字に変換するには、次の例を使用します。

```
SELECT catname, LOWER(catname) FROM category ORDER BY 1,2;

+-----------+-----------+
|  catname  |   lower   |
+-----------+-----------+
| Classical | classical |
| Jazz      | jazz      |
| MLB       | mlb       |
| MLS       | mls       |
| Musicals  | musicals  |
| NBA       | nba       |
| NFL       | nfl       |
| NHL       | nhl       |
| Opera     | opera     |
| Plays     | plays     |
| Pop       | pop       |
+-----------+-----------+
```

LPAD 関数および RPAD 関数

これらの関数は、指定された長さに基づいて、文字列の前または後に文字を付加します。

構文

```
LPAD(string1, length, [ string2 ])
```

```
RPAD(string1, length, [ string2 ])
```

引数

string1

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

length

関数の結果の長さを定義する整数。文字列の長さはバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。指定された長さより string1 が長い場合は、(右側が) 切り捨てられます。length がゼロまたは負の数値である場合は、関数の結果が空の文字列になります。

string2

(オプション) string1 の前または後に付加する 1 つ以上の文字。この引数が指定されなかった場合は、スペースが使用されます。

戻り型

VARCHAR

例

次の例では、TICKIT サンプルデータベースの EVENT テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

指定された一連のイベント名を切り捨てて 20 文字にして、20 文字に満たない名前の前にはスペースを付加するには、次の例を使用します。

```
SELECT LPAD(eventname, 20) FROM event
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         lpad         |
+----------------------+
```

```
|              Salome |
|        Il Trovatore |
|       Boris Godunov |
|     Gotterdammerung |
|La Cenerentola (Cind |
+-----------------------+
```

上記と同じ一連のイベント名を切り捨てて 20 文字にするが、20 文字に満たない名前の後には 0123456789 を付加するには、次の例を使用します。

```
SELECT RPAD(eventname, 20,'0123456789') FROM event
WHERE eventid BETWEEN 1 AND 5 ORDER BY 1;

+----------------------+
|         rpad         |
+----------------------+
| Boris Godunov0123456 |
| Gotterdammerung01234 |
| Il Trovatore01234567 |
| La Cenerentola (Cind |
| Salome01234567890123 |
+----------------------+
```

LTRIM 関数

文字列の先頭から文字を切り捨てます。トリム文字リスト内の文字のみを含む最長の文字列を削除します。入力文字列にトリム文字がないときには、トリミングは完了です。

構文

```
LTRIM( string [, trim_chars] )
```

引数

string

トリミングする文字列列、式、または文字列リテラル。

trim_chars

文字列の先頭からトリミングする文字を表す、文字列の列、式、または文字列リテラル。指定しなかった場合、スペースがトリム文字として使用されます。

戻り型

LTRIM 関数は、入力文字列のデータ型 (CHAR または VARCHAR) と同じデータ型の文字列を返します。

例

次の例は、listime 列から年をトリミングします。文字列リテラル '2008-' のトリム文字は、左からトリミングされる文字を示します。トリム文字 '028-' を使用した場合も、同じ結果が得られます。

```
select listid, listtime, ltrim(listtime, '2008-')
from listing
order by 1, 2, 3
limit 10;

listid |      listtime       |     ltrim
-------+---------------------+----------------
     1 | 2008-01-24 06:43:29 | 1-24 06:43:29
     2 | 2008-03-05 12:25:29 | 3-05 12:25:29
     3 | 2008-11-01 07:35:33 | 11-01 07:35:33
     4 | 2008-05-24 01:18:37 | 5-24 01:18:37
     5 | 2008-05-17 02:29:11 | 5-17 02:29:11
     6 | 2008-08-15 02:08:13 | 15 02:08:13
     7 | 2008-11-15 09:38:15 | 11-15 09:38:15
     8 | 2008-11-09 05:07:30 | 11-09 05:07:30
     9 | 2008-09-09 08:03:36 | 9-09 08:03:36
    10 | 2008-06-17 09:44:54 | 6-17 09:44:54
```

LTRIM は、文字列の先頭にあるとき、trim_chars の文字をすべて削除します。次の例は、文字 'C'、'D'、および 'G' が VENUENAME の先頭にある場合 (VARCHAR 列) 、これらの文字を切り捨てます。

```
select venueid, venuename, ltrim(venuename, 'CDG')
from venue
where venuename like '%Park'
order by 2
limit 7;

venueid | venuename                    | btrim
--------+------------------------------+--------------------------
    121 | ATT Park                     | ATT Park
```

```
     109 | Citizens Bank Park           | itizens Bank Park
     102 | Comerica Park                | omerica Park
       9 | Dick's Sporting Goods Park | ick's Sporting Goods Park
      97 | Fenway Park                  | Fenway Park
     112 | Great American Ball Park    | reat American Ball Park
     114 | Miller Park                  | Miller Park
```

次の例では、venueid 列から取得されたしたトリム文字 2 を使用します。

```
select ltrim('2008-01-24 06:43:29', venueid)
from venue where venueid=2;

ltrim
------------------
008-01-24 06:43:29
```

次の例では、2 が '0' トリム文字の前にあるため、文字はトリミングされません。

```
select ltrim('2008-01-24 06:43:29', '0');

ltrim
-------------------
2008-01-24 06:43:29
```

次の例では、デフォルトのスペーストリム文字を使用して、文字列の先頭から 2 つのスペースをトリミングします。

```
select ltrim('  2008-01-24 06:43:29');

ltrim
-------------------
2008-01-24 06:43:29
```

OCTETINDEX 関数

OCTETINDEX 関数は、文字列内の部分文字列の位置をバイト数として返します。

構文

```
OCTETINDEX(substring, string)
```

引数

substring

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

string

CHAR 文字列、VARCHAR 文字列、あるいは CHAR または VARCHAR 型に暗黙的に評価される式。

戻り型

INTEGER

OCTETINDEX 関数は、string 内の substring の位置に対応する INTEGER 値をバイト数として返します。ここで、string の最初の文字は 1 としてカウントされます。string にマルチバイト文字が含まれていない場合、結果は CHARINDEX 関数の結果と等しくなります。string に substring が含まれない場合、この関数は 0 を返します。substring が空の場合、この関数は 1 を返します。

例

文字列 Amazon Redshift のサブ文字列 q の位置を返すには、次の例を使用します。substring は string に含まれていないため、この例では 0 を返します。

```
SELECT OCTETINDEX('q', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          0 |
+------------+
```

文字列 Amazon Redshift の空のサブ文字列の位置を返すには、次の例を使用します。substring は空であるため、この例では 1 を返します。

```
SELECT OCTETINDEX('', 'Amazon Redshift');

+------------+
| octetindex |
```

```
+------------+
|          1 |
+------------+
```

文字列 Amazon Redshift のサブ文字列 Redshift の位置を返すには、次の例を使用します。この例では、substring が string の 8 バイト目から始まるため、8 を返します。

```
SELECT OCTETINDEX('Redshift', 'Amazon Redshift');

+------------+
| octetindex |
+------------+
|          8 |
+------------+
```

文字列 Amazon Redshift のサブ文字列 Redshift の位置を返すには、次の例を使用します。この例では、string の最初の 6 文字が 2 バイト文字であるため、21 を返します。

```
SELECT OCTETINDEX('Redshift', 'Άμαζον Amazon Redshift');

+------------+
| octetindex |
+------------+
|         21 |
+------------+
```

OCTET_LENGTH 関数

指定された文字列の長さをバイト数として返します。

構文

```
OCTET_LENGTH(expression)
```

引数

expression

CHAR 文字列、VARCHAR 文字列、VARBYTE 式、あるいは CHAR、VARCHAR、または VARBYTE 型に暗黙的に評価される式。

戻り型

INTEGER

OCTET_LENGTH 関数は、入力文字列のバイト数を示す整数を返します。

入力が文字列の場合、LEN関数は、マルチバイト文字列のバイト数ではなく、この文字列の実際の文字数を返します。例えば、4 バイトの中国語文字を 3 つ保存するためには、VARCHAR(12) 列が必要です。OCTET_LENGTH 関数はその文字の 12 を返し、LEN 関数は同じ文字列に対して 3 を返します。

使用に関する注意事項

式が CHAR 文字列である場合、この関数は CHAR 文字列の長さを返します。例えば、CHAR(6) 入力の出力は CHAR(6) です。

式が VARCHAR 文字列である場合、末尾のスペースはカウントされます。

例

末尾にスペースが 3 つある文字列 francais が CHAR および VARCHAR 型にキャストされる場合に、バイト数を返すには、次の例を使用します。詳細については、「CAST 関数」を参照してください。

```
SELECT OCTET_LENGTH(CAST('francais   ' AS CHAR(15))) AS octet_length_char,
 OCTET_LENGTH(CAST('francais   ' AS VARCHAR(15))) AS octet_length_varchar;

+-------------------+----------------------+
| octet_length_char | octet_length_varchar |
+-------------------+----------------------+
|                15 |                   11 |
+-------------------+----------------------+
```

文字列 français のバイト数および文字数を返すには、次の例を使用します。

```
SELECT OCTET_LENGTH('français'), LEN('français');

+--------------+-----+
| octet_length | len |
+--------------+-----+
|            9 |   8 |
```

```
+--------------+-----+
```

文字列 `français` が VARBYTE にキャストされる場合にバイト数を返すには、次の例を使用します。

```
SELECT OCTET_LENGTH(CAST('français' AS VARBYTE));

+--------------+
| octet_length |
+--------------+
|            9 |
+--------------+
```

POSITION 関数

文字列内の指定されたサブ文字列の位置を返します。

同様の関数については、「CHARINDEX 関数」および「STRPOS 関数」を参照してください。

構文

```
POSITION(substring IN string )
```

引数

substring

　string 内を検索するサブ文字列。

string

　検索する文字列または列。

戻り型

POSITION 関数は、サブ文字列の位置 (0 ではなく 1 から始まる) に対応する `INTEGER` を返します。position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。文字列内のサブ文字列がない場合、POSITION は `0` を返します。

例

dog という語の中での文字列 `fish` の位置を返すには、次の例を使用します。

```
SELECT POSITION('fish' IN 'dog');

+-----------+
|  position |
+-----------+
|         0 |
+-----------+
```

dogfish という語の中での文字列 fish の位置を返すには、次の例を使用します。

```
SELECT POSITION('fish' IN 'dogfish');

+-----------+
|  position |
+-----------+
|         4 |
+-----------+
```

以下の例は、TICKIT サンプルデータベースの SALES テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブル内でコミッションが 999.00 を上回る販売取引の数を返すには、次の例を使用します。このコマンドは、999.00 を超えるコミッションをカウントします。これは、小数点以下の桁数がコミッション値の先頭から 4 桁より大きいかどうかを調べることでカウントされます。

```
SELECT DISTINCT POSITION('.' IN commission), COUNT (POSITION('.' IN commission))
FROM sales
WHERE POSITION('.' IN commission) > 4
GROUP BY POSITION('.' IN commission)
ORDER BY 1,2;

+-----------+-------+
|  position | count |
+-----------+-------+
|         5 |   629 |
+-----------+-------+
```

QUOTE_IDENT 関数

QUOTE_IDENT 関数は、指定された文字列の先頭と末尾を二重引用符で囲んだ文字列として返します。この関数の出力は、SQL ステートメントで識別子として使用できます。二重引用符が埋め込まれている場合、この関数は適宜これを二重化します。

QUOTE_IDENT は、文字列が非識別子文字あるいは小文字になってしまう場合に、有効な識別子を作成するために必要な場所のみに二重引用符を追加します。いつでも一重引用符の文字列に戻るには、QUOTE_LITERAL を使用します。

構文

```
QUOTE_IDENT(string)
```

引数

string

CHAR または VARCHAR 文字列。

戻り型

QUOTE_IDENT 関数は、入力 string と同じ型を返します。

例

二重引用符で囲んで文字列 "CAT" を返すには、次の例を使用します。

```
SELECT QUOTE_IDENT('"CAT"');

+-------------+
| quote_ident |
+-------------+
| """CAT"""   |
+-------------+
```

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATNAME 列を引用符で囲んで返すには、次の例を使用します。

```
SELECT catid, QUOTE_IDENT(catname)
FROM category
ORDER BY 1,2;

+-------+-------------+
| catid | quote_ident |
+-------+-------------+
|     1 | "MLB"       |
|     2 | "NHL"       |
|     3 | "NFL"       |
|     4 | "NBA"       |
|     5 | "MLS"       |
|     6 | "Musicals"  |
|     7 | "Plays"     |
|     8 | "Opera"     |
|     9 | "Pop"       |
|    10 | "Jazz"      |
|    11 | "Classical" |
+-------+-------------+
```

QUOTE_LITERAL 関数

QUOTE_LITERAL 関数は、指定された文字列を一重引用符付き文字列として返します。これにより、SQL ステートメントで文字列リテラルとして使用できます。入力パラメータが数値である場合、QUOTE_LITERAL はそれを文字列として扱います。埋め込まれている一重引用符およびバックスラッシュは適宜、二重引用符に変更されます。

構文

```
QUOTE_LITERAL(string)
```

引数

string

CHAR または VARCHAR 文字列。

戻り型

QUOTE_LITERAL 関数は、入力 string と同じデータ型である CHAR または VARCHAR の文字列を返します。

例

一重引用符を付けて文字列 ''CAT'' を返すには、次の例を使用します。

```
SELECT QUOTE_LITERAL('''CAT''');

+---------------+
| quote_literal |
+---------------+
| '''CAT'''     |
+---------------+
```

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATNAME 列を一重引用符で囲んで返すには、次の例を使用します。

```
SELECT catid, QUOTE_LITERAL(catname)
FROM category
ORDER BY 1,2;

+-------+---------------+
| catid | quote_literal |
+-------+---------------+
|     1 | 'MLB'         |
|     2 | 'NHL'         |
|     3 | 'NFL'         |
|     4 | 'NBA'         |
|     5 | 'MLS'         |
|     6 | 'Musicals'    |
|     7 | 'Plays'       |
|     8 | 'Opera'       |
|     9 | 'Pop'         |
|    10 | 'Jazz'        |
|    11 | 'Classical'   |
+-------+---------------+
```

CATID 列を一重引用符で囲んで返すには、次の例を使用します。

```
SELECT QUOTE_LITERAL(catid), catname
FROM category
ORDER BY 1,2;
```

```
+---------------+-----------+
| quote_literal |  catname  |
+---------------+-----------+
| '1'           | MLB       |
| '10'          | Jazz      |
| '11'          | Classical |
| '2'           | NHL       |
| '3'           | NFL       |
| '4'           | NBA       |
| '5'           | MLS       |
| '6'           | Musicals  |
| '7'           | Plays     |
| '8'           | Opera     |
| '9'           | Pop       |
+---------------+-----------+
```

REGEXP_COUNT 関数

文字列で正規表現パターンを検索し、この指定されたパターンが文字列内に出現する回数を示す整数を返します。一致がない場合、この関数は `0` を返します。正規表現の詳細については、Wikipedia の「POSIX 演算子」と「正規表現」を参照してください。

構文

```
REGEXP_COUNT( source_string, pattern [, position [, parameters ] ] )
```

引数

source_string

CHAR または VARCHAR 文字列。

pattern

正規表現パターンを表す UTF-8 文字列リテラル。詳細については、「POSIX 演算子」を参照してください。

position

(オプション) 検索を開始する source_string 内の位置を示す正の `INTEGER`。position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。デフォルト: 1。position が 1 より小さい場合、source_string の最初の文字から検索が開始されます。position が source_string の文字数より大きい場合、結果は `0` になります。

parameters

(オプション) 関数がパターンとどのように一致するかを示す 1 つ以上のリテラル文字列。取り得る値には以下のものがあります。

- c – 大文字と小文字を区別する一致を実行します。デフォルトでは大文字と小文字を区別するマッチングを使用します。
- i – 大文字と小文字を区別しない一致を実行します。
- p – Perl 互換正規表現 (PCRE) 言語でパターンを解釈します。PCRE の詳細については、Wikipedia の「Perl 互換の正規表現」を参照してください。

戻り型

INTEGER

例

3 文字のシーケンスが出現する回数をカウントするには、次の例を使用します。

```
SELECT REGEXP_COUNT('abcdefghijklmnopqrstuvwxyz', '[a-z]{3}');

+--------------+
| regexp_count |
+--------------+
|            8 |
+--------------+
```

大文字と小文字を区別しない一致を使用して、文字列 FOX の出現をカウントするには、次の例を使用します。

```
SELECT REGEXP_COUNT('the fox', 'FOX', 1, 'i');

+--------------+
| regexp_count |
+--------------+
|            1 |
+--------------+
```

PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索するには、次の例を使用します。この例では、PCRE で特定の先読みの意味を持つ ?= 演算子を

使用します。この例では、大文字と小文字を区別して、このような単語の出現回数をカウントします。

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'p');

+--------------+
| regexp_count |
+--------------+
|            2 |
+--------------+
```

PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索するには、次の例を使用します。これは、PCRE で特定の意味を持つ ?= 演算子を使用します。この例では、このような単語の出現回数をカウントしますが、大文字と小文字を区別しない一致結果を使用する点で前の例とは異なります。

```
SELECT REGEXP_COUNT('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 'ip');

+--------------+
| regexp_count |
+--------------+
|            3 |
+--------------+
```

次の例では、TICKIT サンプルデータベースの USERS テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

最上位ドメイン名が org または edu である回数をカウントするには、次の例を使用します。

```
SELECT email, REGEXP_COUNT(email,'@[^.]*\.(org|edu)') FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------+--------------+
|                     email                     | regexp_count |
+-----------------------------------------------+--------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |            1 |
| Suspendisse.tristique@nonnisiAenean.edu       |            1 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |            0 |
| sed@lacusUtnec.ca                             |            0 |
```

```
+---------------------------------------------------+---------------+
```

REGEXP_INSTR 関数

文字列で正規表現パターンを検索し、一致するサブ文字列の開始位置を示す整数を返します。一致がない場合、この関数は 0 を返します。REGEXP_INSTR は 関数に似ていますが、文字列で正規表現パターンを検索することができます。正規表現の詳細については、Wikipedia の「POSIX 演算子」と「正規表現」を参照してください。

構文

```
REGEXP_INSTR( source_string, pattern [, position [, occurrence] [, option [, parameters
 ] ] ] ] )
```

引数

source_string

検索する文字列式 (列名など)。

pattern

正規表現パターンを表す UTF-8 文字列リテラル。詳細については、「POSIX 演算子」を参照してください。

position

(オプション) 検索を開始する source_string 内の位置を示す正の INTEGER。position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。デフォルト: 1。position が 1 より小さい場合、source_string の最初の文字から検索が開始されます。position が source_string の文字数より大きい場合、結果は 0 になります。

occurrence

(オプション) このパターンのどの出現を使用するかを示す正の INTEGER。REGEXP_INSTR は、最初の *occurrence*-1 の一致をスキップします。デフォルト: 1。occurrence が 1 未満、または source_string の文字数を超える場合、検索は無視され、結果は 0 となります。

option

(オプション) 一致の先頭文字の戻り位置 (0)、または続く一致の末尾の最初の文字位置 (1) のどちらかを示す値。ゼロ以外の値は 1 と同じです。デフォルト値は 0 です。

parameters

(オプション) 関数がパターンとどのように一致するかを示す 1 つ以上のリテラル文字列。取り得る値には以下のものがあります。

- c – 大文字と小文字を区別する一致を実行します。デフォルトでは大文字と小文字を区別するマッチングを使用します。
- i – 大文字と小文字を区別しない一致を実行します。
- e – 部分式を使用して部分文字列を抽出します。

  パターンに部分式が含まれる場合、REGEXP_INSTR は最初の部分式をパターンで使用して部分文字列を一致させます。REGEXP_INSTR は最初の部分式のみを考慮します。追加の部分式は無視されます。パターンに部分式がない場合、REGEXP_INSTR は「e」パラメータを無視します。
- p – Perl 互換正規表現 (PCRE) 言語でパターンを解釈します。PCRE の詳細については、Wikipedia の「Perl 互換の正規表現」を参照してください。

戻り型

整数

例

次の例では、TICKIT サンプルデータベースの USERS テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

ドメイン名を開始する @ 文字を検索し、最初の一致の開始位置を返すには、次の例を使用します。

```
SELECT email, REGEXP_INSTR(email, '@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------+---------------+
|                     email                     | regexp_instr  |
+-----------------------------------------------+---------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |            21 |
| Suspendisse.tristique@nonnisiAenean.edu       |            22 |
| amet.faucibus.ut@condimentumegetvolutpat.ca   |            17 |
| sed@lacusUtnec.ca                             |             4 |
+-----------------------------------------------+---------------+
```

Center という単語のバリエーションを検索し、最初の一致の開始位置を返すには、次の例を使用します。

```
SELECT venuename, REGEXP_INSTR(venuename,'[cC]ent(er|re)$')
FROM venue
WHERE REGEXP_INSTR(venuename,'[cC]ent(er|re)$') > 0
ORDER BY venueid LIMIT 4;

+-----------------------+--------------+
|       venuename       | regexp_instr |
+-----------------------+--------------+
| The Home Depot Center |           16 |
| Izod Center           |            6 |
| Wachovia Center       |           10 |
| Air Canada Centre     |           12 |
+-----------------------+--------------+
```

大文字と小文字を区別しない一致ロジックを使用して、文字列 FOX が最初に出現する開始位置を検索するには、次の例を使用します。

```
SELECT REGEXP_INSTR('the fox', 'FOX', 1, 1, 0, 'i');

+--------------+
| regexp_instr |
+--------------+
|            5 |
+--------------+
```

PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索するには、次の例を使用します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。この例では、2 番目の単語の開始位置を見つけます。

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'p');

+--------------+
| regexp_instr |
+--------------+
|           21 |
+--------------+
```

PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索するには、次の例を使用します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。この例では、2 番目の単語の開始位置を検索しますが、大文字と小文字を区別しない一致結果を使用する点で前の例とは異なります。

```
SELECT REGEXP_INSTR('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 0, 'ip');

+--------------+
| regexp_instr |
+--------------+
|           15 |
+--------------+
```

REGEXP_REPLACE 関数

文字列で正規表現パターンを検索し、このパターンのすべての出現を特定の文字列に置き換えます。REGEXP_REPLACE は REPLACE 関数 関数に似ていますが、文字列で正規表現パターンを検索することができます。正規表現の詳細については、Wikipedia の「POSIX 演算子」と「正規表現」を参照してください。

REGEXP_REPLACE は、TRANSLATE 関数や REPLACE 関数 と似ています。ただし、TRANSLATE は複数の単一文字置換を行い、REPLACE は 1 つの文字列全体を別の文字列に置換しますが、REGEXP_REPLACE を使用すると正規表現パターンの文字列を検索できます。

構文

```
REGEXP_REPLACE( source_string, pattern [, replace_string [ , position [, parameters
 ] ] ] )
```

引数

source_string

検索する CHAR または VARCHAR 文字列式 (列名など)。

pattern

正規表現パターンを表す UTF-8 文字列リテラル。詳細については、「POSIX 演算子」を参照してください。

replace_string

(オプション) パターンのすべての出現に置き換わる列名などの `CHAR` または `VARCHAR` 文字列式。デフォルトは空の文字列 ("") です。

position

(オプション) 検索を開始する source_string 内の位置を示す正の整数。position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。デフォルト: 1。position が 1 より小さい場合、source_string の最初の文字から検索が開始されます。position が source_string の文字数より大きい場合、結果は source_string になります。

parameters

(オプション) 関数がパターンとどのように一致するかを示す 1 つ以上のリテラル文字列。取り得る値には以下のものがあります。

- c – 大文字と小文字を区別する一致を実行します。デフォルトでは大文字と小文字を区別するマッチングを使用します。
- i – 大文字と小文字を区別しない一致を実行します。
- p – Perl 互換正規表現 (PCRE) 言語でパターンを解釈します。PCRE の詳細については、Wikipedia の「Perl 互換の正規表現」を参照してください。

戻り型

VARCHAR

pattern または replace_string のいずれかが `NULL` の場合、戻り値は `NULL` を返します。

例

大文字と小文字を区別しない一致を使用して、値 `quick brown fox` 内の文字列 `FOX` のすべての出現を置き換えるには、次の例を使用します。

```
SELECT REGEXP_REPLACE('the fox', 'FOX', 'quick brown fox', 1, 'i');

+---------------------+
|   regexp_replace    |
+---------------------+
| the quick brown fox |
+---------------------+
```

次の例では、PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。そのような単語が出現するたびに値 [hidden] に置き換えるには、次の例を使用します。

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'p');

+-------------------------------+
|        regexp_replace         |
+-------------------------------+
| [hidden] plain A1234 [hidden] |
+-------------------------------+
```

次の例では、PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。そのような単語が出現するたびに値 [hidden] に置き換えるが、大文字と小文字を区別しない一致を使用することが前の例とは異なる場合は、次の例を使用します。

```
SELECT REGEXP_REPLACE('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 '[hidden]', 1, 'ip');

+----------------------------------+
|          regexp_replace          |
+----------------------------------+
| [hidden] plain [hidden] [hidden] |
+----------------------------------+
```

次の例では、TICKIT サンプルデータベースの USERS テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

E メールアドレスから @ とドメイン名を削除するには、次の例を使用します。

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.(org|gov|com|edu|ca)$')
FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------+-----------------------+
|                     email                     |    regexp_replace     |
+-----------------------------------------------+-----------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu | Etiam.laoreet.libero  |
| Suspendisse.tristique@nonnisiAenean.edu       | Suspendisse.tristique |
```

```
| amet.faucibus.ut@condimentumegetvolutpat.ca    | amet.faucibus.ut       |
| sed@lacusUtnec.ca                              | sed                    |
+-----------------------------------------------+-----------------------+
```

E メールアドレスのドメイン名を `internal.company.com` に置き換えるには、次の例を使用します。

```
SELECT email, REGEXP_REPLACE(email, '@.*\\.[[:alpha:]]{2,3}','@internal.company.com')
FROM users
ORDER BY userid LIMIT 4;

+-----------------------------------------------
+--------------------------------------------+
|                      email                    |                regexp_replace
      |
+-----------------------------------------------
+--------------------------------------------+
| Etiam.laoreet.libero@sodalesMaurisblandit.edu |
 Etiam.laoreet.libero@internal.company.com  |
| Suspendisse.tristique@nonnisiAenean.edu       |
 Suspendisse.tristique@internal.company.com |
| amet.faucibus.ut@condimentumegetvolutpat.ca   | amet.faucibus.ut@internal.company.com
      |
| sed@lacusUtnec.ca                             | sed@internal.company.com
      |
+-----------------------------------------------
+--------------------------------------------+
```

REGEXP_SUBSTR 関数

正規表現パターンで検索して、文字列から文字を返します。REGEXP_SUBSTR は SUBSTRING 関数 関数に似ていますが、文字列で正規表現パターンを検索することができます。この関数が正規表現を文字列内のどの文字とも一致させることができない場合、空の文字列を返します。正規表現の詳細については、Wikipedia の「POSIX 演算子」と「正規表現」を参照してください。

構文

```
REGEXP_SUBSTR( source_string, pattern [, position [, occurrence [, parameters ] ] ] )
```

引数

source_string

検索する文字列式。

pattern

正規表現パターンを表す UTF-8 文字列リテラル。詳細については、「POSIX 演算子」を参照してください。

position

検索を開始する source_string 内の位置を示す正の整数。位置はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。デフォルトは 1 です。position が 1 より小さい場合、source_string の最初の文字から検索が開始されます。position が source_string の文字数より大きい場合、結果は空の文字列 ("") になります。

occurrence

このパターンのどの出現を使用するかを示す正の整数。REGEXP_SUBSTR は、最初の出現 - 1 一致をスキップします。デフォルトは 1 です。出現が 1 未満、またはソース文字列の文字数以上の場合、検索は無視され、結果は NULL となります。

パラメータ

関数がパターンと一致するかを示す 1 つ以上のリテラル文字列。取り得る値には以下のものがあります。

- c – 大文字と小文字を区別する一致を実行します。デフォルトでは大文字と小文字を区別するマッチングを使用します。
- i – 大文字と小文字を区別しない一致を実行します。
- e – 部分式を使用して部分文字列を抽出します。

  パターンに部分式が含まれる場合、REGEXP_SUBSTR は最初の部分式をパターンで使用して部分文字列を一致させます。部分式は、かっこで囲まれたパターン内の式です。例えば、`'This is a (\\w+)'` というパターンでは、最初の式と `'This is a '` という文字列とそれに続く単語が一致します。e パラメータを指定した REGEXP_SUBSTR は、パターンを返す代わりに、部分式の中の文字列だけを返します。

  REGEXP_SUBSTR は最初の部分式のみを考慮します。追加の部分式は無視されます。パターンに部分式がない場合、REGEXP_SUBSTR は「e」パラメータを無視します。

- p – Perl 互換正規表現 (PCRE) 言語でパターンを解釈します。PCRE の詳細については、Wikipedia の「Perl 互換の正規表現」を参照してください。

戻り型

VARCHAR

例

次の例では、メールアドレスの @ 文字とドメイン拡張の間の分が返されます。クエリされる users データは Amazon Redshift のサンプルデータからのものです。詳細については、「サンプルデータベース」を参照してください。

```
SELECT email, regexp_substr(email,'@[^.]*')
FROM users
ORDER BY userid LIMIT 4;

                     email                     |      regexp_substr
-----------------------------------------------+--------------------------
 Suspendisse.tristique@nonnisiAenean.edu       | @nonnisiAenean
 amet.faucibus.ut@condimentumegetvolutpat.ca   | @condimentumegetvolutpat
 sed@lacusUtnec.ca                             | @lacusUtnec
 Cum@accumsan.com                              | @accumsan
```

次の例では、大文字と小文字を区別しない一致を使用して、文字列 FOX の最初の出現に対応する入力の部分を返します。

```
SELECT regexp_substr('the fox', 'FOX', 1, 1, 'i');

 regexp_substr
---------------
 fox
```

次の例では、大文字と小文字を区別しない一致を使用して、文字列 FOX の 2 回目の出現に対応する入力部分を返します。2 回目の出現がないため、結果は NULL (空) になります。

```
SELECT regexp_substr('the fox', 'FOX', 1, 2, 'i');

 regexp_substr
---------------
```

次の例では、小文字で始まる入力の最初の部分を返します。これは、同じ SELECT ステートメントで c パラメータを指定しない場合と機能的には同じです。

```
SELECT regexp_substr('THE SECRET CODE IS THE LOWERCASE PART OF 1931abc0EZ.', '[a-z]+',
 1, 1, 'c');

 regexp_substr
---------------
 abc
```

次の例では、PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。この例では、2 番目の単語に対応する入力部分を返します。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'p');

 regexp_substr
---------------
 a1234
```

次の例では、PCRE 言語で記述されたパターンを使用して、少なくとも 1 つの数字と 1 つの小文字を含む単語を検索します。PCRE で特定の先読みの意味を持つ ?= 演算子を使用します。この例では、2 番目の単語に対応する入力部分を返しますが、大文字と小文字を区別しない一致結果を使用する点で前の例とは異なります。

```
SELECT regexp_substr('passwd7 plain A1234 a1234', '(?=[^ ]*[a-z])(?=[^ ]*[0-9])[^ ]+',
 1, 2, 'ip');

 regexp_substr
---------------
 A1234
```

次の例では、部分式を使用して、大文字と小文字を区別しないマッチングにより、パターン `'this is a (\\w+)'` と一致する 2 番目の文字列を見つけます。かっこ内の部分式を返します。

```
SELECT regexp_substr(
```

```
                 'This is a cat, this is a dog. This is a mouse.',
                 'this is a (\\w+)', 1, 2, 'ie');

 regexp_substr
---------------
 dog
```

REPEAT 関数

指定された回数だけ文字列を繰り返します。入力パラメータが数値である場合、REPEAT はそれを文字列として扱います。

REPLICATE 関数 のシノニム。

構文

```
REPEAT(string, integer)
```

引数

string

最初の入力パラメータは、繰り返す文字列です。

integer

2 番目のパラメータは、文字列を繰り返す回数を示す INTEGER です。

戻り型

VARCHAR

例

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATEGORY テーブル内の CATID 列の値を 3 回繰り返すには、次の例を使用します。

```
SELECT catid, REPEAT(catid,3)
FROM category
```

```
ORDER BY 1,2;

+-------+--------+
| catid | repeat |
+-------+--------+
|     1 |    111 |
|     2 |    222 |
|     3 |    333 |
|     4 |    444 |
|     5 |    555 |
|     6 |    666 |
|     7 |    777 |
|     8 |    888 |
|     9 |    999 |
|    10 | 101010 |
|    11 | 111111 |
+-------+--------+
```

REPLACE 関数

既存の文字列内の一連の文字をすべて、指定された他の文字に置き換えます。

REPLACE は、TRANSLATE 関数や REGEXP_REPLACE 関数 と似ています。ただし、TRANSLATE は複数の単一文字置換を行い、REGEXP_REPLACE を使用すると正規表現パターンの文字列を検索できますが、REPLACE は 1 つの文字列全体を別の文字列に置換します。

構文

```
REPLACE(string, old_chars, new_chars)
```

引数

string

検索する CHAR 型または VARCHAR 型の文字列

old_chars

置き換える CHAR または VARCHAR 型の文字列。

new_chars

old_string を置き換える新しい CHAR 型または VARCHAR 型の文字列。

戻り型

VARCHAR

old_chars または new_chars のいずれかが NULL の場合、戻り値は NULL です。

例

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATGROUP フィールド内の文字列 Shows を Theatre に変換するには、次の例を使用します。

```
SELECT catid, catgroup, REPLACE(catgroup, 'Shows', 'Theatre')
FROM category
ORDER BY 1,2,3;

+-------+----------+----------+
| catid | catgroup | replace  |
+-------+----------+----------+
|     1 | Sports   | Sports   |
|     2 | Sports   | Sports   |
|     3 | Sports   | Sports   |
|     4 | Sports   | Sports   |
|     5 | Sports   | Sports   |
|     6 | Shows    | Theatre  |
|     7 | Shows    | Theatre  |
|     8 | Shows    | Theatre  |
|     9 | Concerts | Concerts |
|    10 | Concerts | Concerts |
|    11 | Concerts | Concerts |
+-------+----------+----------+
```

REPLICATE 関数

REPEAT 関数のシノニム。

「REPEAT 関数」を参照してください。

REVERSE 関数

REVERSE 関数は、文字列に対して機能し、文字を逆順に返します。たとえば、reverse('abcde')は edcba を返します。この関数は、数値データ型と日付データ型に加

え、文字データ型に対しても機能します。ただしほとんどの場合、文字列に対して実用的な値が生成されます。

構文

```
REVERSE( expression )
```

引数

expression

文字反転のターゲットを表す文字、日付、タイムスタンプ、または数値のデータ型を使用した式。すべての式は、暗黙的に VARCHAR 文字列に変換されます。CHAR 文字列内の末尾の空白は無視されます。

戻り型

VARCHAR

例

次の例では、TICKIT サンプルデータベースの USERS テーブルと SALES テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

USERS テーブルから、5 つの異なる都市名およびそれらに対応する反転した名前を選択するには、次の例を使用します。

```
SELECT DISTINCT city AS cityname, REVERSE(cityname)
FROM users
ORDER BY city LIMIT 5;

+----------+----------+
| cityname | reverse  |
+----------+----------+
| Aberdeen | needrebA |
| Abilene  | enelibA  |
| Ada      | adA      |
| Agat     | tagA     |
| Agawam   | mawagA   |
+----------+----------+
```

文字列として変換された 5 つの販売 ID およびそれらに対応する反転した ID を選択するには、次の例を使用します。

```
SELECT salesid, REVERSE(salesid)
FROM sales
ORDER BY salesid DESC LIMIT 5;

+---------+---------+
| salesid | reverse |
+---------+---------+
|  172456 |  654271 |
|  172455 |  554271 |
|  172454 |  454271 |
|  172453 |  354271 |
|  172452 |  254271 |
+---------+---------+
```

RTRIM 関数

RTRIM 関数は、指定された一連の文字を文字列の末尾から切り捨てます。トリム文字リスト内の文字のみを含む最長の文字列を削除します。入力文字列にトリム文字がないときには、トリミングは完了です。

構文

```
RTRIM( string, trim_chars )
```

引数

string

    トリミングする文字列列、式、または文字列リテラル。

trim_chars

    文字列の末尾から切り捨てる文字を表す、文字列の列、式、または文字列リテラル。指定しなかった場合、スペースがトリム文字として使用されます。

戻り型

string 引数と同じデータ型の文字列。

例

次の例では、文字列 ' abc ' の先頭および末尾の空白を切り捨てます。

```
select '     abc    ' as untrim, rtrim('     abc    ') as trim;

untrim    | trim
----------+------
   abc    |    abc
```

次の例では、文字列 'xyzaxyzbxyzcxyz' から末尾の文字列 'xyz' を削除します。末尾にある 'xyz' は削除されましたが、文字列内部にあるその文字列は削除されません。

```
select 'xyzaxyzbxyzcxyz' as untrim,
rtrim('xyzaxyzbxyzcxyz', 'xyz') as trim;

     untrim      |   trim
-----------------+-----------
 xyzaxyzbxyzcxyz | xyzaxyzbxyzc
```

次の例では、trim_chars リスト 'tes' のいずれかの文字と一致する文字列 'setuphistorycassettes' の末尾の部分を削除します。入力文字列の末尾にある trim_chars リストに含まれていない別の文字の前に出現する t、e または s が削除されます。

```
SELECT rtrim('setuphistorycassettes', 'tes');

     rtrim
-----------------
 setuphistoryca
```

次の例は、文字 'Park' が存在する VENUENAME の末尾から、これらの文字を切り捨てます。

```
select venueid, venuename, rtrim(venuename, 'Park')
from venue
order by 1, 2, 3
limit 10;

venueid |         venuename          |          rtrim
--------+----------------------------+-------------------------
      1 | Toyota Park                | Toyota
      2 | Columbus Crew Stadium      | Columbus Crew Stadium
```

```
     3 | RFK Stadium                | RFK Stadium
     4 | CommunityAmerica Ballpark  | CommunityAmerica Ballp
     5 | Gillette Stadium           | Gillette Stadium
     6 | New York Giants Stadium    | New York Giants Stadium
     7 | BMO Field                  | BMO Field
     8 | The Home Depot Center      | The Home Depot Cente
     9 | Dick's Sporting Goods Park | Dick's Sporting Goods
    10 | Pizza Hut Park             | Pizza Hut
```

RTRIM は、文字 P、a、r、または k が VENUENAME の末尾にあるとき、それらをすべて削除することに注意してください。

SOUNDEX 関数

SOUNDEX 関数は、入力文字列の最初の文字と、指定した文字列の英語の発音を表す音の 3 桁のエンコードで構成される American Soundex 値を返します。例えば、`Smith` と `Smyth` は同じ Soundex 値を持ちます。

構文

```
SOUNDEX(string)
```

引数

string

American Soundex コード値に変換する `CHAR` または `VARCHAR` 文字列を指定します。

戻り型

VARCHAR(4)

使用に関する注意事項

SOUNDEX 関数は、a～z および A～Z を含む、英字のアルファベットで小文字と大文字の ASCII 文字のみを変換します。SOUNDEX 関数は、他の文字を無視します。SOUNDEX は、スペースで区切られた複数の単語の文字列に対して、単一の Soundex 値を返します。

```
SELECT SOUNDEX('AWS Amazon');

+---------+
```

```
| soundex |
+---------+
| A252    |
+---------+
```

SOUNDEX は、入力文字列に英語の文字が含まれていない場合、空の文字列を返します。

```
SELECT SOUNDEX('+-*/%');

+---------+
| soundex |
+---------+
|         |
+---------+
```

例

`Amazon` の Soundex 値を返すには、次の例を使用します。

```
SELECT SOUNDEX('Amazon');

+---------+
| soundex |
+---------+
| A525    |
+---------+
```

`smith` および `smyth` の Soundex 値を返すには、次の例を使用します。Soundex の値は同じであることに注意してください。

```
SELECT SOUNDEX('smith'), SOUNDEX('smyth');

+-------+-------+
| smith | smyth |
+-------+-------+
| S530  | S530  |
+-------+-------+
```

SPLIT_PART 関数

指定された区切り記号で文字列を分割し、指定された位置にあるパートを返します。

構文

```
SPLIT_PART(string, delimiter, position)
```

引数

string

分割する文字列の列、式、または文字列リテラル。文字列には CHAR 型または VARCHAR 型を指定できます。

delimiter

入力文字列のセクションを示す区切り文字列。

delimiter がリテラルである場合は、それを一重引用符で囲みます。

position

返す文字列の部分の位置 (1 からカウント)。1 以上の整数である必要があります。位置が文字列の部分の数より大きい場合、SPLIT_PART は空の文字列を返します。文字列で区切り文字が見つからない場合、戻り値には指定された部分の内容が含まれます。これは、文字列全体または空の値である可能性があります。

戻り型

CHAR 文字列または VARCHAR 文字列 (文字列パラメータと同じ型)。

例

次の例では、$ 区切り文字を使用して文字列リテラルを複数の部分に分割し、2 番目の部分を返します。

```
select split_part('abc$def$ghi','$',2)

split_part
----------
def
```

次の例では、$ 区切り文字を使用して文字列リテラルを複数の部分に分割します。4 の部分が見つからないため、空の文字列が返されます。

```
select split_part('abc$def$ghi','$',4)
```

```
split_part
----------
```

次の例では、# 区切り文字を使用して文字列リテラルを複数の部分に分割します。区切り文字が見つからないため、最初の部分である文字列全体が返されます。

```
select split_part('abc$def$ghi','#',1)

split_part
------------
abc$def$ghi
```

次の例は、タイムスタンプフィールド LISTTIME を年コンポーネント、月コンポーネント、および日コンポーネントに分割します。

```
select listtime, split_part(listtime,'-',1) as year,
split_part(listtime,'-',2) as month,
split_part(split_part(listtime,'-',3),' ',1) as day
from listing limit 5;

      listtime       | year | month | day
---------------------+------+-------+------
 2008-03-05 12:25:29 | 2008 | 03    | 05
 2008-09-09 08:03:36 | 2008 | 09    | 09
 2008-09-26 05:43:12 | 2008 | 09    | 26
 2008-10-04 02:00:30 | 2008 | 10    | 04
 2008-01-06 08:33:11 | 2008 | 01    | 06
```

次の例は、LISTTIME タイムスタンプフィールドを選択し、それを '-' 文字で分割して月 (LISTTIME 文字列の 2 番目の部分) を取得してから、各月のエントリ数をカウントします。

```
select split_part(listtime,'-',2) as month, count(*)
from listing
group by split_part(listtime,'-',2)
order by 1, 2;

 month | count
-------+-------
    01 | 18543
    02 | 16620
```

```
03 | 17594
04 | 16822
05 | 17618
06 | 17158
07 | 17626
08 | 17881
09 | 17378
10 | 17756
11 | 12912
12 | 4589
```

STRPOS 関数

指定された文字列内のサブ文字列の位置を返します。

同様の関数については、「CHARINDEX 関数」および「POSITION 関数」を参照してください。

構文

```
STRPOS(string, substring )
```

引数

string

最初の入力パラメータは、検索する CHAR または VARCHAR 文字列です。

substring

2 番目のパラメータは、string 内で検索するサブ文字列です。

戻り型

INTEGER

STRPOS 関数は、substring の位置に対応する INTEGER (0 ではなく 1 から始まる) を返します。位置はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。

使用に関する注意事項

string 内で substring が見つからない場合、STRPOS は 0 を返します。

```
SELECT STRPOS('dogfish', 'fist');

+--------+
| strpos |
+--------+
|      0 |
+--------+
```

例

dogfish 内の fish の位置を示すには、次の例を使用します。

```
SELECT STRPOS('dogfish', 'fish');

+--------+
| strpos |
+--------+
|      4 |
+--------+
```

次の例では、TICKIT サンプルデータベースの SALES テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

SALES テーブル内で COMMISSION が 999.00 を上回る販売取引の数を返すには、次の例を使用します。

```
SELECT DISTINCT STRPOS(commission, '.'),
COUNT (STRPOS(commission, '.'))
FROM sales
WHERE STRPOS(commission, '.') > 4
GROUP BY STRPOS(commission, '.')
ORDER BY 1, 2;

+--------+-------+
| strpos | count |
+--------+-------+
|      5 |   629 |
+--------+-------+
```

STRTOL 関数

基数が指定された数の文字列式を、相当する整数値に変換します。変換後の値は、符号付き 64 ビットの範囲内である必要があります。

構文

```
STRTOL(num_string, base)
```

引数

num_string

変換する数値の文字列式。num_string が空 ('') であるか、null 文字 ('\0') で始まる場合、変換後の値は 0 になります。num_string が NULL 値を含む列である場合、STRTOL は NULL を返します。文字列の冒頭は任意の数の空白があっても構わず、オプションで正符号 '+' または負符号 '-' 1 個を後ろに付加して正または負を示すことができます。デフォルトは '+' です。base が 16 の場合、オプションで文字列の冒頭に '0x' を使用できます。

base

2～36 の INTEGER。

戻り型

BIGINT

num_string が null の場合、この関数は NULL を返します。

例

文字列と基数のペアを整数に変換するには、次の例を使用します。

```
SELECT STRTOL('0xf',16);

+--------+
| strtol |
+--------+
|     15 |
+--------+
```

```
SELECT STRTOL('abcd1234',16);

+------------+
|   strtol   |
+------------+
| 2882343476 |
+------------+

SELECT STRTOL('1234567', 10);

+---------+
| strtol  |
+---------+
| 1234567 |
+---------+

SELECT STRTOL('1234567', 8);

+--------+
| strtol |
+--------+
| 342391 |
+--------+

SELECT STRTOL('110101', 2);

+--------+
| strtol |
+--------+
|     53 |
+--------+

SELECT STRTOL('\0', 2);

+--------+
| strtol |
+--------+
|      0 |
+--------+
```

SUBSTRING 関数

文字列内で、指定された開始位置からの文字列のサブセットを返します。

入力が文字列の場合、抽出される文字の開始位置および文字数はバイト数ではなく文字数に基づきます。つまり、マルチバイト文字は 1 文字としてカウントされます。入力がバイナリ式の場合、開始位置と抽出される部分文字列はバイト数に基づきます。負の長さを指定することはできませんが、開始位置を負に指定することは可能です。

構文

```
SUBSTRING(character_string FROM start_position [ FOR number_characters ] )
```

```
SUBSTRING(character_string, start_position, number_characters )
```

```
SUBSTRING(binary_expression, start_byte, number_bytes )
```

```
SUBSTRING(binary_expression, start_byte )
```

引数

character_string

検索する文字列。文字データ型以外のデータ型は、文字列のように扱われます。

start_position

文字列内で抽出を開始する位置 (1 から始まる)。start_position はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。負の数を指定することもできます。

number_characters

抽出する文字の数 (サブ文字列の長さ)。number_characters はバイト数ではなく文字数に基づくため、マルチバイト文字は 1 文字としてカウントされます。負の数を指定することはできません。

binary_expression

検索する VARBYTE データ型の binary_expression。

start_byte

バイナリ表現内の抽出を開始する (先頭を 1 とする) 位置。負の数を指定することもできます。

number_bytes

抽出するバイト数 (サブ文字列の長さ)。負の数を指定することはできません。

戻り型

入力に応じて、VARCHAR 型または VARBYTE 型を取ります。

使用に関する注意事項

以下は、start_position と number_characters を使用して文字列のさまざまな位置から部分文字列を抽出する方法の例です。

次の例では、6 番目の文字で始まる 4 文字の文字列を返します。

```
select substring('caterpillar',6,4);
substring
-----------
pill
(1 row)
```

start_position + number_characters が文字列の長さを超える場合、SUBSTRING は、start_position から文字列末尾までのサブ文字列を返します。次に例を示します。

```
select substring('caterpillar',6,8);
substring
-----------
pillar
(1 row)
```

`start_position` が負の数または 0 である場合、SUBSTRING 関数は、文字列の先頭文字から `start_position` + `number_characters` -1 文字までをサブ文字列として返します。次に例を示します。

```
select substring('caterpillar',-2,6);
substring
-----------
cat
(1 row)
```

start_position + number_characters -1 が 0 以下である場合、SUBSTRING は空の文字列を返します。次に例を示します。

```
select substring('caterpillar',-5,4);
substring
-----------

(1 row)
```

例

次の例は、LISTING テーブル内の LISTTIME 文字列から月を返します。

```
select listid, listtime,
substring(listtime, 6, 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

次の例は上記と同じですが、FROM...FOR オプションを使用します。

```
select listid, listtime,
substring(listtime from 6 for 2) as month
from listing
order by 1, 2, 3
limit 10;

 listid |      listtime       | month
```

```
--------+---------------------+-------
      1 | 2008-01-24 06:43:29 | 01
      2 | 2008-03-05 12:25:29 | 03
      3 | 2008-11-01 07:35:33 | 11
      4 | 2008-05-24 01:18:37 | 05
      5 | 2008-05-17 02:29:11 | 05
      6 | 2008-08-15 02:08:13 | 08
      7 | 2008-11-15 09:38:15 | 11
      8 | 2008-11-09 05:07:30 | 11
      9 | 2008-09-09 08:03:36 | 09
     10 | 2008-06-17 09:44:54 | 06
(10 rows)
```

文字列にマルチバイト文字が含まれる可能性がある場合、SUBSTRING を使用して文字列の先頭部分を期待どおりに抽出することはできません。これは、マルチバイト文字列の長さを、文字数ではなくバイト数に基づいて指定する必要があるためです。バイト数での長さに基づいて文字列の最初のセグメントを取得するためには、文字列を VARCHAR(byte_length) として CAST することで文字列を切り捨てます。このとき、byte_length は必要な長さとなります。次の例では、文字列 `'Fourscore and seven'` から最初の 5 バイトを抽出します。

```
select cast('Fourscore and seven' as varchar(5));

varchar
-------
Fours
```

次に、バイナリ値 abc の開始位置が負の場合の例を示します。開始位置が -3 であるため、サブ文字列はバイナリ値の先頭から抽出されます。結果は、バイナリ部分文字列の 16 進数表現として自動的に表示されます。

```
select substring('abc'::varbyte, -3);

 substring
-----------
 616263
```

次に、バイナリ値 abc の開始位置が 1 の場合の例を示します。長さが指定されていないため、文字列の抽出は文字列の開始位置から末尾までを対象に行われます。結果は、バイナリ部分文字列の 16 進数表現として自動的に表示されます。

```
select substring('abc'::varbyte, 1);
```

```
 substring
-----------
 616263
```

次に、バイナリ値 abc の開始位置が 3 の場合の例を示します。長さが指定されていないため、文字列の抽出は文字列の開始位置から末尾までを対象に行われます。結果は、バイナリ部分文字列の 16 進数表現として自動的に表示されます。

```
select substring('abc'::varbyte, 3);

 substring
-----------
 63
```

次に、バイナリ値 abc に対して開始位置が 2 の場合の例を示します。文字列は開始位置から位置 10 まで抽出されますが、文字列の末尾は位置 3 になります。結果は、バイナリ部分文字列の 16 進数表現として自動的に表示されます。

```
select substring('abc'::varbyte, 2, 10);

 substring
-----------
 6263
```

次に、バイナリ値 abc に対して開始位置が 2 の場合の例を示します。文字列は開始位置から 1 バイト分抽出されます。結果は、バイナリ部分文字列の 16 進数表現として自動的に表示されます。

```
select substring('abc'::varbyte, 2, 1);

 substring
-----------
 62
```

次の例では、入力文字列 `Ana` の最後のスペースの後に表示される最初の名前 `Silva, Ana` を返します。

```
select reverse(substring(reverse('Silva, Ana'), 1, position(' ' IN reverse('Silva,
 Ana'))))
```

```
 reverse
-----------
 Ana
```

TEXTLEN 関数

LEN 関数のシノニム。

「LEN 関数」を参照してください。

TRANSLATE 関数

任意の式において、指定された文字をすべて、指定された別の文字に置き換えます。既存の文字は、characters_to_replace 引数および characters_to_substitute 引数内の位置により置換文字にマッピングされます。characters_to_replace 引数で characters_to_substitute 引数よりも多くの文字が指定されている場合、characters_to_replace 引数からの余分な文字は戻り値で省略されます。

TRANSLATE は、REPLACE 関数や REGEXP_REPLACE 関数 と似ています。ただし、REPLACE は 1 つの文字列全体を別の文字列に置換し、REGEXP_REPLACE を使用すると正規表現パターンの文字列を検索できますが、TRANSLATE は複数の単一文字置換を行います。

いずれかの引数が null である場合、戻り値は NULL になります。

構文

```
TRANSLATE( expression, characters_to_replace, characters_to_substitute )
```

引数

expression

変換する式。

characters_to_replace

置換する文字を含む文字列。

characters_to_substitute

代入する文字を含む文字列。

戻り型

VARCHAR

例

文字列内の複数の文字を置き換えるには、次の例を使用します。

```
SELECT TRANSLATE('mint tea', 'inea', 'osin');

+-----------+
| translate |
+-----------+
| most tin  |
+-----------+
```

次の例では、TICKIT サンプルデータベースの USERS テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

列内のすべての値のアットマーク (@) をピリオドに置き換えるには、次の例を使用します。

```
SELECT email, TRANSLATE(email, '@', '.') as obfuscated_email
FROM users LIMIT 10;

+---------------------------------------+---------------------------------------+
|                 email                 |           obfuscated_email            |
+---------------------------------------+---------------------------------------+
| Cum@accumsan.com                      | Cum.accumsan.com                      |
| lorem.ipsum@Vestibulumante.com        | lorem.ipsum.Vestibulumante.com        |
| non.justo.Proin@ametconsectetuer.edu  | non.justo.Proin.ametconsectetuer.edu  |
| non.ante.bibendum@porttitortellus.org | non.ante.bibendum.porttitortellus.org |
| eros@blanditatnisi.org                | eros.blanditatnisi.org                |
| augue@Donec.ca                        | augue.Donec.ca                        |
| cursus@pedeacurna.edu                 | cursus.pedeacurna.edu                 |
| at@Duis.com                           | at.Duis.com                           |
| quam@facilisisvitaeorci.ca            | quam.facilisisvitaeorci.ca            |
| mi.lorem@nunc.edu                     | mi.lorem.nunc.edu                     |
+---------------------------------------+---------------------------------------+
```

列内のすべての値のスペースをアンダースコアに置き換え、ピリオドを削除するには、次の例を使用します。

```
SELECT city, TRANSLATE(city, ' .', '_')
```

```
FROM users
WHERE city LIKE 'Sain%' OR city LIKE 'St%'
GROUP BY city
ORDER BY city;

+-----------------+---------------+
|      city       |   translate   |
+-----------------+---------------+
| Saint Albans    | Saint_Albans  |
| Saint Cloud     | Saint_Cloud   |
| Saint Joseph    | Saint_Joseph  |
| Saint Louis     | Saint_Louis   |
| Saint Paul      | Saint_Paul    |
| St. George      | St_George     |
| St. Marys       | St_Marys      |
| St. Petersburg  | St_Petersburg |
| Stafford        | Stafford      |
| Stamford        | Stamford      |
| Stanton         | Stanton       |
| Starkville      | Starkville    |
| Statesboro      | Statesboro    |
| Staunton        | Staunton      |
| Steubenville    | Steubenville  |
| Stevens Point   | Stevens_Point |
| Stillwater      | Stillwater    |
| Stockton        | Stockton      |
| Sturgis         | Sturgis       |
+-----------------+---------------+
```

TRIM 関数

空白または指定した文字で文字列を切り捨てます。

構文

```
TRIM( [ BOTH | LEADING | TRAILING ] [trim_chars FROM ] string )
```

引数

BOTH | LEADING | TRAILING

(オプション) 文字をどこから切り捨てるかを指定します。先頭と末尾の文字を削除するには BOTH を、先頭の文字のみを削除するには LEADING を、末尾の文字のみを削除するには

TRAILING を使用します。このパラメータを省略すると、先頭と末尾の両方の文字が削除されます。

trim_chars

(オプション) 文字列から切り捨てられる文字。このパラメータを省略すると、空白が切り捨てられます。

string

切り捨てる文字列。

戻り型

TRIM 関数は、VARCHAR 型または CHAR 型の文字列を返します。TRIM 関数を SQL コマンドで使用すると、Amazon Redshift が結果を暗黙的に VARCHAR に変換します。SQL 関数の SELECT リストで TRIM 関数を使用した場合は、Amazon Redshift が結果を暗黙的に変換しないため、データ型の不一致によるエラーを回避するために、変換の明示的な実行が必要になることがあります。明示的な変換については、CAST 関数および CONVERT 関数 関数を参照してください。

例

文字列　dog　の先頭および末尾の空白を切り捨てるには、次の例を使用します。

```
SELECT TRIM('    dog    ');

+-------+
| btrim |
+-------+
| dog   |
+-------+
```

文字列　dog　の先頭および末尾の空白を切り捨てるには、次の例を使用します。

```
SELECT TRIM(BOTH FROM '    dog    ');

+-------+
| btrim |
+-------+
| dog   |
+-------+
```

文字列 "dog" から先頭の二重引用符を削除するには、次の例を使用します。

```
SELECT TRIM(LEADING '"' FROM'"dog"');

+-------+
| ltrim |
+-------+
| dog"  |
+-------+
```

文字列 "dog" から末尾の二重引用符を削除するには、次の例を使用します。

```
SELECT TRIM(TRAILING '"' FROM'"dog"');

+-------+
| rtrim |
+-------+
| "dog  |
+-------+
```

TRIM は、trim_chars のいずれかの文字が string の先頭または末尾にある場合、これらの文字をすべて削除します。次の例では、文字 'C'、'D'、および 'G' が VENUENAME の先頭または末尾にある場合 (VARCHAR 列)、これらの文字を切り捨てます。詳細については、「サンプルデータベース」を参照してください。

```
SELECT venueid, venuename, TRIM('CDG' FROM venuename)
FROM venue
WHERE venuename LIKE '%Park'
ORDER BY 2
LIMIT 7;

+---------+----------------------------+---------------------------+
| venueid |         venuename          |           btrim           |
+---------+----------------------------+---------------------------+
|     121 | AT&T Park                  | AT&T Park                 |
|     109 | Citizens Bank Park         | itizens Bank Park         |
|     102 | Comerica Park              | omerica Park              |
|       9 | Dick's Sporting Goods Park | ick's Sporting Goods Park |
|      97 | Fenway Park                | Fenway Park               |
|     112 | Great American Ball Park   | reat American Ball Park   |
|     114 | Miller Park                | Miller Park               |
```

```
+---------+----------------------------+---------------------------+
```

UPPER 関数

文字列を大文字に変換します。UPPER は、UTF-8 マルチバイト文字に対応しています (1 文字につき最大で 4 バイトまで)。

構文

```
UPPER(string)
```

引数

string

入力パラメータは VARCHAR 文字列 (または CHAR など、暗黙的に VARCHAR に変換できるその他のデータ型) です。

戻り型

UPPER 関数は、入力文字列のデータ型と同じデータ型の文字列を返します。例えば、入力が VARCHAR 文字列の場合、この関数は VARCHAR 文字列を返します。

例

次の例では、TICKIT サンプルデータベースの CATEGORY テーブルからのデータを使用します。詳細については、「サンプルデータベース」を参照してください。

CATNAME フィールドを大文字に変換するには、次の例を使用します。

```
SELECT catname, UPPER(catname)
FROM category
ORDER BY 1,2;

+-----------+-----------+
|  catname  |   upper   |
+-----------+-----------+
| Classical | CLASSICAL |
| Jazz      | JAZZ      |
| MLB       | MLB       |
| MLS       | MLS       |
| Musicals  | MUSICALS  |
```

```
| NBA        | NBA        |
| NFL        | NFL        |
| NHL        | NHL        |
| Opera      | OPERA      |
| Plays      | PLAYS      |
| Pop        | POP        |
+-----------+-----------+
```

SUPER 型の情報関数

以下では、SUPER データ型の入力から動的情報を取得するために Amazon Redshift でサポートされる、SQL の型情報関数について説明しています。

トピック

DECIMAL_PRECISION 関数

保存される小数点以下の最大合計桁数の精度をチェックします。この数値には、小数点の左桁と右桁の両方が含まれます。精度の範囲は 1～38 で、デフォルトは 38 です。

構文

```
DECIMAL_PRECISION(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

INTEGER

例

テーブル t に DECIMAL_PRECISION 関数を適用するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_PRECISION(s) FROM t;

+-------------------+
| decimal_precision |
+-------------------+
|                 6 |
+-------------------+
```

DECIMAL_SCALE 関数

小数点の右側に保存される小数点以下の桁数を確認します。スケールの範囲は 0 から精度ポイントまでで、デフォルトは 0 です。

構文

```
DECIMAL_SCALE(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

INTEGER

例

テーブル t に DECIMAL_SCALE 関数を適用するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (3.14159);

SELECT DECIMAL_SCALE(s) FROM t;

+---------------+
| decimal_scale |
+---------------+
|             5 |
+---------------+
```

IS_ARRAY 関数

変数が配列であるかどうかをチェックします。変数が配列の場合、この関数は `true` を返します。この関数には、空の配列も含まれています。それ以外の場合は、この関数は null を含む他のすべての値に対して `false` を返します。

構文

```
IS_ARRAY(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

[1,2] が IS_ARRAY 関数を使用する配列であるかどうか確認するには、次の例を使用します。

```
SELECT IS_ARRAY(JSON_PARSE('[1,2]'));

+----------+
| is_array |
+----------+
| true     |
+----------+
```

IS_BIGINT 関数

値が BIGINT であるかどうか確認します。IS_BIGINT 関数は、64 ビット範囲のスケール 0 の数値に対して true を返します。それ以外の場合、この関数は、null および浮動小数点数を含む他のすべての値に対して false を返します。

IS_BIGINT 関数は、IS_INTEGER のスーパーセットです。

構文

```
IS_BIGINT(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

5 が IS_ARRAY 関数を使用する BIGINT であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_BIGINT(s) FROM t;

+---+-----------+
| s | is_bigint |
+---+-----------+
| 5 | true      |
+---+-----------+
```

IS_BOOLEAN 関数

値が BOOLEAN であるかどうか確認します。IS_BOOLEAN 関数は、定数 JSON ブール値に対して true を返します。この関数は、null を含むその他の値に対して false を返します。

構文

```
IS_BOOLEAN(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

TRUE が IS_BOOLEAN 関数を使用する BOOLEAN であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (TRUE);
```

```
SELECT s, IS_BOOLEAN(s) FROM t;

+------+------------+
|  s   | is_boolean |
+------+------------+
| true | true       |
+------+------------+
```

IS_CHAR 関数

値が CHAR であるかどうか確認します。IS_CHAR 関数は、ASCII 文字のみを含む文字列に対して true を返します。これは、CHAR 型は ASCII 形式の文字のみを保存できるためです。この関数は、他の値に対して false を返します。

構文

```
IS_CHAR(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

t が IS_CHAR 関数を使用する CHAR であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('t');

SELECT s, IS_CHAR(s) FROM t;

+-----+---------+
|  s  | is_char |
+-----+---------+
```

```
| "t" | true    |
+-----+---------+
```

IS_DECIMAL 関数

値が DECIMAL であるかどうか確認します。IS_DECIMAL 関数は、浮動小数点ではない数値に対して true を返します。この関数は、null を含むその他の値に対して false を返します。

IS_DECIMAL 関数は、IS_BIGINT のスーパーセットです。

構文

```
IS_DECIMAL(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

1.22 が IS_DECIMAL 関数を使用する DECIMAL であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (1.22);

SELECT s, IS_DECIMAL(s) FROM t;

+------+------------+
|  s   | is_decimal |
+------+------------+
| 1.22 | true       |
+------+------------+
```

IS_FLOAT 関数

値が浮動小数点数であるかどうかをチェックします。IS_FLOAT 関数は、浮動小数点数 (FLOAT4 および FLOAT8) に対して true を返します。この関数は、他の値に対して false を返します。

IS_DECIMAL セットと IS_FLOAT セットは互いに素です。

構文

```
IS_FLOAT(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

2.22::FLOAT が IS_FLOAT 関数を使用する FLOAT であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES(2.22::FLOAT);

SELECT s, IS_FLOAT(s) FROM t;

+---------+----------+
|    s    | is_float |
+---------+----------+
| 2.22e+0 | true     |
+---------+----------+
```

IS_INTEGER 関数

32 ビット範囲のスケール 0 の数値については true を返し、それ以外の値 (null と浮動小数点数を含む) については false を返します。

IS_INTEGER 関数は、IS_SMALLINT 関数のスーパーセットです。

構文

```
IS_INTEGER(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

5 が IS_INTEGER 関数を使用する INTEGER であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_INTEGER(s) FROM t;

+---+------------+
| s | is_integer |
+---+------------+
| 5 | true       |
+---+------------+
```

IS_OBJECT 関数

変数がオブジェクトであるかどうかをチェックします。IS_OBJECT 関数は、空のオブジェクトを含むオブジェクトに対して true を返します。この関数は、null を含むその他の値に対して false を返します。

構文

```
IS_OBJECT(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

{"name": "Joe"} が IS_OBJECT 関数を使用するオブジェクトであるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s super);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_OBJECT(s) FROM t;

+----------------+-----------+
|       s        | is_object |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SCALAR 関数

変数がスカラーであるかどうかをチェックします。IS_SCALAR 関数は、配列またはオブジェクトではない任意の値に対して `true` を返します。この関数は、null を含むその他の値に対して `false` を返します。

IS_ARRAY、IS_OBJECT、および IS_SCALAR のセットは、null 以外のすべての値をカバーします。

構文

```
IS_SCALAR(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

{"name": "Joe"} が IS_SCALAR 関数を使用するスカラーであるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (JSON_PARSE('{"name": "Joe"}'));

SELECT s, IS_SCALAR(s.name) FROM t;

+----------------+-----------+
|       s        | is_scalar |
+----------------+-----------+
| {"name":"Joe"} | true      |
+----------------+-----------+
```

IS_SMALLINT 関数

変数が SMALLINT であるかどうか確認します。IS_SMALLINT 関数は、16 ビット範囲のスケール 0 の数値に対して true を返します。この関数は、null および浮動小数点数を含む他のすべての値に対して false を返します。

構文

```
IS_SMALLINT(super_expression)
```

引数

super_expression

SUPER 式または列。

戻る

BOOLEAN

例

5 が IS_SMALLINT 関数を使用する SMALLINT であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES (5);

SELECT s, IS_SMALLINT(s) FROM t;

+---+-------------+
| s | is_smallint |
+---+-------------+
| 5 | true        |
+---+-------------+
```

IS_VARCHAR 関数

変数が VARCHAR であるかどうか確認します。IS_VARCHAR 関数は、すべての文字列に対して true を返します。この関数は、他の値に対して false を返します。

IS_VARCHAR 関数は、IS_CHAR 関数のスーパーセットです。

構文

```
IS_VARCHAR(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

BOOLEAN

例

abc が IS_VARCHAR 関数を使用する VARCHAR であるかどうか確認するには、次の例を使用します。

```
CREATE TABLE t(s SUPER);

INSERT INTO t VALUES ('abc');

SELECT s, IS_VARCHAR(s) FROM t;

+-------+------------+
|   s   | is_varchar |
+-------+------------+
| "abc" | true       |
+-------+------------+
```

JSON_SIZE 関数

JSON_SIZE 関数は、文字列にシリアル化されたときに、指定した SUPER 式のバイト数を返します。

構文

```
JSON_SIZE(super_expression)
```

引数

super_expression

SUPER 定数または式。

戻り型

INTEGER

JSON_SIZE 関数は、入力文字列のバイト数を示す INTEGER を返します。この値は文字数とは異なります。例えば、UTF-8 文字「#」(黒いドット) のサイズは、1 文字ですが 3 バイトです。

使用に関する注意事項

JSON_SIZE (x) は、機能的には OCTET_LENGTH (JSON_SERIALIZE) と同じです。ただし、指定された SUPER 式がシリアル化時にシステムの VARCHAR 制限を超える場合、JSON_SERIALIZE はエラーを返すことに注意してください。JSON_SIZE にはこの制限はありません。

例

文字列にシリアル化された SUPER 値の長さを返すには、次の例を使用します。

```
SELECT JSON_SIZE(JSON_PARSE('[10001,10002,"#"]'));

+-----------+
| json_size |
+-----------+
|        19 |
+-----------+
```

指定された SUPER 式の長さは 17 文字ですが、「#」は 3 バイトの文字なので、JSON_SIZE は 19 を返すことに注意してください。

JSON_TYPEOF 関数

JSON_TYPEOF スカラー関数は、SUPER 値の動的型に応じて、ブール値、数値、文字列、オブジェクト、配列、または null の値を持つ VARCHAR を返します。

構文

```
JSON_TYPEOF(super_expression)
```

引数

super_expression

SUPER 式または列。

戻り型

VARCHAR

例

JSON_TYPEOF 関数を使用して、配列 [1,2] の JSON の型を確認するには、次の例を使用します。

```
SELECT JSON_TYPEOF(ARRAY(1,2));

+-------------+
| json_typeof |
+-------------+
| array       |
+-------------+
```

JSON_TYPEOF 関数を使用して、オブジェクト {"name":"Joe"} の JSON 型を確認するには、次の例を使用します。

```
SELECT JSON_TYPEOF(JSON_PARSE('{"name":"Joe"}'));

+-------------+
| json_typeof |
+-------------+
| object      |
+-------------+
```

SIZE

SUPER 型の定数または式のバイナリのメモリ内サイズを INTEGER で返します。

構文

```
SIZE(super_expression)
```

引数

super_expression

SUPER 型の定数または式。

戻り型

INTEGER

例

SIZE を使用して複数の SUPER 型式のメモリ内サイズを取得するには、次の例を使用します。

```
CREATE TABLE test_super_size(a SUPER);

INSERT INTO test_super_size
VALUES
  (null),
  (TRUE),
  (JSON_PARSE('[0,1,2,3]')),
  (JSON_PARSE('{"a":0,"b":1,"c":2,"d":3}'))
;

SELECT a, SIZE(a)
FROM test_super_size
ORDER BY 2, 1;

+---------------------------+------+
|             a             | size |
+---------------------------+------+
| true                      |    4 |
| NULL                      |    4 |
| [0,1,2,3]                 |   23 |
| {"a":0,"b":1,"c":2,"d":3} |   52 |
+---------------------------+------+
```

VARBYTE 関数と演算子

Amazon Redshift で VARBYTE データ型をサポートする関数と演算子は以下のとおりです。

- VARBYTE 演算子
- FROM_HEX
- FROM_VARBYTE
- GETBIT
- TO_HEX
- TO_VARBYTE
- CONCAT
- LEN

- LENGTH 関数
- OCTET_LENGTH
- SUBSTRING 関数

VARBYTE 演算子

以下は VARBYTE 演算子の一覧です。この演算子は、VARBYTE データ型のバイナリ値に対し機能します。2 つの入力の内いずれかが null であると、結果は null になります。

サポートされている演算子

演算子	説明	戻り型
<	Less than	BOOLEAN
<=	Less than or equal	BOOLEAN
=	Equal	BOOLEAN
>	Greater than	BOOLEAN
>=	Greater than or equal	BOOLEAN
!= または <>	等しくない	BOOLEAN
||	連結	VARBYTE
+	連結	VARBYTE
~	ビット単位 NOT	VARBYTE
&	ビット単位 AND	VARBYTE

演算子	説明	戻り型
		ビット単位 OR	VARBYTE
#	ビット単位 XOR	VARBYTE

例

次の例では、'a'::VARBYTE の値は 61 であり 'b'::VARBYTE の値は 62 です。:: は、文字列を VARBYTE データ型にキャストします。キャストデータ型の詳細については、「CAST」を参照してください。

< 演算子を使用して、'a' が 'b' より小さいかどうかを比較するには、次の例を使用します。

```
SELECT 'a'::VARBYTE < 'b'::VARBYTE AS less_than;

+-----------+
| less_than |
+-----------+
| true      |
+-----------+
```

= 演算子を使用して、'a' が 'b' と等しいかどうかを比較するには、次の例を使用します。

```
SELECT 'a'::VARBYTE = 'b'::VARBYTE AS equal;

+-------+
| equal |
+-------+
| false |
+-------+
```

|| 演算子を使用して 2 つのバイナリ値を連結するには、次の例を使用します。

```
SELECT 'a'::VARBYTE || 'b'::VARBYTE AS concat;

+--------+
| concat |
```

```
+--------+
|   6162 |
+--------+
```

+ 演算子を使用して 2 つのバイナリ値を連結するには、次の例を使用します。

```
SELECT 'a'::VARBYTE + 'b'::VARBYTE AS concat;

+--------+
| concat |
+--------+
|   6162 |
+--------+
```

FROM_VARBYTE 関数を使用して入力バイナリ値の各ビットを否定するには、次の例を使用します。文字列 'a' は 01100001 と評価されます。詳細については、「FROM_VARBYTE」を参照してください。

```
SELECT FROM_VARBYTE(~'a'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
|     10011110 |
+--------------+
```

2 つの入力バイナリ値に対して & 演算子を適用するには、次の例を使用します。文字列 'a' は 01100001 と評価され、'b' は 01100010 と評価されます。

```
SELECT FROM_VARBYTE('a'::VARBYTE & 'b'::VARBYTE, 'binary');

+--------------+
| from_varbyte |
+--------------+
|     01100000 |
+--------------+
```

FROM_HEX 関数

FROM_HEX は 16 進数をバイナリ数に変換します。

構文

```
FROM_HEX(hex_string)
```

引数

hex_string

変換する 16 進数文字列 (VARCHAR または TEXT) です。形式はリテラル値である必要があります。

戻り型

VARBYTE

例

'6162' の 16 進数表現をバイナリ値に変換するには、次の例を使用します。結果は、バイナリ値の 16 進数表現として自動的に出力されます。

```
SELECT FROM_HEX('6162');

+----------+
| from_hex |
+----------+
|     6162 |
+----------+
```

FROM_VARBYTE 関数

FROM_VARBYTE は、バイナリ値を指定した形式の文字列に変換します。

構文

```
FROM_VARBYTE(binary_value, format)
```

引数

binary_value

VARBYTE データ型のバイナリ値です。

format

返される文字列のフォーマット。大文字と小文字を区別しない有効な値は hex、binary、utf8 (utf-8 と utf_8 も) および base64 です。

戻り型

VARCHAR

例

バイナリ値 'ab' を 16 進数値に変換するには、次の例を使用します。

```
SELECT FROM_VARBYTE('ab', 'hex');

+--------------+
| from_varbyte |
+--------------+
|         6162 |
+--------------+
```

'4d' のバイナリ表現を返すには、次の例を使用します。'4d' のバイナリ表現は文字列 01001101 です。

```
SELECT FROM_VARBYTE(FROM_HEX('4d'), 'binary');

+--------------+
| from_varbyte |
+--------------+
|     01001101 |
+--------------+
```

GETBIT 関数

GETBIT は、指定されたインデックスのバイナリ値のビット値を返します。

構文

```
GETBIT(binary_value, index)
```

引数

binary_value

VARBYTE データ型のバイナリ値です。

index

ビット値として返されるバイナリ値を指定するインデックスの番号です。バイナリ値はビットの配列であり、右端のビット (最下位ビット) から左端のビット (最上位ビット) に向かい、0 から始まるインデックス番号が付けられています。

戻り型

INTEGER

例

バイナリ値 from_hex('4d') のインデックス 2 にあるビットを返すには、次の例を使用します。'4d' のバイナリ表現は 01001101 です。

```
SELECT GETBIT(FROM_HEX('4d'), 2);

+--------+
| getbit |
+--------+
|      1 |
+--------+
```

from_hex('4d') が返すバイナリ値で 8 つのインデックス位置にあるビットを返すには、次の例を使用します。'4d' のバイナリ表現は 01001101 です。

```
SELECT GETBIT(FROM_HEX('4d'), 7), GETBIT(FROM_HEX('4d'), 6),
  GETBIT(FROM_HEX('4d'), 5), GETBIT(FROM_HEX('4d'), 4),
  GETBIT(FROM_HEX('4d'), 3), GETBIT(FROM_HEX('4d'), 2),
  GETBIT(FROM_HEX('4d'), 1), GETBIT(FROM_HEX('4d'), 0);

+--------+--------+--------+--------+--------+--------+--------+--------+
| getbit | getbit | getbit | getbit | getbit | getbit | getbit | getbit |
+--------+--------+--------+--------+--------+--------+--------+--------+
|      0 |      1 |      0 |      0 |      1 |      1 |      0 |      1 |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

TO_HEX 関数

TO_HEX は、数値またはバイナリ値を 16 進数表現に変換します。

構文

```
TO_HEX(value)
```

引数

value

変換する数値またはバイナリ値 (VARBYTE) のいずれかです。

戻り型

VARCHAR

例

数値を 16 進数表現に変換するには、次の例を使用します。

```
SELECT TO_HEX(2147676847);

+----------+
|  to_hex  |
+----------+
| 8002f2af |
+----------+
```

'abc' の VARBYTE 表現を 16 進数値に変換するには、次の例を使用します。

```
SELECT TO_HEX('abc'::VARBYTE);

+--------+
| to_hex |
+--------+
| 616263 |
+--------+
```

テーブルを作成し、'abc' の VARBYTE 表現を 16 進数として挿入して、その値を保持している列を選択するには、次の例を使用します。

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT TO_HEX('abc'::VARBYTE);
SELECT vc FROM t;

+--------+
|   vc   |
+--------+
| 616263 |
+--------+
```

VARBYTE 値を VARCHAR 値にキャストする際の形式が UTF-8 であることを示すには、次の例を使用します。

```
CREATE TABLE t (vc VARCHAR);
INSERT INTO t SELECT 'abc'::VARBYTE::VARCHAR;

SELECT vc FROM t;

+-----+
| vc  |
+-----+
| abc |
+-----+
```

TO_VARBYTE 関数

TO_VARBYTE は、文字列の形式を指定して、その文字列をバイナリ値に変換します。

構文

```
TO_VARBYTE(string, format)
```

引数

string

CHAR または VARCHAR 文字列。

format

入力文字列の形式。大文字と小文字を区別しない有効な値は `hex`、`binary`、`utf8` (`utf-8` と `utf_8` も) および `base64` です。

戻り型

VARBYTE

例

16 進数 6162 をバイナリ値に変換するには、次の例を使用します。結果は、バイナリ値の 16 進数表現として自動的に出力されます。

```
SELECT TO_VARBYTE('6162', 'hex');

+------------+
| to_varbyte |
+------------+
|       6162 |
+------------+
```

4d のバイナリ表現を返すには、次の例を使用します。'4d'のバイナリ表現は `01001101` です。

```
SELECT TO_VARBYTE('01001101', 'binary');

+------------+
| to_varbyte |
+------------+
|         4d |
+------------+
```

UTF-8 の文字列 `'a'` をバイナリ値に変換するには、次の例を使用します。結果は、バイナリ値の 16 進数表現として自動的に出力されます。

```
SELECT TO_VARBYTE('a', 'utf8');

+------------+
| to_varbyte |
+------------+
|         61 |
+------------+
```

16 進数値の文字列 `'4'` をバイナリ値に変換するには、次の例を使用します。16 進数の文字列の長さが奇数の場合、`0` が先頭に追加され有効な 16 進数が形成されます。

```
SELECT TO_VARBYTE('4', 'hex');
```

```
+-------------+
| to_varbyte |
+-------------+
|          04 |
+-------------+
```

Window 関数

ウィンドウ関数を使用すると、分析的なビジネスクエリをより効率的に作成できます。ウィンドウ関数はパーティションまたは結果セットの「ウィンドウ」で演算し、ウィンドウのすべての行に値を返します。それに対して、ウィンドウ以外の関数は、結果セットの行ごとに計算を実行します。結果の行を集計するグループ関数とは異なり、ウィンドウ関数はテーブル式のすべての行を保持します。

戻り値はこのウィンドウの行セットの値を使用して計算されます。ウィンドウはテーブルの各行に、追加の属性を計算するために使用する行のセットを定義します。ウィンドウはウィンドウ仕様 (OVER 句) を使用して定義され、次の 3 つの主要な概念に基づいています。

- ウィンドウのパーティション、列のグループを形成 (PARTITION 句)
- ウィンドウの並び順、各パーティション内の行の順序またはシーケンスの定義 (ORDER BY 句)
- ウィンドウのフレーム、各行に関連して定義され、行のセットをさらに制限 (ROWS 仕様)

ウィンドウ関数は、最後の ORDER BY 句を除いて、クエリで実行される最後の演算のセットです。すべての結合およびすべての WHERE、GROUP BY、および HAVING 句は、ウィンドウ関数が処理される前に完了されます。そのため、ウィンドウ関数は選択リストまたは ORDER BY 句のみに表示できます。複数のウィンドウ関数は、別のフレーム句を持つ 1 つのクエリ内で使用できます。ウィンドウ関数は、CASE などの他のスカラー式でも使用できます。

ウィンドウ関数はネストできません。例えば、集計関数 SUM はウィンドウ関数 SUM 内に表示される可能性がありますが、ウィンドウ関数 SUM は別のウィンドウ関数 SUM 内に表示できません。ウィンドウ関数が別のウィンドウ関数にネストされているため、以下はサポートされません。

```
SELECT SUM(SUM(selectcol) OVER (PARTITION BY ordercol)) OVER (Partition by ordercol)
 FROM t;
```

ウィンドウ関数の構文の概要

ウィンドウ関数は、次のような標準構文に従います。

```
function (expression) OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list [ frame_clause ] ] )
```

ここで、function は、このセクションで説明している関数の 1 つです。

expr_list は次のとおりです。

```
expression | column_name [, expr_list ]
```

order_list は次のとおりです。

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

frame_clause は次のとおりです。

```
ROWS
{ UNBOUNDED PRECEDING | unsigned_value PRECEDING | CURRENT ROW } |

{ BETWEEN
{ UNBOUNDED PRECEDING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | unsigned_value { PRECEDING | FOLLOWING } | CURRENT ROW }}
```

引数

function

ウィンドウ関数。詳細については、個々の関数の説明を参照してください。

OVER

ウィンドウの仕様を定義する句。OVER 句はウィンドウ関数に必須であり、ウィンドウ関数を他の SQL 関数と区別します。

PARTITION BY expr_list

(オプション) PARTITION BY 句は結果セットをパーティションに再分割します。これは GROUP BY 句と似ています。パーティション句が存在する場合、関数は各パーティションの行に対して

計算されます。パーティション句が指定されていない場合、1 つのパーティションにテーブル全体が含まれ、関数は完全なテーブルに対して計算されます。

ランク付け関数 DENSE_RANK、NTILE、RANK、および ROW_NUMBER では、結果セットのすべての行でグローバルな比較が必要です。PARTITION BY clauseを使用すると、クエリオプティマイザーは、パーティションに応じて複数のスライスにワークロードを分散させることにより、個々の集計を並列で実行できます。PARTITION BY 句がない場合、集計ステップを 1 つのスライスで順次実行する必要があり、特に大規模なクラスターではパフォーマンスに大きな悪影響を与えることがあります。

Amazon Redshift は、PARTITION BY 句で文字列リテラルをサポートしていません。

ORDER BY order_list

(オプション) ウィンドウ関数は、ORDER BY で順序仕様に従ってソートされた各パーティション内の行に適用されます。この ORDER BY 句は、frame_clause の ORDER BY 句とは異なり、両者はまったく無関係です。ORDER BY 句は、PARTITION BY 句なしで使用できます。

ランク付け関数の場合、ORDER BY 句はランク付けの値に使用する基準を特定します。集計関数の場合、パーティションで分割された行は、集計関数がフレームごとに計算される前に順序付けされる必要があります。ウィンドウ関数の種類の詳細については、「Window 関数」を参照してください。

列識別子または列識別を検証する式は、順序リストで必要とされます。定数も定数式も、列名の代用として使用することはできません。

NULL 値は独自のグループとして扱われ、NULLS FIRST または NULLS LAST オプションに従ってソートおよびランク付けされます。デフォルトでは、NULL 値は昇順ではソートされて最後にランク付けされ、降順ではソートされて最初にランク付けされます。

Amazon Redshift は、ORDER BY 句で文字列リテラルをサポートしていません。

ORDER BY 句を省略した場合、行の順序は不確定になります。

> **Note**
>
> Amazon Redshift などの並列システムでは、ORDER BY 句がデータの一意および全体の並び順を生成しない場合、行の順序は不確定になります。つまり、ORDER BY 式が重複した値を生成する場合 (部分的ソート)、これらの行の戻り値の順序は Amazon Redshift の実行によって異なることがあります。そのため、ウィンドウ関数は予期しない結果また

は矛盾した結果を返す場合があります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

column_name

パーティション化または順序付けされる列の名前。

ASC | DESC

次のように、式のソート順を定義するオプション:

- ASC: 昇順 (数値の場合は低から高、文字列の場合は「A」から「Z」など) オプションを指定しない場合、データはデフォルトでは昇順にソートされます。
- DESC: 降順 (数値の場合は高から低、文字列の場合は「Z」から「A」)。

NULLS FIRST | NULLS LAST

NULL を NULL 以外の値より先に順序付けするか、NULL 以外の値の後に順序付けするかを指定するオプション。デフォルトでは、NULL は昇順ではソートされて最後にランク付けされ、降順ではソートされて最初にランク付けされます。

frame_clause

集計関数では、ORDER BY を使用する場合、フレーム句は関数のウィンドウで行のセットをさらに絞り込みます。これは、順序付けされた結果内の行のセットを含めるか、または除外できるようにします。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。

frame 句は、ランク付け関数には適用されません。また、集計関数の OVER 句の中に ORDER BY 句がない場合は、フレーム句は必要ありません。ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。

ORDER BY 句が指定されていない場合、暗黙的なフレームはバインドされません (ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING と同じ)。

ROWS

この句は、現在の行からの物理オフセットを指定してウィンドウフレームを定義します。

この句は、現在のウィンドウの行、または現在の行の値を組み合わせるパーティションを指定します。また、現在の行の前後に配置される行の位置を指定する引数を使用します。すべてのウィンドウフレームの参照点は現在の行です。ウィンドウフレームがパーティションで次にスライドすると、各行は順に現在の行になります。

フレームは、次のように現在の行までと現在の行を含む行の簡易セットである場合があります。

```
{UNBOUNDED PRECEDING | offset PRECEDING | CURRENT ROW}
```

また、次の 2 つの境界の間の行のセットである場合があります。

```
BETWEEN
{ UNBOUNDED PRECEDING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
AND
{ UNBOUNDED FOLLOWING | offset { PRECEDING | FOLLOWING } | CURRENT ROW }
```

UNBOUNDED PRECEDING はウィンドウがパーティションの最初の行で開始することを示し、offset PRECEDING はウィンドウが現在の行の前のオフセット値と等しい行数で開始することを示します。デフォルトは UNBOUNDED PRECEDING です。

CURRENT ROW は、ウィンドウが現在の行で開始または終了することを示します。

UNBOUNDED FOLLOWING はウィンドウがパーティションの最後の行で終了することを示し、offset FOLLOWING はウィンドウが現在の行の後のオフセット値と等しい行数で終了することを示します。

offset は、現在の行の前後にある物理的な行数を識別します。この場合、offset は、正の数値に評価される定数である必要があります。例えば、5 FOLLOWING は現在の行より 5 行後のフレームを終了します。

BETWEEN が指定されていない場合、フレームは現在の行により暗黙的に区切られます。例えば、`ROWS 5 PRECEDING` は `ROWS BETWEEN 5 PRECEDING AND CURRENT ROW` と同じです。また、`ROWS UNBOUNDED FOLLOWING`は `ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING` と同じです。

> **Note**
>
> 開始境界が終了境界よりも大きいフレームを指定することはできません。例えば、以下のフレームはいずれも指定することができません。
>
> ```
> between 5 following and 5 preceding
> between current row and 2 preceding
> between 3 following and current row
> ```

ウィンドウ関数用データの一意の並び順

ウィンドウ関数の ORDER BY 句がデータの一意および全体の並び順を生成しない場合、行の順序は不確定になります。ORDER BY 式が重複した値 (部分的な順序付け) を生成する場合、これらの行の戻り値の順序は実行時によって異なる可能性があります。この場合、ウィンドウ関数は予期しない結果または矛盾した結果を返す場合があります。

例えば、次のクエリは、複数の実行にわたって異なる結果を返します。これらの異なる結果は、`order by dateid` が SUM Window 関数でデータの一意の順序を生成しないために発生します。

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |   1730.00 |     1730.00
1827 |    708.00 |     2438.00
1827 |    234.00 |     2672.00
...

select dateid, pricepaid,
sum(pricepaid) over(order by dateid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;

dateid | pricepaid |   sumpaid
--------+-----------+-------------
1827 |    234.00 |      234.00
1827 |    472.00 |      706.00
1827 |    347.00 |     1053.00
...
```

この場合、2 番目の ORDER BY 列をウィンドウ関数に追加すると問題を解決できます。

```
select dateid, pricepaid,
sum(pricepaid) over(order by dateid, pricepaid rows unbounded preceding) as sumpaid
from sales
group by dateid, pricepaid;
```

```
dateid | pricepaid | sumpaid
--------+-----------+---------
1827 |    234.00 |  234.00
1827 |    337.00 |  571.00
1827 |    347.00 |  918.00
...
```

サポートされている関数

Amazon Redshift は、集計とランク付けという 2 つのタイプのウィンドウ関数をサポートします。

サポートされる集約関数は次のとおりです。

- AVG ウィンドウ関数
- COUNT ウィンドウ関数
- CUME_DIST ウィンドウ関数
- DENSE_RANK ウィンドウ関数
- FIRST_VALUE ウィンドウ関数
- LAG ウィンドウ関数
- LAST_VALUE ウィンドウ関数
- LEAD ウィンドウ関数
- LISTAGG ウィンドウ関数
- MAX ウィンドウ関数
- MEDIAN ウィンドウ関数
- MIN ウィンドウ関数
- NTH_VALUE ウィンドウ関数
- PERCENTILE_CONT ウィンドウ関数
- PERCENTILE_DISC ウィンドウ関数
- RATIO_TO_REPORT ウィンドウ関数
- STDDEV_SAMP および STDDEV_POP ウィンドウ関数 (STDDEV_SAMP と STDDEV はシノニムです)
- SUM ウィンドウ関数
- VAR_SAMP および VAR_POP ウィンドウ関数 (VAR_SAMP と VARIANCE はシノニムです)

サポートされる集計関数は次のとおりです。

- DENSE_RANK ウィンドウ関数
- NTILE ウィンドウ関数
- PERCENT_RANK ウィンドウ関数
- RANK ウィンドウ関数
- ROW_NUMBER ウィンドウ関数

ウィンドウ関数例のサンプルテーブル

ウィンドウ関数別の例をそれぞれの説明と共に参照できます。一部の例で使用している WINSALES という名前のテーブルには、以下の 11 行が含まれています。

SALESID	DATEID	SELLERID	BUYERID	QTY	QTY_SHIPPED
30001	8/2/2003	3	B	10	10
10001	12/24/2003	1	C	10	10
10005	12/24/2003	1	A	30	
40001	1/9/2004	4	A	40	
10006	1/18/2004	1	C	10	
20001	2/12/2004	2	B	20	20
40005	2/12/2004	4	A	10	10
20002	2/16/2004	2	C	20	20
30003	4/18/2004	3	B	15	
30004	4/18/2004	3	B	20	
30007	9/7/2004	3	C	30	

次のスクリプトは、WINSALES テーブルの例を作成し、値を入力します。

```
CREATE TABLE winsales(
  salesid int,
  dateid date,
  sellerid int,
  buyerid char(10),
  qty int,
  qty_shipped int);

INSERT INTO winsales VALUES
  (30001, '8/2/2003', 3, 'b', 10, 10),
  (10001, '12/24/2003', 1, 'c', 10, 10),
  (10005, '12/24/2003', 1, 'a', 30, null),
  (40001, '1/9/2004', 4, 'a', 40, null),
  (10006, '1/18/2004', 1, 'c', 10, null),
  (20001, '2/12/2004', 2, 'b', 20, 20),
  (40005, '2/12/2004', 4, 'a', 10, 10),
  (20002, '2/16/2004', 2, 'c', 20, 20),
  (30003, '4/18/2004', 3, 'b', 15, null),
  (30004, '4/18/2004', 3, 'b', 20, null),
  (30007, '9/7/2004', 3, 'c', 30, null);
```

AVG ウィンドウ関数

AVG ウィンドウ関数は入力式の値の平均 (算術平均) を返します。AVG 関数は数値に対してはたらき、NULL 値は無視します。

構文

```
AVG ( [ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数はカウントに使用する式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で AVG 関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

AVG 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、および DOUBLE PRECISION です。

AVG 関数でサポートされる戻り値の型は次のとおりです。

- SMALLINT または INTEGER 引数の場合は BIGINT
- BIGINT 引数の場合は NUMERIC
- 浮動小数点の引数の場合は DOUBLE PRECISION

例

次の例では、日付で販売数のローリング平均を計算します。日付 ID および販売 ID によって結果は順序付けられます。

```
select salesid, dateid, sellerid, qty,
avg(qty) over
(order by dateid, salesid rows unbounded preceding) as avg
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | avg
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  16
40001 | 2004-01-09 |        4 |  40 |  22
10006 | 2004-01-18 |        1 |  10 |  20
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  18
20002 | 2004-02-16 |        2 |  20 |  18
30003 | 2004-04-18 |        3 |  15 |  18
30004 | 2004-04-18 |        3 |  20 |  18
30007 | 2004-09-07 |        3 |  30 |  19
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

COUNT ウィンドウ関数

COUNT ウィンドウ関数は式で定義された行をカウントします。

COUNT 関数には 2 つのバリエーションがあります。COUNT(*) は null を含むかどうかにかかわらず、ターゲットテーブルのすべての行をカウントします。COUNT(expression) は、特定の列または式にある NULL 以外の値を持つ行数を計算します。

構文

```
COUNT ( * | [ ALL ] expression) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数はカウントに使用する式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で COUNT 関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

COUNT 関数は引数のデータ型をすべてサポートします。

COUNT 関数でサポートされる戻り値の型は BIGINT です。

例

次の例では、データウィンドウの先頭から販売 ID、数量、すべての行のカウントを示します。

```
select salesid, qty,
count(*) over (order by salesid rows unbounded preceding) as count
from winsales
```

```
order by salesid;

salesid | qty | count
---------+-----+-----
10001 |  10 |   1
10005 |  30 |   2
10006 |  10 |   3
20001 |  20 |   4
20002 |  20 |   5
30001 |  10 |   6
30003 |  15 |   7
30004 |  20 |   8
30007 |  30 |   9
40001 |  40 |   10
40005 |  10 |   11
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、データウィンドウの先頭から販売 ID、数量、非 Null 行をカウントする方法を示します。(WINSALES テーブルの QTY_SHIPPED 列には NULL が含まれます)

```
select salesid, qty, qty_shipped,
count(qty_shipped)
over (order by salesid rows unbounded preceding) as count
from winsales
order by salesid;

salesid | qty | qty_shipped | count
---------+-----+-------------+-------
10001 |  10 |          10 |   1
10005 |  30 |             |   1
10006 |  10 |             |   1
20001 |  20 |          20 |   2
20002 |  20 |          20 |   3
30001 |  10 |          10 |   4
30003 |  15 |             |   4
30004 |  20 |             |   4
30007 |  30 |             |   4
40001 |  40 |             |   4
40005 |  10 |          10 |   5
(11 rows)
```

CUME_DIST ウィンドウ関数

ウィンドウまたはパーティション内の値の累積分布を計算します。昇順の場合、累積分布は以下の式を使用して特定されます。

`count of rows with values <= x / count of rows in the window or partition`

ここで、x は ORDER BY 句で指定された列の現在の行の値と等しくなります。以下のデータセットは、この式の使用方法を示しています。

```
Row# Value   Calculation    CUME_DIST
1        2500    (1)/(5)    0.2
2        2600    (2)/(5)    0.4
3        2800    (3)/(5)    0.6
4        2900    (4)/(5)    0.8
5        3100    (5)/(5)    1.0
```

戻り値の範囲は、0～1 (0 は含みませんが 1 は含みます) です。

構文

```
CUME_DIST ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

引数

OVER

ウィンドウのパーティションを指定する句。OVER 句にウィンドウフレーム仕様を含めることはできません。

PARTITION BY partition_expression

省略可能。OVER 句の各グループのレコードの範囲を設定する式。

ORDER BY order_list

累積分布を計算する式。式は、数値データ型を含んでいるか、そのデータ型に暗黙的に変換できる必要があります。ORDER BY を省略した場合、すべての行について戻り値は 1 です。

ORDER BY で一意の並べ替えが行われない場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

FLOAT8

例

次の例は、各販売者の数量の累積配布を計算します。

```
select sellerid, qty, cume_dist()
over (partition by sellerid order by qty)
from winsales;

sellerid   qty     cume_dist
--------------------------------------------------
1         10.00    0.33
1         10.64    0.67
1         30.37    1
3         10.04    0.25
3         15.15    0.5
3         20.75    0.75
3         30.55    1
2         20.09    0.5
2         20.12    1
4         10.12    0.5
4         40.23    1
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

DENSE_RANK ウィンドウ関数

DENSE_RANK ウィンドウ関数は、OVER 句の ORDER BY 式に基づいて、値のグループの値のランクを特定します。オプションの PARTITION BY 句がある場合、ランク付けは行のグループごとにリセットされます。ランク付け条件が同じ値の行は、同じランクを受け取ります。DENSE_RANK 関数は、2 行以上で同点となった場合、ランク付けされた値の順位に差はないことが、RANK とは異なります。例えば、2 行が 1 位にランク付けされると、次のランクは 2 位になります。

同じクエリに PARTITION BY および ORDER BY 句のあるランク付け関数を使用することができます。

構文

```
DENSE_RANK() OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

引数

( )

この関数は引数を受け取りませんが、空のかっこは必要です。

OVER

DENSE_RANK 関数のウィンドウ句。

PARTITION BY expr_list

(オプション) ウィンドウを定義する 1 つ以上の式。

ORDER BY order_list

(オプション) ランク付けの値が基とする式。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。ORDER BY を省略した場合、すべての行について戻り値は 1 です。

ORDER BY で一意の並べ替えが行われない場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

INTEGER

例

次の例では、ウィンドウ関数のサンプルテーブルを使用しています。詳細については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、販売数量によってテーブルを順序付けして、各行にデンス値のランクと標準のランクの両方を割り当てます。結果はウィンドウ関数の結果が提供された後にソートされます。

```
SELECT salesid, qty,
DENSE_RANK() OVER(ORDER BY qty DESC) AS d_rnk,
RANK() OVER(ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY 2,1;

+---------+-----+-------+-----+
| salesid | qty | d_rnk | rnk |
+---------+-----+-------+-----+
|   10001 |  10 |     5 |   8 |
|   10006 |  10 |     5 |   8 |
|   30001 |  10 |     5 |   8 |
|   40005 |  10 |     5 |   8 |
|   30003 |  15 |     4 |   7 |
|   20001 |  20 |     3 |   4 |
|   20002 |  20 |     3 |   4 |
|   30004 |  20 |     3 |   4 |
|   10005 |  30 |     2 |   2 |
|   30007 |  30 |     2 |   2 |
|   40001 |  40 |     1 |   1 |
+---------+-----+-------+-----+
```

DENSE_RANK および RANK 関数が同じクエリで並べて使用される場合、同じ行のセットに割り当てるランク付けの違いに注意してください。

sellerid によってテーブルをパーティション分割し、数量によって各パーティションを順序付けして、行ごとにデンス値のランクを割り当てるには、次の例を使用します。結果はウィンドウ関数の結果が提供された後にソートされます。

```
SELECT salesid, sellerid, qty,
DENSE_RANK() OVER(PARTITION BY sellerid ORDER BY qty DESC) AS d_rnk
FROM winsales
ORDER BY 2,3,1;

+---------+----------+-----+-------+
| salesid | sellerid | qty | d_rnk |
+---------+----------+-----+-------+
|   10001 |        1 |  10 |     2 |
|   10006 |        1 |  10 |     2 |
|   10005 |        1 |  30 |     1 |
|   20001 |        2 |  20 |     1 |
|   20002 |        2 |  20 |     1 |
```

```
|   30001 |        3 |  10 |      4 |
|   30003 |        3 |  15 |      3 |
|   30004 |        3 |  20 |      2 |
|   30007 |        3 |  30 |      1 |
|   40005 |        4 |  10 |      2 |
|   40001 |        4 |  40 |      1 |
+---------+----------+-----+-------+
```

最後の例を正しく使用するには、次のコマンドを使用して WINSALES テーブルに行を挿入します。この行には、別の行と同じ buyerid、sellerid、および qtysold があります。これにより、前の例では 2 つの行が同数になり、DENSE_RANK 関数と RANK 関数の違いが示されます。

```
INSERT INTO winsales VALUES(30009, '2/2/2003', 3, 'b', 20, NULL);
```

buyerid と sellerid によってテーブルをパーティション分割し、数量によって各パーティションを順序付けして、行ごとにデンス値のランクと標準のランクの両方を割り当てるには、次の例を使用します。結果はウィンドウ関数が適用された後にソートされます。

```
SELECT salesid, sellerid, qty, buyerid,
DENSE_RANK() OVER(PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS d_rnk,
RANK() OVER (PARTITION BY buyerid, sellerid ORDER BY qty DESC) AS rnk
FROM winsales
ORDER BY rnk;

+---------+----------+-----+---------+-------+-----+
| salesid | sellerid | qty | buyerid | d_rnk | rnk |
+---------+----------+-----+---------+-------+-----+
|   20001 |        2 |  20 | b       |     1 |   1 |
|   30007 |        3 |  30 | c       |     1 |   1 |
|   10006 |        1 |  10 | c       |     1 |   1 |
|   10005 |        1 |  30 | a       |     1 |   1 |
|   20002 |        2 |  20 | c       |     1 |   1 |
|   30009 |        3 |  20 | b       |     1 |   1 |
|   40001 |        4 |  40 | a       |     1 |   1 |
|   30004 |        3 |  20 | b       |     1 |   1 |
|   10001 |        1 |  10 | c       |     1 |   1 |
|   40005 |        4 |  10 | a       |     2 |   2 |
|   30003 |        3 |  15 | b       |     2 |   3 |
|   30001 |        3 |  10 | b       |     3 |   4 |
+---------+----------+-----+---------+-------+-----+
```

FIRST_VALUE ウィンドウ関数

順序付けられた行のセットとすると、FIRST_VALUE はウィンドウフレームの最初の行に関して指定された式の値を返します。

フレームの最後の行を選択する方法については、「LAST_VALUE ウィンドウ関数」を参照してください。

構文

```
FIRST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

IGNORE NULLS

このオプションが FIRST_VALUE で使用される場合、関数は NULL ではないフレームの最初の値 (値がすべて NULL の場合は NULL) を返します。

RESPECT NULLS

Amazon Redshift は使用される行を決定するために null 値を含める必要があることを示します。IGNORE NULLS を指定しない場合、RESPECT NULLS はデフォルトでサポートされます。

OVER

関数にウィンドウ句を導入します。

PARTITION BY expr_list

1 つ以上の式で関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY 句が指定されていない場合、ORDER BY はテーブル全体をソートします。ORDER BY 句を指定する場合、frame_clause も指定する必要があります。

FIRST_VALUE 関数の結果は、データの並び順によって異なります。以下の場合、結果は不確定になります。

- ORDER BY 句が指定されておらず、パーティションに式に使用する 2 つの異なる値が含まれる場合
- 式が ORDER BY リストの同じ値に対応する異なる値を検証する場合。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

戻り型

これらの関数は、Amazon Redshift のプリミティブデータ型を使用する式をサポートします。戻り値の型は式のデータ型と同じです。

例

次の例では、サンプル TICKIT データの VENUE テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

次の例は、収容能力によって順序付けられた結果 (高から低) で、VENUE テーブルの各会場の座席数を返します。FIRST_VALUE 関数は、フレームの最初の行 (この場合、最高座席数の行) に対応する会場名を選択するために使用されます。結果は州によってパーティションで分割されるため、VENUESTATE 値が変更されると、新しい最初の値が選択されます。ウィンドウフレームはバインドされていないため、同じ最初の値が各パーティションの行ごとに選択されます。

カリフォルニアでは、`Qualcomm Stadium`が最高座席数 (`70561`) であるため、この名前は `CA` パーティションのすべての行に対する最初の値です。

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;
```

```
venuestate | venueseats |           venuename            |         first_value
-----------+------------+--------------------------------
+-----------------------------
CA         |      70561 | Qualcomm Stadium               | Qualcomm Stadium
CA         |      69843 | Monster Park                   | Qualcomm Stadium
CA         |      63026 | McAfee Coliseum                | Qualcomm Stadium
CA         |      56000 | Dodger Stadium                 | Qualcomm Stadium
CA         |      45050 | Angel Stadium of Anaheim       | Qualcomm Stadium
CA         |      42445 | PETCO Park                     | Qualcomm Stadium
CA         |      41503 | AT&T Park                      | Qualcomm Stadium
CA         |      22000 | Shoreline Amphitheatre         | Qualcomm Stadium
CO         |      76125 | INVESCO Field                  | INVESCO Field
CO         |      50445 | Coors Field                    | INVESCO Field
DC         |      41888 | Nationals Park                 | Nationals Park
FL         |      74916 | Dolphin Stadium                | Dolphin Stadium
FL         |      73800 | Jacksonville Municipal Stadium | Dolphin Stadium
FL         |      65647 | Raymond James Stadium          | Dolphin Stadium
FL         |      36048 | Tropicana Field                | Dolphin Stadium
...
```

次の例は、IGNORE NULLS オプションの使用方法と VENUE テーブルへの新しい行の追加への見込みを示します。

```
insert into venue values(2000,null,'Stanford','CA',90000);
```

この新しい行には、VENUENAME 行の NULL 値が含まれます。次に、以前にこのセクションで示した FIRST_VALUE クエリを繰り返します。

```
select venuestate, venueseats, venuename,
first_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |         venuename          | first_value
-----------+------------+----------------------------+-------------
CA         |      90000 | NULL                       | NULL
CA         |      70561 | Qualcomm Stadium           | NULL
CA         |      69843 | Monster Park               | NULL
...
```

新しい行には最高値の VENUESEATS 値 (90000) が含まれ、その VENUENAME は NULL となるため、FIRST_VALUE 関数は CA パーティションに NULL を返します。関数式のこのような行を無視するには、IGNORE NULLS オプションを関数の引数に追加します。

```
select venuestate, venueseats, venuename,
first_value(venuename) ignore nulls
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venuestate='CA')
order by venuestate;

venuestate | venueseats |          venuename           |   first_value
------------+------------+----------------------------+------------------
CA         |      90000 | NULL                       | Qualcomm Stadium
CA         |      70561 | Qualcomm Stadium           | Qualcomm Stadium
CA         |      69843 | Monster Park               | Qualcomm Stadium
...
```

LAG ウィンドウ関数

LAG ウィンドウ関数は、パーティションの現在の行より上 (前) の指定されたオフセットの行の値を返します。

構文

```
LAG (value_expr [, offset ])
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

引数

value_expr

関数の対象となる列または式。

offset

値を返す現在の行より前の行数を指定するオプションのパラメータ。オフセットは整数の定数、または整数を検証する式にすることができます。オフセットを指定していない場合、Amazon Redshift はデフォルト値として 1 を使用します。0 のオフセットは現在の行を示します。

IGNORE NULLS

Amazon Redshift が使用する行を決定するときに null 値をスキップすることを指定するオプションの仕様。IGNORE NULLS がリストされていない場合、Null 値が含まれます。

> **Note**
>
> NVL または COALESCE 式を使用し、Null 値を別の値で置換できます。詳細については、「NVL および COALESCE 関数」を参照してください。

RESPECT NULLS

Amazon Redshift は使用される行を決定するために null 値を含める必要があることを示します。IGNORE NULLS を指定しない場合、RESPECT NULLS はデフォルトでサポートされます。

OVER

ウィンドウのパーティションおよび並び順を指定します。OVER 句にウィンドウフレーム仕様を含めることはできません。

PARTITION BY window_partition

OVER 句の各グループのレコードの範囲を設定するオプションの引数。

ORDER BY window_ordering

各パーティション内の行をソートします。

LAG ウィンドウ関数は、Amazon Redshift のデータ型を使用する式をサポートします。戻り値の型は value_expr の型と同じです。

例

次の例は、購入者 ID 3 の購入者に販売されたチケット数と購入者 3 がチケットを購入した時刻を示します。購入者 3 の以前の販売と各販売を比較するには、クエリは販売ごとに以前の販売数を返します。2008 年 1 月 16 日より前に購入されていないため、最初の以前の販売数は Null です。

```
select buyerid, saletime, qtysold,
lag(qtysold,1) over (order by buyerid, saletime) as prev_qtysold
from sales where buyerid = 3 order by buyerid, saletime;
```

```
buyerid |      saletime       | qtysold | prev_qtysold
---------+---------------------+---------+--------------
3 | 2008-01-16 01:06:09 |       1 |
3 | 2008-01-28 02:10:01 |       1 |            1
3 | 2008-03-12 10:39:53 |       1 |            1
3 | 2008-03-13 02:56:07 |       1 |            1
3 | 2008-03-29 08:21:39 |       2 |            1
3 | 2008-04-27 02:39:01 |       1 |            2
3 | 2008-08-16 07:04:37 |       2 |            1
3 | 2008-08-22 11:45:26 |       2 |            2
3 | 2008-09-12 09:11:25 |       1 |            2
3 | 2008-10-01 06:22:37 |       1 |            1
3 | 2008-10-20 01:55:51 |       2 |            1
3 | 2008-10-28 01:30:40 |       1 |            2
(12 rows)
```

LAST_VALUE ウィンドウ関数

順序付けられた行のセットを考慮し、LAST_VALUE 関数は、フレームの最後の行に関する式の値を返します。

フレームの最初の行を選択する方法については、「FIRST_VALUE ウィンドウ関数」を参照してください。

構文

```
LAST_VALUE( expression )[ IGNORE NULLS | RESPECT NULLS ]
OVER (
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

IGNORE NULLS

この関数は NULL ではないフレームの最後の値 (値がすべて NULL の場合は NULL) を返します。

RESPECT NULLS

Amazon Redshift は使用される行を決定するために null 値を含める必要があることを示します。IGNORE NULLS を指定しない場合、RESPECT NULLS はデフォルトでサポートされます。

OVER

関数にウィンドウ句を導入します。

PARTITION BY expr_list

1 つ以上の式で関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY 句が指定されていない場合、ORDER BY はテーブル全体をソートします。ORDER BY 句を指定する場合、frame_clause も指定する必要があります。

結果は、データの並び順によって異なります。以下の場合、結果は不確定になります。

- ORDER BY 句が指定されておらず、パーティションに式に使用する 2 つの異なる値が含まれる場合
- 式が ORDER BY リストの同じ値に対応する異なる値を検証する場合。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

戻り型

これらの関数は、Amazon Redshift のプリミティブデータ型を使用する式をサポートします。戻り値の型は式のデータ型と同じです。

例

次の例では、サンプル TICKIT データの VENUE テーブルを使用します。詳細については、「サンプルデータベース」を参照してください。

次の例は、収容能力によって順序付けられた結果 (高から低) で、VENUE テーブルの各会場の座席数を返します。LAST_VALUE 関数は、フレームの最後の行 (この場合、最小座席数の行) に対

応する会場名を選択するために使用されます。結果は州によってパーティションで分割されるため、VENUESTATE 値が変更されると、新しい最後の値が選択されます。ウィンドウフレームはバインドされていないため、同じ最後の値が各パーティションの行ごとに選択されます。

カリフォルニアでは、`Shoreline Amphitheatre`の座席数が一番低い (22000) ため、この値がパーティションのすべての行に返されます。

```
select venuestate, venueseats, venuename,
last_value(venuename)
over(partition by venuestate
order by venueseats desc
rows between unbounded preceding and unbounded following)
from (select * from venue where venueseats >0)
order by venuestate;

venuestate | venueseats |           venuename              |          last_value
-----------+------------+----------------------------------
+------------------------------
CA         |      70561 | Qualcomm Stadium                 | Shoreline Amphitheatre
CA         |      69843 | Monster Park                     | Shoreline Amphitheatre
CA         |      63026 | McAfee Coliseum                  | Shoreline Amphitheatre
CA         |      56000 | Dodger Stadium                   | Shoreline Amphitheatre
CA         |      45050 | Angel Stadium of Anaheim         | Shoreline Amphitheatre
CA         |      42445 | PETCO Park                       | Shoreline Amphitheatre
CA         |      41503 | AT&T Park                        | Shoreline Amphitheatre
CA         |      22000 | Shoreline Amphitheatre           | Shoreline Amphitheatre
CO         |      76125 | INVESCO Field                    | Coors Field
CO         |      50445 | Coors Field                      | Coors Field
DC         |      41888 | Nationals Park                   | Nationals Park
FL         |      74916 | Dolphin Stadium                  | Tropicana Field
FL         |      73800 | Jacksonville Municipal Stadium   | Tropicana Field
FL         |      65647 | Raymond James Stadium            | Tropicana Field
FL         |      36048 | Tropicana Field                  | Tropicana Field
...
```

LEAD ウィンドウ関数

LEAD ウィンドウ関数は、パーティションの現在の行より下 (後) の指定されたオフセットの行の値を返します。

構文

```
LEAD (value_expr [, offset ])
```

```
[ IGNORE NULLS | RESPECT NULLS ]
OVER ( [ PARTITION BY window_partition ] ORDER BY window_ordering )
```

引数

value_expr

関数の対象となる列または式。

offset

値を返す現在の行より下の行数を指定するオプションのパラメータ。オフセットは整数の定数、または整数を検証する式にすることができます。オフセットを指定していない場合、Amazon Redshift はデフォルト値として 1 を使用します。`0` のオフセットは現在の行を示します。

IGNORE NULLS

Amazon Redshift が使用する行を決定するときに null 値をスキップすることを指定するオプションの仕様。IGNORE NULLS がリストされていない場合、Null 値が含まれます。

> **Note**
>
> NVL または COALESCE 式を使用し、Null 値を別の値で置換できます。詳細については、「NVL および COALESCE 関数」を参照してください。

RESPECT NULLS

Amazon Redshift は使用される行を決定するために null 値を含める必要があることを示します。IGNORE NULLS を指定しない場合、RESPECT NULLS はデフォルトでサポートされます。

OVER

ウィンドウのパーティションおよび並び順を指定します。OVER 句にウィンドウフレーム仕様を含めることはできません。

PARTITION BY window_partition

OVER 句の各グループのレコードの範囲を設定するオプションの引数。

ORDER BY window_ordering

各パーティション内の行をソートします。

LEAD ウィンドウ関数は、Amazon Redshift のデータ型を使用する式をサポートします。戻り値の型は value_expr の型と同じです。

例

次の例は、チケットが 2008 年 1 月 1 日および 2008 年 1 月 2 日に販売された SALES テーブルのイベントの手数料、および次の販売のチケット販売に支払った手数料を示します。次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

```
SELECT eventid, commission, saletime, LEAD(commission, 1) over ( ORDER BY saletime ) AS
 next_comm
FROM sales
WHERE saletime BETWEEN '2008-01-09 00:00:00' AND '2008-01-10 12:59:59'
LIMIT 10;

+---------+------------+---------------------+-----------+
| eventid | commission |      saletime       | next_comm |
+---------+------------+---------------------+-----------+
|    1664 |       13.2 | 2008-01-09 01:00:21 |      69.6 |
|     184 |       69.6 | 2008-01-09 01:00:36 |     116.1 |
|    6870 |      116.1 | 2008-01-09 01:02:37 |      11.1 |
|    3718 |       11.1 | 2008-01-09 01:05:19 |     205.5 |
|    6772 |      205.5 | 2008-01-09 01:14:04 |      38.4 |
|    3074 |       38.4 | 2008-01-09 01:26:50 |     209.4 |
|    5254 |      209.4 | 2008-01-09 01:29:16 |      26.4 |
|    3724 |       26.4 | 2008-01-09 01:40:09 |      57.6 |
|    5303 |       57.6 | 2008-01-09 01:40:21 |      51.6 |
|    3678 |       51.6 | 2008-01-09 01:42:54 |      43.8 |
+---------+------------+---------------------+-----------+
```

次の例では、SALES テーブルのイベントの手数料と、同じイベントの次の販売においてチケット販売に支払った手数料との最大差額を示します。この例は、GROUP BY 句で LEAD を使用する方法を示しています。ウィンドウ関数は集計句では許可されないため、この例ではサブクエリを使用しています。次の例では、TICKIT サンプルデータを使用します。詳細については、「サンプルデータベース」を参照してください。

```
SELECT eventid, eventname, max(next_comm_diff) as max_commission_difference
FROM
(
    SELECT sales.eventid, eventname, commission - LEAD(commission, 1) over (ORDER BY
 sales.eventid, saletime) AS next_comm_diff
```

```
    FROM sales JOIN event ON sales.eventid = event.eventid
)
GROUP BY eventid, eventname
ORDER BY eventid

LIMIT 10

| eventid | eventname                     | max_commission_difference |
+---------+-----------------------------+---------------------------+
| 1       | Gotterdammerung               | 7.95                      |
| 2       | Boris Godunov                 | 227.85                    |
| 3       | Salome                        | 1350.9                    |
| 4       | La Cenerentola (Cinderella)   | 790.05                    |
| 5       | Il Trovatore                  | 214.05                    |
| 6       | L Elisir d Amore              | 510.9                     |
| 7       | Doctor Atomic                 | 180.6                     |
| 9       | The Fly                       | 147                       |
| 10      | Rigoletto                     | 186.6                     |
+---------+-----------------------------+---------------------------+
```

LISTAGG ウィンドウ関数

クエリの各グループについて、LISTAGG ウィンドウ関数は、ORDER BY 式に従ってそのグループの行をソートしてから、それらの値を 1 つの文字列に連結します。

構文

```
LISTAGG( [DISTINCT] expression [, 'delimiter' ] )
[ WITHIN GROUP (ORDER BY order_list) ]
OVER ( [PARTITION BY partition_expression] )
```

引数

DISTINCT

(オプション) 連結する前に、指定された式から重複した値を削除する句。末尾のスペースは無視されるので、'a'と 'a ' という文字列は重複として扱われます。LISTAGG は、発生した最初の値を使用します。詳細については、「末尾の空白の重要性」を参照してください。

aggregate_expression

集計する値を返す任意の有効な式 (列名など)。NULL 値および空の文字列は無視されます。

delimiter

(オプション) 連結された値を区切る文字列定数。デフォルトは NULL です。

WITHIN GROUP (ORDER BY order_list)

(オプション) 集計値のソート順を指定する句。ORDER BY により一意の順序になる場合にのみ確定的です。デフォルトでは、すべての行を集計し、1 つの値を返します。

OVER

ウィンドウのパーティションを指定する句。OVER 句にウィンドウの並び順またはウィンドウフレーム仕様を含めることはできません。

PARTITION BY partition_expression

(オプション) OVER 句のグループごとにレコードの範囲を設定します。

戻り値

VARCHAR(MAX)。結果セットが最大 VARCHAR サイズ (64K - 1、または 65535) より大きい場合、LISTAGG は以下のエラーを返します。

```
Invalid operation: Result size exceeds LISTAGG limit
```

例

次の例は、WINSALES テーブルを使用します。WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

以下の例は、販売者 ID のリストを販売者 ID 順で返します。

```
select listagg(sellerid)
within group (order by sellerid)
over() from winsales;

  listagg
------------
 11122333344
...
...
 11122333344
 11122333344
```

```
(11 rows)
```

以下の例は、購入者 B の販売者 ID のリストを日付順で返します。

```
select listagg(sellerid)
within group (order by dateid)
over () as seller
from winsales
where buyerid = 'b' ;

  seller
---------
    3233
    3233
    3233
    3233
```

以下の例は、購入者 B について販売日付のカンマ区切りリストを返します。

```
select listagg(dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

             dates
-------------------------------------------
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-04-18,2004-02-12
```

次の例は、DISTINCT を使用して、購入者 B の一意の販売日のリストを返します。

```
select listagg(distinct dateid,',')
within group (order by sellerid desc,salesid asc)
over () as dates
from winsales
where buyerid  = 'b';

           dates
--------------------------------
```

```
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
2003-08-02,2004-04-18,2004-02-12
```

以下の例は、各購入者 ID について販売 ID のカンマ区切りリストを返します。

```
select buyerid,
listagg(salesid,',')
within group (order by salesid)
over (partition by buyerid) as sales_id
from winsales
order by buyerid;

+---------+-------------------------+
| buyerid |        sales_id         |
+---------+-------------------------+
| a       | 10005,40001,40005       |
| a       | 10005,40001,40005       |
| a       | 10005,40001,40005       |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| b       | 20001,30001,30003,30004 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
| c       | 10001,10006,20002,30007 |
+---------+-------------------------+
```

MAX ウィンドウ関数

MAX ウィンドウ関数は入力式の最大値を返します。MAX 関数は数値に対してはたらき、NULL 値は無視します。

構文

```
MAX ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数は式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数のウィンドウ句を指定する句。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で MAX 関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

データ型を入力として受け入れます。同じデータ型を expression として返します。

例

次の例では、データウィンドウの先頭から販売 ID、数量、最大数を示します。

```
select salesid, qty,
max(qty) over (order by salesid rows unbounded preceding) as max
from winsales
order by salesid;

salesid | qty | max
```

```
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  30
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  30
30001 |  10 |  30
30003 |  15 |  30
30004 |  20 |  30
30007 |  30 |  30
40001 |  40 |  40
40005 |  10 |  40
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、制限されたフレームで販売 ID、数量、および最大数を示します。

```
select salesid, qty,
max(qty) over (order by salesid rows between 2 preceding and 1 preceding) as max
from winsales
order by salesid;

salesid | qty | max
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  30
20001 |  20 |  30
20002 |  20 |  20
30001 |  10 |  20
30003 |  15 |  20
30004 |  20 |  15
30007 |  30 |  20
40001 |  40 |  30
40005 |  10 |  40
(11 rows)
```

MEDIAN ウィンドウ関数

ウィンドウまたはパーティションの値の範囲について、その中央値を計算します。範囲の Null 値は無視されます。

MEDIAN は、連続型分散モデルを前提とする逆分散関数です。

構文

```
MEDIAN ( median_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

引数

median_expression

中央値を特定する値を提供する式 (列名など)。式は、数値または日時データ型を含んでいるか、それらのデータ型に暗黙的に変換できる必要があります。

OVER

ウィンドウのパーティションを指定する句。OVER 句にウィンドウの並び順またはウィンドウフレーム仕様を含めることはできません。

PARTITION BY partition_expression

省略可能。OVER 句の各グループのレコードの範囲を設定する式。

データ型

戻り値の型は、データ型 median_expression によって決まります。次の表は、各 median_expression 式のデータ型に対応する戻り型を示しています。

入力の型	戻り型
INT2、INT4、INT8、NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE

使用に関する注意事項

median_expression 引数が DECIMAL データ型であり、その最大精度が 38 桁である場合、MEDIAN が不正確な結果またはエラーを返す可能性があります。MEDIAN 関数の戻り値が 38 桁を超える場

合、結果は 38 桁までとなり、39 桁以降は切り捨てられるため、精度が失われます。補間中に中間結果が最大精度を超えた場合には、数値オーバーフローが発生し、この関数はエラーを返します。このような状態を回避するため、精度が低いデータ型を使用するか、median_expression 引数を低い精度にキャストすることをお勧めします。

例えば、DECIMAL 引数の SUM 関数のデフォルトの 38 桁の精度を返します。結果のスケールは、引数のスケールと同じです。したがって、例えば、DECIMAL(5,2) 列の SUM は DECIMAL(38,2) データ型を返します。

次の例では、MEDIAN 関数の median_expression 引数で SUM 関数を使用します。PRICEPAID 列のデータ型は DECIMAL(8,2) であるため、SUM 関数は DECIMAL(38,2) を返します。

```
select salesid, sum(pricepaid), median(sum(pricepaid))
over() from sales where salesid < 10 group by salesid;
```

精度の損失またはオーバーフローエラーを回避するには、次の例が示すように、精度が低い DECIMAL データ型に結果をキャストします。

```
select salesid, sum(pricepaid), median(sum(pricepaid)::decimal(30,2))
over() from sales where salesid < 10 group by salesid;
```

例

以下の例では、各販売者の平均販売数量を計算します。

```
select sellerid, qty, median(qty)
over (partition by sellerid)
from winsales
order by sellerid;


sellerid qty median
---------------------------
1  10 10.0
1  10 10.0
1  30 10.0
2  20 20.0
2  20 20.0
3  10 17.5
3  15 17.5
3  20 17.5
```

```
3   30 17.5
4   10 25.0
4   40 25.0
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

MIN ウィンドウ関数

MIN ウィンドウ関数は入力式の最小値を返します。MIN 関数は数値に対してはたらき、NULL 値は無視します。

構文

```
MIN ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数は式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で MIN 関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

データ型を入力として受け入れます。同じデータ型を expression として返します。

例

次の例では、データウィンドウの先頭から販売 ID、数量、最小数を示します。

```
select salesid, qty,
min(qty) over
(order by salesid rows unbounded preceding)
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |  10
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  10
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  10
40001 |  40 |  10
40005 |  10 |  10
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、制限されたフレームで販売 ID、数量、および最小数を示します。

```
select salesid, qty,
```

```
min(qty) over
(order by salesid rows between 2 preceding and 1 preceding) as min
from winsales
order by salesid;

salesid | qty | min
---------+-----+-----
10001 |  10 |
10005 |  30 |  10
10006 |  10 |  10
20001 |  20 |  10
20002 |  20 |  10
30001 |  10 |  20
30003 |  15 |  10
30004 |  20 |  10
30007 |  30 |  15
40001 |  40 |  20
40005 |  10 |  30
(11 rows)
```

NTH_VALUE ウィンドウ関数

NTH_VALUE ウィンドウ関数は、ウィンドウの最初の行に関連するウィンドウフレームの指定された行の式の値を返します。

構文

```
NTH_VALUE (expr, offset)
[ IGNORE NULLS | RESPECT NULLS ]
OVER
( [ PARTITION BY window_partition ]
[ ORDER BY window_ordering
                        frame_clause ] )
```

引数

expr

関数の対象となる列または式。

offset

式を返すウィンドウの最初の行に関連する行数を決定します。offset は定数または式にすることができ、0 より大きい正の整数である必要があります。

IGNORE NULLS

Amazon Redshift が使用する行を決定するときに null 値をスキップすることを指定するオプションの仕様。IGNORE NULLS がリストされていない場合、Null 値が含まれます。

RESPECT NULLS

Amazon Redshift は使用される行を決定するために null 値を含める必要があることを示します。IGNORE NULLS を指定しない場合、RESPECT NULLS はデフォルトでサポートされます。

OVER

ウィンドウのパーティション、並び順、およびウィンドウフレームを指定します。

PARTITION BY window_partition

OVER 句のグループごとにレコードの範囲を設定します。

ORDER BY window_ordering

各パーティション内の行をソートします。ORDER BY が省略される場合、デフォルトのフレームはパーティションのすべての行から構成されます。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

NTH_VALUE ウィンドウ関数は、Amazon Redshift のデータ型を使用する式をサポートします。戻り値の型は expr の型と同じです。

例

次の例は、州のその他の会場の座席数を比較して、カリフォルニア、フロリダ、およびニューヨークで 3 番目に大きい会場の座席数を示しています。

```
select venuestate, venuename, venueseats,
nth_value(venueseats, 3)
ignore nulls
over(partition by venuestate order by venueseats desc
rows between unbounded preceding and unbounded following)
as third_most_seats
```

```
from (select * from venue where venueseats > 0 and
venuestate in('CA', 'FL', 'NY'))
order by venuestate;

venuestate |           venuename            | venueseats | third_most_seats
------------+--------------------------------+------------+------------------
CA         | Qualcomm Stadium               |      70561 |            63026
CA         | Monster Park                   |      69843 |            63026
CA         | McAfee Coliseum                |      63026 |            63026
CA         | Dodger Stadium                 |      56000 |            63026
CA         | Angel Stadium of Anaheim       |      45050 |            63026
CA         | PETCO Park                     |      42445 |            63026
CA         | AT&T Park                      |      41503 |            63026
CA         | Shoreline Amphitheatre         |      22000 |            63026
FL         | Dolphin Stadium                |      74916 |            65647
FL         | Jacksonville Municipal Stadium |      73800 |            65647
FL         | Raymond James Stadium          |      65647 |            65647
FL         | Tropicana Field                |      36048 |            65647
NY         | Ralph Wilson Stadium           |      73967 |            20000
NY         | Yankee Stadium                 |      52325 |            20000
NY         | Madison Square Garden          |      20000 |            20000
(15 rows)
```

NTILE ウィンドウ関数

NTILE ウィンドウ関数は、パーティションの順序付けされた行を可能な限り同じサイズの指定されたランク付けグループの数に分割し、任意の行を分類するグループを返します。

構文

```
NTILE (expr)
OVER (
[ PARTITION BY expression_list ]
[ ORDER BY order_list ]
)
```

引数

expr

ランク付けグループの数。パーティションごとに正の整数値 (0 より大きい) になる必要があります。expr 引数は Null を使用することはできません。

OVER

ウィンドウのパーティションと順序を指定する句。OVER 句にウィンドウフレーム仕様を含めることはできません。

PARTITION BY window_partition

省略可能。OVER 句のグループごとのレコードの範囲。

ORDER BY window_ordering

省略可能。各パーティション内の行をソートする式。ORDER BY 句を省略した場合、ランク付けの動作は同じです。

ORDER BY で一意のソートが行われない場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

BIGINT

例

次の例は、2008 年 8 月 26 日に Hamlet のチケットに支払った価格を 4 つのランクグループにランク付けします。結果セットは 17 行であり、1 位から 4 位のランク付けにほぼ均等に分割されます。

```
select eventname, caldate, pricepaid, ntile(4)
over(order by pricepaid desc) from sales, event, date
where sales.eventid=event.eventid and event.dateid=date.dateid and eventname='Hamlet'
and caldate='2008-08-26'
order by 4;

eventname |  caldate   | pricepaid | ntile
-----------+------------+-----------+-------
Hamlet    | 2008-08-26 |   1883.00 |     1
Hamlet    | 2008-08-26 |   1065.00 |     1
Hamlet    | 2008-08-26 |    589.00 |     1
Hamlet    | 2008-08-26 |    530.00 |     1
Hamlet    | 2008-08-26 |    472.00 |     1
Hamlet    | 2008-08-26 |    460.00 |     2
Hamlet    | 2008-08-26 |    355.00 |     2
Hamlet    | 2008-08-26 |    334.00 |     2
Hamlet    | 2008-08-26 |    296.00 |     2
Hamlet    | 2008-08-26 |    230.00 |     3
```

```
Hamlet    | 2008-08-26 |    216.00 |      3
Hamlet    | 2008-08-26 |    212.00 |      3
Hamlet    | 2008-08-26 |    106.00 |      3
Hamlet    | 2008-08-26 |    100.00 |      4
Hamlet    | 2008-08-26 |     94.00 |      4
Hamlet    | 2008-08-26 |     53.00 |      4
Hamlet    | 2008-08-26 |     25.00 |      4
(17 rows)
```

PERCENT_RANK ウィンドウ関数

指定の行のパーセントランクを計算します。パーセントランクは、以下の式を使用して特定されます。

`(x - 1) / (the number of rows in the window or partition - 1)`

ここで、x は現在の行のランクです。以下のデータセットは、この式の使用方法を示しています。

```
Row# Value Rank Calculation PERCENT_RANK
1 15 1 (1-1)/(7-1) 0.0000
2 20 2 (2-1)/(7-1) 0.1666
3 20 2 (2-1)/(7-1) 0.1666
4 20 2 (2-1)/(7-1) 0.1666
5 30 5 (5-1)/(7-1) 0.6666
6 30 5 (5-1)/(7-1) 0.6666
7 40 7 (7-1)/(7-1) 1.0000
```

戻り値の範囲は、0～1 (0 と 1 を含みます) です。どのセットも、最初の行の PERCENT_RANK は 0 になります。

構文

```
PERCENT_RANK ()
OVER (
[ PARTITION BY partition_expression ]
[ ORDER BY order_list ]
)
```

引数

( )

この関数は引数を受け取りませんが、空のかっこは必要です。

OVER

ウィンドウのパーティションを指定する句。OVER 句にウィンドウフレーム仕様を含めることはできません。

PARTITION BY partition_expression

省略可能。OVER 句の各グループのレコードの範囲を設定する式。

ORDER BY order_list

省略可能。パーセントランクを計算する式。式は、数値データ型を含んでいるか、そのデータ型に暗黙的に変換できる必要があります。ORDER BY を省略した場合、すべての行について戻り値は 0 です。

ORDER BY で一意のソートが行われない場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

FLOAT8

例

以下の例では、各販売者の販売数量のパーセントランクを計算します。

```
select sellerid, qty, percent_rank()
over (partition by sellerid order by qty)
from winsales;

sellerid qty  percent_rank
----------------------------------------
1  10.00  0.0
1  10.64  0.5
1  30.37  1.0
3  10.04  0.0
3  15.15  0.33
3  20.75  0.67
3  30.55  1.0
2  20.09  0.0
2  20.12  1.0
4  10.12  0.0
```

```
4  40.23  1.0
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

PERCENTILE_CONT ウィンドウ関数

PERCENTILE_CONT は、連続型分散モデルを前提とする逆分散関数です。これは、パーセンタイル値とソート仕様を取得し、ソート仕様を基準として、そのパーセンタイル値に該当する補間値を返します。

PERCENTILE_CONT は、値の順序付けを行った後に値の間の線形補間を計算します。この関数は、パーセンタイル値 (P) と集計グループの Null ではない行の数 (N) を使用して、ソート使用に従って行の順序付けを行った後に行番号を計算します。この行番号 (RN) は、計算式 RN = (1+ (P*(N-1)) に従って計算されます。集計関数の最終結果は、行番号 CRN = CEILING(RN) および FRN = FLOOR(RN) にある行の値の間の線形補間に基づいて計算されます。

最終結果は次のとおりです。

(CRN = FRN = RN) である場合、結果は (value of expression from row at RN)

そうでない場合、結果は

(CRN - RN) * (value of expression for row at FRN) + (RN - FRN) * (value of expression for row at CRN).

OVER 句では PARTITION 句のみを指定できます。各行に対して PARTITION を指定すると、PERCENTILE_CONT は、特定のパーティション内にある一連の値から、指定したパーセンタイルに該当する値を返します。

構文

```
PERCENTILE_CONT ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

引数

percentile

0 と 1 の間の数値定数。Null は計算では無視されます。

WITHIN GROUP (ORDER BY expr)

パーセンタイルをソートして計算するための数値または日付/時間値を指定します。

OVER

ウィンドウのパーティションを指定します。OVER 句にウィンドウの並び順またはウィンドウフレーム仕様を含めることはできません。

PARTITION BY expr

OVER 句の各グループのレコードの範囲を設定するオプションの引数。

戻り値

戻り型は、WITHIN GROUP 句の ORDER BY 式のデータ型に基づいて決定されます。次の表は、各 ORDER BY 式のデータ型に対応する戻り型を示しています。

入力の型	戻り型
INT2、INT4、INT8、NUMERIC、DECIMAL	DECIMAL
FLOAT、DOUBLE	DOUBLE
DATE	DATE
TIMESTAMP	TIMESTAMP

使用に関する注意事項

ORDER BY 式が DECIMAL データ型であり、その最大精度が 38 桁である場合、PERCENTILE_CONT が不正確な結果またはエラーを返す可能性があります。PERCENTILE_CONT 関数の戻り値が 38 桁を超える場合、結果は 38 桁までとなり、39 桁以降は切り捨てられるため、精度が失われます。補間中に中間結果が最大精度を超えた場合には、数値オーバーフローが発生し、この関数はエラーを返します。このような状態を回避するため、精度が低いデータ型を使用するか、ORDER BY 式を低い精度にキャストすることをお勧めします。

例えば、DECIMAL 引数の SUM 関数のデフォルトの 38 桁の精度を返します。結果のスケールは、引数のスケールと同じです。したがって、例えば、DECIMAL(5,2) 列の SUM は DECIMAL(38,2) データ型を返します。

次の例は、PERCENTILE_CONT 関数の ORDER BY 句で SUM 関数を使用します。PRICEPAID 列のデータ型は DECIMAL(8,2) であるため、SUM 関数は DECIMAL(38,2) を返します。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid) desc) over()
from sales where salesid < 10 group by salesid;
```

精度の損失またはオーバーフローエラーを回避するには、次の例が示すように、精度が低い DECIMAL データ型に結果をキャストします。

```
select salesid, sum(pricepaid), percentile_cont(0.6)
within group (order by sum(pricepaid)::decimal(30,2) desc) over()
from sales where salesid < 10 group by salesid;
```

例

次の例は、WINSALES テーブルを使用します。WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over() as median from winsales;

 sellerid | qty | median
----------+-----+--------
        1 |  10 |   20.0
        1 |  10 |   20.0
        3 |  10 |   20.0
        4 |  10 |   20.0
        3 |  15 |   20.0
        2 |  20 |   20.0
        3 |  20 |   20.0
        2 |  20 |   20.0
        3 |  30 |   20.0
        1 |  30 |   20.0
        4 |  40 |   20.0
(11 rows)
```

```
select sellerid, qty, percentile_cont(0.5)
within group (order by qty)
over(partition by sellerid) as median from winsales;
```

```
 sellerid | qty | median
----------+-----+--------
        2 |  20 |   20.0
        2 |  20 |   20.0
        4 |  10 |   25.0
        4 |  40 |   25.0
        1 |  10 |   10.0
        1 |  10 |   10.0
        1 |  30 |   10.0
        3 |  10 |   17.5
        3 |  15 |   17.5
        3 |  20 |   17.5
        3 |  30 |   17.5
(11 rows)
```

次の例は、ワシントン州の販売者のチケット販売の PERCENTILE_CONT と PERCENTILE_DISC を計算します。

```
SELECT sellerid, state, sum(qtysold*pricepaid) sales,
percentile_cont(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over(),
percentile_disc(0.6) within group (order by sum(qtysold*pricepaid::decimal(14,2) )
 desc) over()
from sales s, users u
where s.sellerid = u.userid and state = 'WA' and sellerid < 1000
group by sellerid, state;

 sellerid | state |  sales  | percentile_cont | percentile_disc
----------+-------+---------+-----------------+-----------------
      127 | WA    | 6076.00 |         2044.20 |         1531.00
      787 | WA    | 6035.00 |         2044.20 |         1531.00
      381 | WA    | 5881.00 |         2044.20 |         1531.00
      777 | WA    | 2814.00 |         2044.20 |         1531.00
       33 | WA    | 1531.00 |         2044.20 |         1531.00
      800 | WA    | 1476.00 |         2044.20 |         1531.00
        1 | WA    | 1177.00 |         2044.20 |         1531.00
(7 rows)
```

PERCENTILE_DISC ウィンドウ関数

PERCENTILE_DISC は、離散型分散モデルを前提とする逆分散関数です。これは、パーセンタイル値とソート仕様を取得し、特定のセットからエレメントを返します。

特定のパーセンタイル値が P である場合、PERCENTILE_DISC は ORDER BY 句の式の値をソートし、P と同じであるか P より大きい最少累積分散値 (同じソート仕様を基準とする) を持つ値を返します。

OVER 句では PARTITION 句のみを指定できます。

構文

```
PERCENTILE_DISC ( percentile )
WITHIN GROUP (ORDER BY expr)
OVER (  [ PARTITION BY expr_list ]  )
```

引数

percentile

0 と 1 の間の数値定数。Null は計算では無視されます。

WITHIN GROUP (ORDER BY expr)

パーセンタイルをソートして計算するための数値または日付/時間値を指定します。

OVER

ウィンドウのパーティションを指定します。OVER 句にウィンドウの並び順またはウィンドウフレーム仕様を含めることはできません。

PARTITION BY expr

OVER 句の各グループのレコードの範囲を設定するオプションの引数。

戻り値

WITHIN GROUP 句の ORDER BY 式と同じデータ型。

例

次の例は、WINSALES テーブルを使用しています。WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

```
SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER() AS MEDIAN FROM winsales;
```

```
+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 3        | 10  | 20     |
| 1        | 10  | 20     |
| 1        | 10  | 20     |
| 4        | 10  | 20     |
| 3        | 15  | 20     |
| 2        | 20  | 20     |
| 2        | 20  | 20     |
| 3        | 20  | 20     |
| 1        | 30  | 20     |
| 3        | 30  | 20     |
| 4        | 40  | 20     |
+----------+-----+--------+

SELECT sellerid, qty, PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS MEDIAN FROM winsales;

+----------+-----+--------+
| sellerid | qty | median |
+----------+-----+--------+
| 4        | 10  | 10     |
| 4        | 40  | 10     |
| 3        | 10  | 15     |
| 3        | 15  | 15     |
| 3        | 20  | 15     |
| 3        | 30  | 15     |
| 2        | 20  | 20     |
| 2        | 20  | 20     |
| 1        | 10  | 10     |
| 1        | 10  | 10     |
| 1        | 30  | 10     |
+----------+-----+--------+
```

販売者 ID ごとの数量の PERCENTILE_DISC (0.25) と PERCENTILE_DISC (0.75) を求めるには、以下の例を使用します。

```
SELECT sellerid, qty, PERCENTILE_DISC(0.25)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile1 FROM winsales;
```

```
+----------+-----+-----------+
| sellerid | qty | quartile1 |
+----------+-----+-----------+
| 4        | 10  | 10        |
| 4        | 40  | 10        |
| 2        | 20  | 20        |
| 2        | 20  | 20        |
| 3        | 10  | 10        |
| 3        | 15  | 10        |
| 3        | 20  | 10        |
| 3        | 30  | 10        |
| 1        | 10  | 10        |
| 1        | 10  | 10        |
| 1        | 30  | 10        |
+----------+-----+-----------+

SELECT sellerid, qty, PERCENTILE_DISC(0.75)
WITHIN GROUP (ORDER BY qty)
OVER(PARTITION BY sellerid) AS quartile3 FROM winsales;

+----------+-----+-----------+
| sellerid | qty | quartile3 |
+----------+-----+-----------+
| 3        | 10  | 20        |
| 3        | 15  | 20        |
| 3        | 20  | 20        |
| 3        | 30  | 20        |
| 4        | 10  | 40        |
| 4        | 40  | 40        |
| 2        | 20  | 20        |
| 2        | 20  | 20        |
| 1        | 10  | 30        |
| 1        | 10  | 30        |
| 1        | 30  | 30        |
+----------+-----+-----------+
```

RANK ウィンドウ関数

RANK ウィンドウ関数は、OVER 句の ORDER BY 式に基づいて、値のグループの値のランクを決定します。オプションの PARTITION BY 句がある場合、ランク付けは行のグループごとにリセットされます。ランク付け条件が同じ値の行は、同じランクを受け取ります。Amazon Redshift は、同順位

である行の数を同ランクに追加して次のランクを計算するため、ランクが連続した数にならない場合があります。例えば、2 行が 1 位にランク付けされると、次のランクは 3 位になります。

RANK と DENSE_RANK ウィンドウ関数 では異なる点があり、DENSE_RANK では、2 行以上で同点となった場合、ランク付けされた値の順位に差はありません。例えば、2 行が 1 位にランク付けされると、次のランクは 2 位になります。

同じクエリに PARTITION BY および ORDER BY 句のあるランク付け関数を使用することができます。

構文

```
RANK () OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list ]
)
```

引数

( )

この関数は引数を受け取りませんが、空のかっこは必要です。

OVER

RANK 関数のウィンドウ句。

PARTITION BY expr_list

省略可能。ウィンドウを定義する 1 つ以上の式。

ORDER BY order_list

省略可能。ランク付けの値が基とする列を定義します。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。ORDER BY を省略した場合、すべての行について戻り値は 1 です。

ORDER BY で一意のソートが行われない場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

INTEGER

例

次の例では、販売数量でテーブルで順序付けして (デフォルトは昇順)、行ごとにランクを割り当てます。1 のランク値は、もっとも高いランクの値です。結果はウィンドウ関数の結果が提供された後にソートされます。

```
select salesid, qty,
rank() over (order by qty) as rnk
from winsales
order by 2,1;

salesid | qty | rnk
--------+-----+-----
10001 |  10 |  1
10006 |  10 |  1
30001 |  10 |  1
40005 |  10 |  1
30003 |  15 |  5
20001 |  20 |  6
20002 |  20 |  6
30004 |  20 |  6
10005 |  30 |  9
30007 |  30 |  9
40001 |  40 |  11
(11 rows)
```

この例の ORDER BY 句の外側には、Amazon Redshift がこのクエリが実行されるごとにソートされた結果を一貫して返すように、列 2 および 1 が含まれることに注意してください。例えば、販売 ID 10001 および 10006 の行は、QTY と RNK の値が同じです。列 1 によって最終的な結果セットを順序付けると、行 10001 は常に 10006 の前になります。WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、ウィンドウ関数 (`order by qty desc`) の順序付けは逆順になります。これで、最高ランク値が最大の QTY 値に適用されます。

```
select salesid, qty,
rank() over (order by qty desc) as rank
from winsales
order by 2,1;

 salesid | qty | rank
---------+-----+-----
```

```
   10001 |  10 |   8
   10006 |  10 |   8
   30001 |  10 |   8
   40005 |  10 |   8
   30003 |  15 |   7
   20001 |  20 |   4
   20002 |  20 |   4
   30004 |  20 |   4
   10005 |  30 |   2
   30007 |  30 |   2
   40001 |  40 |   1
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、SELLERID によってテーブルをパーティション分割し、数量で各パーティションを順序付けして (降順)、行ごとにランクを割り当てます。結果はウィンドウ関数の結果が提供された後にソートされます。

```
select salesid, sellerid, qty, rank() over
(partition by sellerid
order by qty desc) as rank
from winsales
order by 2,3,1;

salesid | sellerid | qty | rank
--------+----------+-----+-----
  10001 |        1 |  10 |  2
  10006 |        1 |  10 |  2
  10005 |        1 |  30 |  1
  20001 |        2 |  20 |  1
  20002 |        2 |  20 |  1
  30001 |        3 |  10 |  4
  30003 |        3 |  15 |  3
  30004 |        3 |  20 |  2
  30007 |        3 |  30 |  1
  40005 |        4 |  10 |  2
  40001 |        4 |  40 |  1
(11 rows)
```

RATIO_TO_REPORT ウィンドウ関数

ウィンドウまたはパーティションの値の合計に対する、ある値の比率を計算します。値を報告する比率は、次の式を使用して特定されます。

`value of` ratio_expression `argument for the current row / sum of` ratio_expression `argument for the window or partition`

以下のデータセットは、この式の使用方法を示しています。

```
Row# Value Calculation RATIO_TO_REPORT
1 2500 (2500)/(13900) 0.1798
2 2600 (2600)/(13900) 0.1870
3 2800 (2800)/(13900) 0.2014
4 2900 (2900)/(13900) 0.2086
5 3100 (3100)/(13900) 0.2230
```

戻り値の範囲は、0～1 (0 と 1 を含みます) です。ratio_expression が NULL である場合、戻り値は NULL です。partition_expression 内の値が一意である場合、関数はその値として 1 を返します。

構文

```
RATIO_TO_REPORT ( ratio_expression )
OVER ( [ PARTITION BY partition_expression ] )
```

引数

ratio_expression

比率を特定する値を提供する式 (列名など)。式は、数値データ型を含んでいるか、そのデータ型に暗黙的に変換できる必要があります。

ratio_expression で他の分析関数を使用することはできません。

OVER

ウィンドウのパーティションを指定する句。OVER 句にウィンドウの並び順またはウィンドウフレーム仕様を含めることはできません。

PARTITION BY partition_expression

省略可能。OVER 句の各グループのレコードの範囲を設定する式。

戻り型

FLOAT8

例

次の例は、WINSALES テーブルを使用しています。WINSALES テーブルを作成する方法については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、すべての販売者の数量の合計に対する、各行の販売者の数量の、対レポート比率を計算します。

```
select sellerid, qty, ratio_to_report(qty)
over()
from winsales
order by sellerid;

sellerid  qty    ratio_to_report
--------------------------------------
1         30     0.13953488372093023
1         10     0.046511627906976744
1         10     0.046511627906976744
2         20     0.09302325581395349
2         20     0.09302325581395349
3         30     0.13953488372093023
3         20     0.09302325581395349
3         15     0.06976744186046512
3         10     0.046511627906976744
4         10     0.046511627906976744
4         40     0.18604651162790697
```

以下の例では、各販売者の販売数量の比率をパーティションごとに計算します。

```
select sellerid, qty, ratio_to_report(qty)
over(partition by sellerid)
from winsales;

sellerid   qty    ratio_to_report
-------------------------------------------
2          20     0.5
2          20     0.5
4          40     0.8
```

```
4          10     0.2
1          10     0.2
1          30     0.6
1          10     0.2
3          10     0.13333333333333333
3          15     0.2
3          20     0.26666666666666666
3          30     0.4
```

ROW_NUMBER ウィンドウ関数

OVER 句の ORDER BY 式に基づいて、行グループ内の現在の行の (1 からカウントした) 序数を割り当てます。オプションの PARTITION BY 句がある場合、序数は行グループごとにリセットされます。ORDER BY 式で同じ値を持つ行には、確定的でない方法で異なる行番号が割り当てられます。

構文

```
ROW_NUMBER() OVER(
  [ PARTITION BY expr_list ]
  [ ORDER BY order_list ]
)
```

引数

( )

この関数は引数を受け取りませんが、空のかっこは必要です。

OVER

ROW_NUMBER 関数のウィンドウ関数句。

PARTITION BY expr_list

オプション。結果を行のセットに分割する 1 つ以上の列式。

ORDER BY order_list

オプション。セット内の行の順序を定義する 1 つ以上の列式。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

ORDER BY で一意の順序付けが行われない、または省略した場合、行の順序は不確定になります。詳細については、「ウィンドウ関数用データの一意の並び順」を参照してください。

戻り型

BIGINT

例

以下の例では WINSALES テーブルを使用します。WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、QTY によってテーブルを (昇順で) 順序付け、各行に行番号を割り当てます。結果はウィンドウ関数の結果が提供された後にソートされます。

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
   ORDER BY qty ASC) AS row
FROM winsales
ORDER BY 4,1;

salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |    1
   10001 |        1 |  10 |    2
   10006 |        1 |  10 |    3
   40005 |        4 |  10 |    4
   30003 |        3 |  15 |    5
   20001 |        2 |  20 |    6
   20002 |        2 |  20 |    7
   30004 |        3 |  20 |    8
   10005 |        1 |  30 |    9
   30007 |        3 |  30 |   10
   40001 |        4 |  40 |   11
```

次の例では、SELLERID によってテーブルをパーティション化し、QTY によって各パーティションを (昇順で) 順序付けし、各行に行番号を割り当てます。結果はウィンドウ関数の結果が提供された後にソートされます。

```
SELECT salesid, sellerid, qty,
ROW_NUMBER() OVER(
  PARTITION BY sellerid
  ORDER BY qty ASC) AS row_by_seller
FROM winsales
ORDER BY 2,4;
```

```
 salesid | sellerid | qty | row_by_seller
---------+----------+-----+-----
   10001 |        1 |  10 |    1
   10006 |        1 |  10 |    2
   10005 |        1 |  30 |    3
   20001 |        2 |  20 |    1
   20002 |        2 |  20 |    2
   30001 |        3 |  10 |    1
   30003 |        3 |  15 |    2
   30004 |        3 |  20 |    3
   30007 |        3 |  30 |    4
   40005 |        4 |  10 |    1
   40001 |        4 |  40 |    2
```

次の例は、オプション句を使用しない場合の結果を示しています。

```
SELECT salesid, sellerid, qty, ROW_NUMBER() OVER() AS row
FROM winsales
ORDER BY 4,1;

salesid   sellerid   qty   row
---------+----------+-----+-----
   30001 |        3 |  10 |    1
   10001 |        1 |  10 |    2
   10005 |        1 |  30 |    3
   40001 |        4 |  40 |    4
   10006 |        1 |  10 |    5
   20001 |        2 |  20 |    6
   40005 |        4 |  10 |    7
   20002 |        2 |  20 |    8
   30003 |        3 |  15 |    9
   30004 |        3 |  20 |   10
   30007 |        3 |  30 |   11
```

STDDEV_SAMP および STDDEV_POP ウィンドウ関数

STDDEV_SAMP および STDDEV_POP ウィンドウ関数は、数値のセットの標本標準偏差および母集団標準偏差を返します (整数、10 進数、または浮動小数点)。「STDDEV_SAMP および STDDEV_POP 関数」も参照してください。

STDDEV_SAMP および STDDEV は、同じ関数のシノニムです。

構文

```
STDDEV_SAMP | STDDEV | STDDEV_POP
( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数は式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

STDDEV 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、および DOUBLE PRECISION です。

式のデータ型にかかわらず、STDDEV 関数の戻り値の型は倍精度浮動小数点数です。

例

次の例は、ウィンドウ関数として STDDEV_POP および VAR_POP 関数を使用する方法を示します。このクエリは SALES テーブルの PRICEPAID の値の母集団分散と母集団標準偏差を計算します。

```
select salesid, dateid, pricepaid,
round(stddev_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as stddevpop,
round(var_pop(pricepaid) over
(order by dateid, salesid rows unbounded preceding)) as varpop
from sales
order by 2,1;

salesid | dateid | pricepaid | stddevpop | varpop
--------+--------+-----------+-----------+---------
  33095 |   1827 |    234.00 |         0 |       0
  65082 |   1827 |    472.00 |       119 |   14161
  88268 |   1827 |    836.00 |       248 |   61283
  97197 |   1827 |    708.00 |       230 |   53019
 110328 |   1827 |    347.00 |       223 |   49845
 110917 |   1827 |    337.00 |       215 |   46159
 150314 |   1827 |    688.00 |       211 |   44414
 157751 |   1827 |   1730.00 |       447 |  199679
 165890 |   1827 |   4192.00 |      1185 | 1403323
...
```

例の標本標準偏差および標本分散の関数も同様に使用することができます。

SUM ウィンドウ関数

SUM ウィンドウ関数は入力列または式の値の合計を返します。SUM 関数は数値に対してはたらき、NULL 値を無視します。

構文

```
SUM ( [ ALL ] expression ) OVER
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数は式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で SUM 関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

SUM 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、および DOUBLE PRECISION です。

SUM 関数でサポートされる戻り値の型は次のとおりです。

- SMALLINT または INTEGER 引数の場合は BIGINT
- BIGINT 引数の場合は NUMERIC
- 浮動小数点の引数の場合は DOUBLE PRECISION

例

次の例では、日付および販売 ID によって順序付けされた販売数量の累積 (中間) 合計を作成します。

```
select salesid, dateid, sellerid, qty,
sum(qty) over (order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  20
10005 | 2003-12-24 |        1 |  30 |  50
40001 | 2004-01-09 |        4 |  40 |  90
10006 | 2004-01-18 |        1 |  10 | 100
20001 | 2004-02-12 |        2 |  20 | 120
40005 | 2004-02-12 |        4 |  10 | 130
20002 | 2004-02-16 |        2 |  20 | 150
30003 | 2004-04-18 |        3 |  15 | 165
30004 | 2004-04-18 |        3 |  20 | 185
30007 | 2004-09-07 |        3 |  30 | 215
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、日付による販売数量の累積 (中間) 合計を作成し、結果を販売者 ID でパーティション分割して、パーティション内の日付と販売 ID によって結果を順序付けします。

```
select salesid, dateid, sellerid, qty,
sum(qty) over (partition by sellerid
order by dateid, salesid rows unbounded preceding) as sum
from winsales
order by 2,1;

salesid |   dateid   | sellerid | qty | sum
---------+------------+----------+-----+-----
30001 | 2003-08-02 |        3 |  10 |  10
10001 | 2003-12-24 |        1 |  10 |  10
10005 | 2003-12-24 |        1 |  30 |  40
40001 | 2004-01-09 |        4 |  40 |  40
10006 | 2004-01-18 |        1 |  10 |  50
20001 | 2004-02-12 |        2 |  20 |  20
40005 | 2004-02-12 |        4 |  10 |  50
20002 | 2004-02-16 |        2 |  20 |  40
30003 | 2004-04-18 |        3 |  15 |  25
30004 | 2004-04-18 |        3 |  20 |  45
30007 | 2004-09-07 |        3 |  30 |  75
(11 rows)
```

次の例では、SELLERID および SALESID 列によって順序付けられた結果セットのすべての行に番号をつけます。

```
select salesid, sellerid, qty,
sum(1) over (order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid |  qty | rownum
--------+----------+------+--------
10001 |        1 |   10 |      1
10005 |        1 |   30 |      2
10006 |        1 |   10 |      3
20001 |        2 |   20 |      4
20002 |        2 |   20 |      5
30001 |        3 |   10 |      6
30003 |        3 |   15 |      7
30004 |        3 |   20 |      8
30007 |        3 |   30 |      9
40001 |        4 |   40 |     10
40005 |        4 |   10 |     11
```

```
(11 rows)
```

WINSALES テーブルの説明については、「ウィンドウ関数例のサンプルテーブル」を参照してください。

次の例では、結果セットの行すべてに番号をつけ、SELLERID ごとに結果をパーティション化し、パーティション内の SELLERID および SALESID 列によって結果を順序付けます。

```
select salesid, sellerid, qty,
sum(1) over (partition by sellerid
order by sellerid, salesid rows unbounded preceding) as rownum
from winsales
order by 2,1;

salesid | sellerid | qty | rownum
---------+----------+-----+--------
10001 |        1 |  10 |      1
10005 |        1 |  30 |      2
10006 |        1 |  10 |      3
20001 |        2 |  20 |      1
20002 |        2 |  20 |      2
30001 |        3 |  10 |      1
30003 |        3 |  15 |      2
30004 |        3 |  20 |      3
30007 |        3 |  30 |      4
40001 |        4 |  40 |      1
40005 |        4 |  10 |      2
(11 rows)
```

VAR_SAMP および VAR_POP ウィンドウ関数

VAR_SAMP および VAR_POP ウィンドウ関数は、数値のセットの標本分散および母集団分散を返します (整数、10 進数、または浮動小数点)。「VAR_SAMP および VAR_POP 関数」も参照してください。

VAR_SAMP および VARIANCE は同じ関数のシノニムです。

構文

```
VAR_SAMP | VARIANCE | VAR_POP
( [ ALL ] expression ) OVER
```

```
(
[ PARTITION BY expr_list ]
[ ORDER BY order_list
                        frame_clause ]
)
```

引数

expression

関数の対象となる列または式。

ALL

引数 ALL を指定すると、この関数は式から重複する値をすべて保持します。ALL がデフォルトです。DISTINCT はサポートされません。

OVER

集計関数に使用するウィンドウ句を指定します。OVER 句は、ウィンドウ集計関数を標準セット集計関数と区別します。

PARTITION BY expr_list

1 つ以上の式で関数のウィンドウを定義します。

ORDER BY order_list

各パーティション内の行をソートします。PARTITION BY が指定されていない場合、ORDER BY はテーブル全体を使用します。

frame_clause

ORDER BY 句が集計関数に使用される場合、明示的なフレーム句が必要です。フレーム句は順序付けた結果内の行のセットを含めるか除外して、関数のウィンドウの行のセットを絞り込みます。フレーム句は ROWS キーワードおよび関連する指定子で構成されます。「ウィンドウ関数の構文の概要」を参照してください。

データ型

VARIANCE 関数でサポートされる引数の型は、SMALLINT、INTEGER、BIGINT、NUMERIC、DECIMAL、REAL、および DOUBLE PRECISION です。

式のデータ型にかかわらず、VARIANCE 関数の戻り値の型は倍精度浮動小数点数です。

システム管理関数

トピック

- CHANGE_QUERY_PRIORITY
- CHANGE_SESSION_PRIORITY
- CHANGE_USER_PRIORITY
- CURRENT_SETTING
- PG_CANCEL_BACKEND
- PG_TERMINATE_BACKEND
- REBOOT_CLUSTER
- SET_CONFIG

Amazon Redshift は、いくつかのシステム管理関数をサポートします。

CHANGE_QUERY_PRIORITY

CHANGE_QUERY_PRIORITY を使用すると、スーパーユーザーは、ワークロード管理 (WLM) で実行中または待機中のクエリの優先度を変更できます。

この機能により、スーパーユーザーはシステム内のクエリの優先度をすぐに変更することができます。優先度 `CRITICAL` で実行できるクエリ、ユーザー、またはセッションは 1 つのみです。

構文

```
CHANGE_QUERY_PRIORITY(query_id, priority)
```

引数

query_id

優先度が変更されたクエリのクエリ識別子。`INTEGER` 値は必須です。

priority

クエリに割り当てる新しい優先度。引数は、値 `CRITICAL`、`HIGHEST`、`HIGH`、`NORMAL`、`LOW`、または `LOWEST` を含む文字列である必要があります。

戻り型

なし

例

STV_WLM_QUERY_STATE システムテーブルの query_priority 列を示すには、次の例を使用します。

```
SELECT query, service_class, query_priority, state
FROM stv_wlm_query_state WHERE service_class = 101;

+-------+---------------+----------------+---------+
| query | service_class | query_priority |  state  |
+-------+---------------+----------------+---------+
|  1076 |           101 | Lowest         | Running |
|  1075 |           101 | Lowest         | Running |
+-------+---------------+----------------+---------+
```

スーパーユーザーが change_query_priority 関数を実行して優先度を CRITICAL に変更した結果を示すには、次の例を使用します。

```
SELECT CHANGE_QUERY_PRIORITY(1076, 'Critical');

+--------------------------------------------------------------------------------+
|                              change_query_priority                             |
+--------------------------------------------------------------------------------+
| Succeeded to change query priority. Priority changed from Lowest to Critical. |
+--------------------------------------------------------------------------------+
```

CHANGE_SESSION_PRIORITY

CHANGE_SESSION_PRIORITY を使用すると、スーパーユーザーはシステム内のセッションの優先度をすぐに変更することができます。優先度 CRITICAL で実行できるセッション、ユーザー、またはクエリは 1 つのみです。

構文

```
CHANGE_SESSION_PRIORITY(pid, priority)
```

引数

pid

優先度が変更されたセッションのプロセス識別子。値 -1 は、現在のセッションを表します。INTEGER 値は必須です。

priority

セッションに割り当てる新しい優先度。引数は、値 CRITICAL、HIGHEST、HIGH、NORMAL、LOW、または LOWEST を含む文字列である必要があります。

戻り型

なし

例

現在のセッションを処理しているサーバープロセスのプロセス ID を返すには、次の例を使用します。

```
SELECT pg_backend_pid();

+----------------+
| pg_backend_pid |
+----------------+
|          30311 |
+----------------+
```

この例では、現在のセッションの優先度が LOWEST に変更されます。

```
SELECT CHANGE_SESSION_PRIORITY(30311, 'Lowest');

+---------------------------------------------------------------------------------------
+
|                                 change_session_priority
 |
+---------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority to lowest.
 |
```

```
+--------------------------------------------------------------------------------------
+
```

この例では、現在のセッションの優先度が HIGH に変更されます。

```
SELECT CHANGE_SESSION_PRIORITY(-1, 'High');

+--------------------------------------------------------------------------------------
+
|                                      change_session_priority
           |
+--------------------------------------------------------------------------------------
+
| Succeeded to change session priority. Changed session (pid:30311) priority from
 lowest to high. |
+--------------------------------------------------------------------------------------
+
```

セッションの優先度を変更するストアドプロシージャを作成するには、次の例を使用します。このストアドプロシージャを実行するアクセス許可は、データベースユーザー test_user に付与されます。

```
CREATE OR REPLACE PROCEDURE sp_priority_low(pid IN int, result OUT varchar)
AS $$
BEGIN
  SELECT CHANGE_SESSION_PRIORITY(pid, 'low') into result;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
GRANT EXECUTE ON PROCEDURE sp_priority_low(int) TO test_user;
```

次に、test_userという名前のデータベースユーザーがプロシージャを呼び出します。

```
CALL sp_priority_low(pg_backend_pid());

+-------------------------------------------------------+
|                         result                        |
+-------------------------------------------------------+
| Success. Change session (pid:13155) priority to low. |
+-------------------------------------------------------+
```

CHANGE_USER_PRIORITY

CHANGE_USER_PRIORITY を使用すると、スーパーユーザーは、ワークロード管理 (WLM) で実行中または待機中のユーザーが発行したすべてのクエリの優先度を変更できます。優先度 CRITICAL で実行できるユーザー、セッション、またはクエリは 1 つのみです。

構文

```
CHANGE_USER_PRIORITY(user_name, priority)
```

引数

user_name

クエリの優先度が変更されるデータベースユーザー名。

priority

user_name によって発行されたすべてのクエリに割り当てられる新しい優先度。引数は、値 CRITICAL、HIGHEST、HIGH、NORMAL、LOW、LOWEST、または RESET を含む文字列である必要があります。優先順位を CRITICAL に変更できるのはスーパーユーザーのみです。優先度を RESET に変更すると、user_nameの優先度設定が削除されます。

戻り型

なし

例

ユーザー analysis_user の優先度を LOWEST に変更するには、次の例を使用します。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'lowest');

+-------------------------------------------------------------------------------------+
|                                change_user_priority                                 |
+-------------------------------------------------------------------------------------+
| Succeeded to change user priority. Changed user (analysis_user) priority to lowest. |
+-------------------------------------------------------------------------------------+
```

優先度を LOW に変更するには、次の例を使用します。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'low');

+-------------------------------------------------------------------------------------------------------
+
|                                       change_user_priority
        |
+-------------------------------------------------------------------------------------------------------
+
| Succeeded to change user priority. Changed user (analysis_user) priority from Lowest
 to low. |
+-------------------------------------------------------------------------------------------------------
+
```

優先度をリセットするには、次の例を使用します。

```
SELECT CHANGE_USER_PRIORITY('analysis_user', 'reset');

+-------------------------------------------------------+
|                  change_user_priority                 |
+-------------------------------------------------------+
| Succeeded to reset priority for user (analysis_user). |
+-------------------------------------------------------+
```

CURRENT_SETTING

CURRENT_SETTING は、指定された構成パラメータの現在の値を返します。

この関数は、SHOWコマンドに相当します。

構文

```
current_setting('parameter')
```

次のステートメントは、指定されたセッションコンテキスト変数の現在の値を返します。

```
current_setting('variable_name')
current_setting('variable_name'[, error_if_undefined])
```

引数

parameter

表示するパラメータ値。設定パラメータの一覧については、「設定リファレンス」を参照してください。

variable_name

設定する変数の名前。これは、セッションコンテキスト変数の文字列定数でなければなりません。

error_if_undefined

(オプション) 変数名が存在しない場合の動作を指定するブール値。error_if_undefined がデフォルト値の TRUE に設定されている場合、Amazon Redshift はエラーをスローします。error_if_undefined が FALSE に設定されている場合、Amazon Redshift は NULL を返します。Amazon Redshift は、セッションコンテキスト変数に対してのみ error_if_undefined パラメータをサポートします。これは、入力が設定パラメータの場合には使用できません。

戻り型

文字列 CHAR または VARCHAR を返します。

例

query_group パラメータの現在の設定を返すには、次の例を使用します。

```
SELECT CURRENT_SETTING('query_group');

+-----------------+
| current_setting |
+-----------------+
| unset           |
+-----------------+
```

変数 app_context.user_id の現在の設定を返すには、次の例を使用します。

```
SELECT CURRENT_SETTING('app_context.user_id', FALSE);
```

PG_CANCEL_BACKEND

クエリをキャンセルします。PG_CANCEL_BACKEND は、CANCELコマンドと機能的に同じものです。ユーザー自身が現在実行しているクエリをキャンセルできます。スーパーユーザーはどのクエリでもキャンセルできます。

構文

```
pg_cancel_backend( pid )
```

引数

pid

キャンセルするクエリのプロセス ID (PID)。クエリ ID を指定してクエリをキャンセルすることはできません。クエリのプロセス ID を指定する必要があります。`INTEGER` 値は必須です。

戻り型

なし

使用に関する注意事項

複数のセッションのクエリが同じテーブルのロックを保持している場合、PG_TERMINATE_BACKEND関数を使用してセッションの 1 つを終了することができます。これにより、終了したセッションで現在実行中のトランザクションがあれば、そのすべてのロックが強制的に解放され、トランザクションがロールバックされます。PG__LOCKS カタログテーブルに対してクエリを実行し、現在保持しているロックを表示します。トランザクションブロック (BEGIN … END) 内にあるためクエリをキャンセルできない場合、PG_TERMINATE_BACKEND 関数を使用して、クエリを実行中のセッションを終了できます。

例

現在実行されているクエリをキャンセルするには、キャンセルするクエリのプロセス ID を最初に取得します。現在実行されているすべてのクエリのプロセス ID を確認するには、次のコマンドを実行します。

```
SELECT pid, TRIM(starttime) AS start,
duration, TRIM(user_name) AS user,
```

```
SUBSTRING(query,1,40) AS querytxt
FROM stv_recents
WHERE status = 'Running';

+-----+------------------------+----------+--------+-----------------------------+
| pid |       starttime        | duration |  user  |          querytxt           |
+-----+------------------------+----------+--------+-----------------------------+
| 802 | 2013-10-14 09:19:03.55 |      132 | dwuser | select venuename from venue |
| 834 | 2013-10-14 08:33:49.47 |  1250414 | dwuser | select * from listing;      |
| 964 | 2013-10-14 08:30:43.29 |   326179 | dwuser | select sellerid from sales  |
+-----+------------------------+----------+--------+-----------------------------+
```

プロセス ID 802 のクエリをキャンセルするには、次の例を使用します。

```
SELECT PG_CANCEL_BACKEND(802);
```

PG_TERMINATE_BACKEND

セッションを終了します。ユーザー自身が所有するセッションを終了できます。スーパーユーザーはどのセッションでも終了できます。

構文

```
pg_terminate_backend( pid )
```

引数

pid

終了するセッションのプロセス ID。INTEGER 値は必須です。

戻り型

なし

使用に関する注意事項

同時接続の制限に近づいている場合、PG_TERMINATE_BACKEND を使用してアイドル状態のセッションを終了し、接続を解放することができます。詳細については、Amazon Redshift における制限を参照してください。

複数のセッションのクエリが同じテーブルのロックを保持している場合、PG_TERMINATE_BACKEND を使用してセッションの 1 つを終了することができます。これにより、終了したセッションで現在実行中のトランザクションがあれば、そのすべてのロックが強制的に解放され、トランザクションがロールバックされます。PG_LOCKS カタログテーブルに対してクエリを実行し、現在保持しているロックを表示します。

クエリがトランザクションブロック (BEGIN… END) 内にない場合、CANCELコマンドまたは PG_CANCEL_BACKEND 関数を使用してクエリをキャンセルできます。

例

SVV_TRANSACTIONS テーブルをクエリし、現在のトランザクションで有効なすべてのロックを表示するには、次の例を使用します。

```
SELECT * FROM svv_transactions;

+-----------+--------+-------+------+---------------------+-----------------
+---------------------+----------+---------+
| txn_owner | txn_db |  xid  | pid  |      txn_start      |    lock_mode    |
 lockable_object_type | relation | granted |
+-----------+--------+-------+------+---------------------+-----------------
+---------------------+----------+---------+
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |    51940 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |    52000 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | AccessShareLock | relation
            |   108623 | true    |
| rsuser    | dev    | 96178 | 8585 | 2017-04-12 20:13:07 | ExclusiveLock   |
 transactionid        |          | true    |
+-----------+--------+-------+------+---------------------+-----------------
+---------------------+----------+---------+
```

ロックを保持しているセッションを終了するには、次の例を使用します。

```
SELECT PG_TERMINATE_BACKEND(8585);
```

REBOOT_CLUSTER

クラスターへの接続を閉じずに Amazon Redshift クラスターを再起動します。このコマンドを実行するには、データベースのスーパーユーザー権限を持つ必要があります。

この軽めの再起動が完了すると、Amazon Redshift クラスターはユーザーアプリケーションにエラーを返し、軽めの再起動によって中断されたトランザクションまたはクエリをユーザーアプリケーションに再送信するようにリクエストします。

構文

```
SELECT REBOOT_CLUSTER();
```

SET_CONFIG

構成パラメータを新しい値に設定します。

この関数は、SQL の SET コマンドに相当します。

構文

```
SET_CONFIG('parameter', 'new_value' , is_local)
```

次のステートメントは、セッションコンテキスト変数を新しい設定に設定します。

```
set_config('variable_name', 'new_value' , is_local)
```

引数

parameter

設定するパラメータ。

variable_name

設定する変数の名前。

new_value

パラメータの新しい値。

is_local

true の場合、パラメータ値は現在の取引にのみ適用されます。有効な値は `true` または `1`、および `false` または `0` です。

戻り型

文字列 `CHAR` または `VARCHAR` を返します。

例

現在の取引のみの `query_group` パラメータの値を `test` に設定するには、次の例を使用します。

```
SELECT SET_CONFIG('query_group', 'test', true);

+------------+
| set_config |
+------------+
| test       |
+------------+
```

セッションコンテキスト変数を設定するには、次の例を使用します。

```
SELECT SET_CONFIG('app.username', 'cuddy', FALSE);
```

システム情報関数

Amazon Redshift は、多数のシステム情報関数をサポートします。

トピック

- LAST_USER_QUERY_ID
- PG_BACKEND_PID
- PG_GET_COLS
- PG_GET_GRANTEE_BY_IAM_ROLE
- PG_GET_IAM_ROLE_BY_USER
- PG_GET_LATE_BINDING_VIEW_COLS
- PG_GET_SESSION_ROLES
- PG_LAST_COPY_COUNT
- PG_LAST_COPY_ID
- PG_LAST_UNLOAD_ID
- PG_LAST_QUERY_ID
- PG_LAST_UNLOAD_COUNT
- SLICE_NUM 関数
- USER
- ROLE_IS_MEMBER_OF
- USER_IS_MEMBER_OF
- VERSION

CURRENT_AWS_ACCOUNT

クエリを送信した Amazon Redshift クラスターが関連付けられた AWS アカウントを返します。

構文

```
current_aws_account
```

戻り型

整数を返します。

例

次のクエリは、現在のデータベースの名前を返します。

```
select user, current_aws_account;
```

```
current_user | current_account
-------------+---------------
dwuser       | 987654321

(1 row)
```

CURRENT_DATABASE

現在接続されているデータベースの名前を返します。

構文

```
current_database()
```

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次のクエリは、現在のデータベースの名前を返します。

```
select current_database();

current_database
------------------
tickit
(1 row)
```

CURRENT_NAMESPACE

現在の Amazon Redshift クラスターでは、クラスターの名前空間が返されます。Amazon Redshift クラスター名前空間は、Amazon Redshift クラスターの一意の ID です。

構文

```
current_namespace
```

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次のクエリは、現在の名前空間の名前を返します。

```
select user, current_namespace;
current_user | current_namespace
-------------+-------------------------------------
dwuser       | 86b5169f-01dc-4a6f-9fbb-e2e24359e9a8

(1 row)
```

CURRENT_SCHEMA

検索パスの先頭にあるスキーマ名を返します。このスキーマは、ターゲットスキーマが指定されずに作成されたテーブルまたはその他の名前付きオブジェクトに使用されます。

構文

> **Note**
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。

```
current_schema()
```

戻り型

CURRENT_SCHEMA returns a CHAR or VARCHAR string。

例

次のクエリは、現在のスキーマを返します。

```
select current_schema();

current_schema
----------------
public
```

```
(1 row)
```

CURRENT_SCHEMAS

現在の検索パスに含まれるスキーマの名前の配列を返します。現在の検索パスは、search_path パラメータに定義されます。

構文

> **Note**
>
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。

```
current_schemas(include_implicit)
```

引数

include_implicit

true の場合、暗黙的に組み込まれているシステムスキーマを検索パスに含めるように指定します。有効な値は、`true`および `false` です。一般的にこのパラメータは、`true`の場合、現在のスキーマに加えて、`pg_catalog`スキーマを返します。

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次の例は、現在の検索パスに含まれるスキーマの名前を返しますが、暗黙的に組み込まれているシステムスキーマは含まれません。

```
select current_schemas(false);

current_schemas
-----------------
{public}
```

```
(1 row)
```

次の例は、現在の検索パスに含まれるスキーマの名前を、暗黙的に組み込まれているシステムスキーマを含めて返します。

```
select current_schemas(true);

current_schemas
---------------------
{pg_catalog,public}
(1 row)
```

CURRENT_SESSION_ARN

現在承認されているグローバルユーザーの ARN を返します。グローバルユーザーは、Redshift アカウント、クラスター、および Serverless ワークグループ間で同じ ID を提示します。グローバルユーザーは、IAM アイデンティティセンターまたは IAM ベースのセッション認証を使用してログインします。データレイクユーザーはグローバル AWS ユーザーです。

この関数は通常、マルチダイアレクト AWS Glue ビューを使用するコンテキストで使用されます。IAM アイデンティティセンターと Redshift での ID 管理の詳細については、「Redshift を IAM アイデンティティセンターに接続してユーザーにシングルサインオンエクスペリエンスを提供する」を参照してください。マルチダイアレクト Glue ビューの詳細については、「AWS Glue データカタログでのビューの作成」を参照してください。

構文

```
current_session_arn()
```

戻り型

グローバルに認証されたユーザーの VARCHAR 文字列または null 値を返します。

使用に関する注意事項

ローカルユーザーはサポートされていないため、null レスポンスになります。

例

次のクエリは、現在のセッション ARN の名前を返します。

```
SELECT current_session_arn();

current_session_arn
---------------
arn:aws:iam::123456789012:user/user
(1 row)
```

CURRENT_USER

許可の確認に適用可能な場合、データベースの現在の「有効」なユーザーのユーザー名を返します。このユーザー名は通常、セッションユーザーと同じ名前になりますが、スーパーユーザーが変更する可能性があります。

> **Note**
>
> CURRENT_USER を呼び出すときには、末尾の括弧を使用しないでください。

構文

```
current_user
```

戻り型

CURRENT_USER は NAME データ型を返し、CHAR または VARCHAR 文字列としてキャストできます。

使用に関する注意事項

CREATE_PROCEDURE コマンドの SECURITY DEFINER オプションを使用してストアドプロシージャが作成された場合、そのストアドプロシージャ内から CURRENT_USER 関数を呼び出すと、Amazon Redshift はストアドプロシージャの所有者のユーザー名を返します。

例

次のクエリは、現在のデータベースユーザーの名前を返します。

```
select current_user;
```

```
current_user
--------------
dwuser
(1 row)
```

CURRENT_USER_ID

現在のセッションにログインした Amazon Redshift ユーザーの固有の ID を返します。

構文

```
CURRENT_USER_ID
```

戻り型

CURRENT_USER_ID 関数は整数を返します。

例

次の例は、このセッションのユーザー名および現在のユーザー ID を返します。

```
select user, current_user_id;

 current_user | current_user_id
--------------+-----------------
   dwuser     |               1
(1 row)
```

DEFAULT_IAM_ROLE

Amazon Redshift クラスターに現在関連付けられている、デフォルトの IAM ロールを返します。関連付けられたデフォルトの IAM ロールがない場合、この関数は none を返します。

構文

```
select default_iam_role();
```

戻り型

VARCHAR 型の文字列を返します。

例

次の例では、指定された Amazon Redshift クラスターに現在関連付けられている、デフォルトの IAM ロールを返します。

```
select default_iam_role();
              default_iam_role
-----------------------------------------------
 arn:aws:iam::123456789012:role/myRedshiftRole
(1 row)
```

GET_MOUNTED_ROLE

マルチダイアレクト AWS Glue ビューの一部として呼び出されると、Lake Formation スキーマまたはデータベースのマウントに使用される IAM ロールを返すことができます。マルチダイアレクトとは、SQL が Amazon EMR や Redshift などの複数のクエリエンジンでサポートされていることを意味します。マルチダイアレクト Glue ビューの詳細については、「AWS Glue データカタログでのビューの作成」を参照してください。

構文

```
get_mounted_role()
```

戻り型

VARCHAR 文字列または null 値を返します。

使用に関する注意事項

この関数は、外部の Lake Formation ビュー以外のユースケースに対して null を返します。

例

次のクエリは、Lake Formation リソースをマウントするための ID を返します。

```
CREATE EXTERNAL PROTECTED VIEW external_schema.remote_view AS
SELECT mycol, get_mounted_role() FROM external_schema.remote_table;

mycol | get_mounted_role
----------------------------
1       arn:aws:iam::123456789012:role/salesrole
```

```
(1 row)
```

HAS_ASSUMEROLE_PRIVILEGE

指定されたユーザーが、指定されたコマンドの実行権限がある特定の IAM ロールを持っている場合、Boolean 型の `true` (t) を返します。指定されたコマンドを実行する権限が付与された、特定の IAM ロールがユーザーにない場合、この関数では `false` (f) が返されます。権限の詳細については、「GRANT」を参照してください。

構文

```
has_assumerole_privilege( [ user, ] iam_role_arn, cmd_type)
```

引数

user

IAM ロールに対する権限をチェックするユーザーの名前。デフォルトでは、現在のユーザーが検査されます。スーパーユーザーとユーザーはこの機能を使用できます。ただし、ユーザーは自分の権限しか表示できません。

iam_role_arn

コマンド権限が付与されている IAM ロール。

cmd_type

アクセスが許可されているコマンド。有効な値は以下のとおりです。

- COPY
- UNLOAD
- EXTERNAL FUNCTION
- モデルを作成する

戻り型

BOOLEAN

例

次のクエリは、ユーザー `reg_user1` が COPY コマンドを実行するための `Redshift-S3-Read` ロールの権限を持っていることを確認します。

```
select has_assumerole_privilege('reg_user1', 'arn:aws:iam::123456789012:role/Redshift-
S3-Read', 'copy');
```

```
has_assumerole_privilege
------------------------
true
(1 row)
```

HAS_DATABASE_PRIVILEGE

ユーザーが、指定されたデータベースに対して指定された権限を持っている場合は、`true`を返します。権限の詳細については、「GRANT」を参照してください。

構文

> **Note**
>
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。

```
has_database_privilege( [ user, ] database, privilege)
```

引数

user

データベースに対する権限を検査するユーザーの名前。デフォルトでは、現在のユーザーが検査されます。

database

権限に関連付けられているデータベース。

privilege

検査する権限。有効な値は以下のとおりです。

- CREATE
- TEMPORARY
- TEMP

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次のクエリでは、GUEST ユーザーが TICKIT データベースに対して TEMP 権限を持っていることを確認します。

```
select has_database_privilege('guest', 'tickit', 'temp');

has_database_privilege
------------------------
true
(1 row)
```

HAS_SCHEMA_PRIVILEGE

ユーザーが、指定されたスキーマに対して指定された権限を持っている場合は、`true`を返します。権限の詳細については、「GRANT」を参照してください。

構文

> **Note**
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。

```
has_schema_privilege( [ user, ] schema, privilege)
```

引数

user

スキーマに対する権限を検査するユーザーの名前。デフォルトでは、現在のユーザーが検査されます。

schema

権限に関連付けられているスキーマ。

privilege

検査する権限。有効な値は以下のとおりです。

- CREATE
- USAGE

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次のクエリでは、GUEST ユーザーが PUBLIC スキーマに対して CREATE 権限を持っていることを確認します。

```
select has_schema_privilege('guest', 'public', 'create');

has_schema_privilege
----------------------
true
(1 row)
```

HAS_TABLE_PRIVILEGE

ユーザーが、指定されたテーブルに対して指定された権限を持っている場合、`true` を返し、それ以外の場合は `false` を返します。

構文

> **Note**
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。権限の詳細については、「GRANT」を参照してください。

```
has_table_privilege( [ user, ] table, privilege)
```

引数

user

テーブルに対する権限を検査するユーザーの名前。デフォルトでは、現在のユーザーが検査されます。

table

権限に関連付けられているテーブル。

privilege

検査する権限。有効な値は以下のとおりです。

- SELECT
- INSERT
- UPDATE
- DELETE
- DROP
- REFERENCES

戻り型

BOOLEAN

例

次のクエリでは、GUEST ユーザーが LISTING テーブルに対して SELECT 権限を持っていないことを検出します。

```
select has_table_privilege('guest', 'listing', 'select');

has_table_privilege
---------------------
false
```

次のクエリは、pg_tables と pg_user カタログテーブルからの出力を使用して、選択、挿入、更新、削除などのテーブル権限を一覧表示します。これはサンプルのみです。データベースからスキーマ名とテーブル名を指定する必要がある場合があります。詳細については、「カタログテーブルへのクエリの実行」を参照してください。

```
SELECT
     tablename
     ,usename
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'select') AS sel
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'insert') AS ins
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'update') AS upd
     ,HAS_TABLE_PRIVILEGE(users.usename, tablename, 'delete') AS del
FROM
(SELECT * from pg_tables
WHERE schemaname = 'public' and tablename in ('event','listing')) as tables
,(SELECT * FROM pg_user) AS users;

tablename | usename    |  sel   |  ins  |  upd  | del
----------+------------+--------+-------+-------+-------
event     |  john      |  true  | true  | true  | true
event     |  sally     |  false | false | false | false
event     |  elsa      |  false | false | false | false
listing   |  john      |  true  | true  | true  | true
listing   |  sally     |  false | false | false | false
listing   |  elsa      |  false | false | false | false
```

前のクエリにはクロス結合も含まれています。詳細については、「JOIN 句の例」を参照してください。public スキーマにないテーブルをクエリするには、WHERE 句から schemaname 条件を削除し、クエリの前に次の例を使用します。

```
SET SEARCH_PATH to 'schema_name';
```

LAST_USER_QUERY_ID

現在のセッションで最近完了した、ユーザークエリのクエリ ID を返します。現在のセッションで実行されたクエリが存在しない場合、last_user_query_id は -1 を返します。この関数は、リーダーノードでのみ実行されるクエリについては、クエリ ID を返しません。詳細については、「リーダーノード専用関数」を参照してください。

構文

```
last_user_query_id()
```

戻り型

整数を返します。

例

次のクエリは、現在のセッションで完了されたユーザーが実行した最新クエリの ID を返します。

```
select last_user_query_id();
```

結果は、以下のとおりです。

```
last_user_query_id
-----------------------
    5437
(1 row)
```

次のクエリは、現在のセッションでユーザーが実行して最近完了されたクエリの ID とテキストを返します。

```
select query_id, query_text from sys_query_history where query_id =
 last_user_query_id();
```

結果は、以下のとおりです。

```
 query_id, query_text
---------
+----------------------------------------------------------------------------------------------
 5556975 | select last_user_query_id() limit 100 --RequestID=<unique request ID>;
 TraceID=<unique trace ID>
```

PG_BACKEND_PID

現在のセッションを処理しているサーバープロセスのプロセス ID (PID) を返します。

> **Note**
> PID はグローバルで一意ではありません。時間が経つと再利用できます。

構文

```
pg_backend_pid()
```

戻り型

整数を返します。

例

ログテーブルに PG_BACKEND_PID を関連付けることで、現在のセッションの情報を取得できます。例えば、以下のクエリは、現在のセッションで完了されたクエリのクエリ ID とクエリテキストの一部を返します。

```
select query, substring(text,1,40)
from stl_querytext
where pid =  PG_BACKEND_PID()
order by query desc;

 query |                  substring
-------+------------------------------------------
 14831 | select query, substring(text,1,40) from
 14827 | select query, substring(path,0,80) as pa
 14826 | copy category from 's3://dw-tickit/manif
 14825 | Count rows in target table
 14824 | unload ('select * from category') to 's3
(5 rows)
```

次のログテーブルの pid 列に PG_BACKEND_PID を関連付けることができます (例外についてはかっこで注記を付けています)。

- STL_CONNECTION_LOG
- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_QUERYTEXT
- STL_SESSIONS (process)
- STL_TR_CONFLICT
- STL_UTILITYTEXT
- STV_ACTIVE_CURSORS
- STV_INFLIGHT
- STV_LOCKS (lock_owner_pid)

- STV_RECENTS (process_id)

PG_GET_COLS

テーブルまたはビュー定義の列メタデータを返します。

構文

```
pg_get_cols('name')
```

引数

name

Amazon Redshift テーブルまたはビューの名前。詳細については、「名前と識別子」を参照してください。

戻り型

VARCHAR

使用に関する注意事項

PG_GET_COLS 関数は、テーブルまたはビュー定義の列ごとに、行を 1 つ返します。行には、スキーマ名、関係名、列名、データ型、行番号のカンマ区切りリストが含まれています。SQL の結果の書式設定は、使用する SQL クライアントによって異なります。

例

次の例は、接続されたデータベース dev のユーザーによって作成されたスキーマ public の SALES_VW という名前のビューとスキーマ mytickit1 の sales という名前のテーブルの結果を返します。

次の例は、SALES_VW というビューの列メタデータを返します。

```
select pg_get_cols('sales_vw');

pg_get_cols
-----------------------------------------------------------
(public,sales_vw,salesid,integer,1)
(public,sales_vw,listid,integer,2)
```

```
(public,sales_vw,sellerid,integer,3)
(public,sales_vw,buyerid,integer,4)
(public,sales_vw,eventid,integer,5)
(public,sales_vw,dateid,smallint,6)
(public,sales_vw,qtysold,smallint,7)
(public,sales_vw,pricepaid,"numeric(8,2)",8)
(public,sales_vw,commission,"numeric(8,2)",9)
(public,sales_vw,saletime,"timestamp without time zone",10)
```

次の例は、SALES_VW というビューの列メタデータをテーブル形式で返します。

```
select * from pg_get_cols('sales_vw')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                      | col_num
------------+-----------+------------+-------------------------------+--------
public      | sales_vw  | salesid    | integer                       |       1
public      | sales_vw  | listid     | integer                       |       2
public      | sales_vw  | sellerid   | integer                       |       3
public      | sales_vw  | buyerid    | integer                       |       4
public      | sales_vw  | eventid    | integer                       |       5
public      | sales_vw  | dateid     | smallint                      |       6
public      | sales_vw  | qtysold    | smallint                      |       7
public      | sales_vw  | pricepaid  | numeric(8,2)                  |       8
public      | sales_vw  | commission | numeric(8,2)                  |       9
public      | sales_vw  | saletime   | timestamp without time zone |      10
```

次の例は、スキーマ mytickit1 の SALES というテーブルの列メタデータをテーブル形式で返します。

```
select * from pg_get_cols('"mytickit1"."sales"')
cols(view_schema name, view_name name, col_name name, col_type varchar, col_num int);

view_schema | view_name | col_name   | col_type                      | col_num
------------+-----------+------------+-------------------------------+--------
mytickit1   | sales     | salesid    | integer                       |       1
mytickit1   | sales     | listid     | integer                       |       2
mytickit1   | sales     | sellerid   | integer                       |       3
mytickit1   | sales     | buyerid    | integer                       |       4
mytickit1   | sales     | eventid    | integer                       |       5
mytickit1   | sales     | dateid     | smallint                      |       6
mytickit1   | sales     | qtysold    | smallint                      |       7
mytickit1   | sales     | pricepaid  | numeric(8,2)                  |       8
```

```
mytickit1   | sales     | commission | numeric(8,2)                |       9
mytickit1   | sales     | saletime   | timestamp without time zone |      10
```

PG_GET_GRANTEE_BY_IAM_ROLE

指定された IAM ロールを付与されたすべてのユーザーとグループを返します。

構文

```
pg_get_grantee_by_iam_role('iam_role_arn')
```

引数

iam_role_arn

このロールを付与されたユーザーとグループを返す IAM ロール。

戻り型

VARCHAR

使用に関する注意事項

PG_GET_GRANTEE_BY_IAM_ROLE 関数は、ユーザーまたはグループごとに、行を 1 つ返します。各行には、被付与者名、被付与者のタイプ、および付与された権限が含まれます。被付与者タイプに指定できる値は、パブリックの場合は p、ユーザーの場合は u、グループの場合は g です。

この機能を使用するには、スーパーユーザーである必要があります。

例

次の例は、IAM ロール `Redshift-S3-Write` が `group1` と `reg_user1` に付与されていることを示しています。`group_1` のユーザーは COPY オペレーションに対してのみロールを指定でき、ユーザー `reg_user1` は UNLOAD オペレーションを実行するためにのみロールを指定できます。

```
select pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write');
```

```
  pg_get_grantee_by_iam_role
-----------------------------
 (group_1,g,COPY)
```

```
 (reg_user1,u,UNLOAD)
```

次の PG_GET_GRANTEE_BY_IAM_ROLE 関数の例では、結果をテーブルとしてフォーマットします。

```
select grantee, grantee_type, cmd_type FROM
 pg_get_grantee_by_iam_role('arn:aws:iam::123456789012:role/Redshift-S3-Write')
 res_grantee(grantee text, grantee_type text, cmd_type text) ORDER BY 1,2,3;
```

```
  grantee  | grantee_type | cmd_type
-----------+--------------+----------
 group_1   | g            | COPY
 reg_user1 | u            | UNLOAD
```

PG_GET_IAM_ROLE_BY_USER

ユーザーに付与されたすべての IAM ロールとコマンド権限を返します。

構文

```
pg_get_iam_role_by_user('name')
```

引数

name

IAM ロールを返すユーザーの名前。

戻り型

VARCHAR

使用に関する注意事項

PG_GET_IAM_ROLE_BY_USER 関数は、ロールとコマンド権限のセットごとに、行を 1 つ返します。行には、ユーザー名、IAM ロール、コマンドを含むコンマ区切りのリストが含まれています。

結果の値 `default` は、表示されたコマンドを実行するためにユーザーが使用できる任意のロールを指定できることを示します。

この機能を使用するには、スーパーユーザーである必要があります。

例

次の例では、ユーザー reg_user1 が COPY オペレーションを実行するために使用できる IAM ロールを指定できます。ユーザーは、UNLOAD オペレーションの Redshift-S3-Write ロールを指定することもできます。

```
select pg_get_iam_role_by_user('reg_user1');
```

```
                              pg_get_iam_role_by_user
---------------------------------------------------------------------------------
 (reg_user1,default,COPY)
 (reg_user1,arn:aws:iam::123456789012:role/Redshift-S3-Write,COPY|UNLOAD)
```

次の PG_GET_IAM_ROLE_BY_USER 関数の例では、結果をテーブルとしてフォーマットします。

```
select username, iam_role, cmd FROM pg_get_iam_role_by_user('reg_user1')
 res_iam_role(username text, iam_role text, cmd text);
```

```
 username  |                    iam_role                     | cmd
-----------+-------------------------------------------------+------
 reg_user1 | default                                         | None
 reg_user1 | arn:aws:iam::123456789012:role/Redshift-S3-Read | COPY
```

PG_GET_LATE_BINDING_VIEW_COLS

データベースにあるすべての遅延バインドビューの列メタデータを返します。詳細については、「遅延バインドビュー」を参照してください。

構文

```
pg_get_late_binding_view_cols()
```

戻り型

VARCHAR

使用に関する注意事項

PG_GET_LATE_BINDING_VIEW_COLS 関数は、遅延バインドビューの列ごとに、行を 1 つ返します。行には、スキーマ名、関係名、列名、データ型、行番号のカンマ区切りリストが含まれています。

例

以下の例は、すべての遅延バインドビューの列メタデータを返します。

```
select pg_get_late_binding_view_cols();

pg_get_late_binding_view_cols
------------------------------------------------------------
(public,myevent,eventname,"character varying(200)",1)
(public,sales_lbv,salesid,integer,1)
(public,sales_lbv,listid,integer,2)
(public,sales_lbv,sellerid,integer,3)
(public,sales_lbv,buyerid,integer,4)
(public,sales_lbv,eventid,integer,5)
(public,sales_lbv,dateid,smallint,6)
(public,sales_lbv,qtysold,smallint,7)
(public,sales_lbv,pricepaid,"numeric(8,2)",8)
(public,sales_lbv,commission,"numeric(8,2)",9)
(public,sales_lbv,saletime,"timestamp without time zone",10)
(public,event_lbv,eventid,integer,1)
(public,event_lbv,venueid,smallint,2)
(public,event_lbv,catid,smallint,3)
(public,event_lbv,dateid,smallint,4)
(public,event_lbv,eventname,"character varying(200)",5)
(public,event_lbv,starttime,"timestamp without time zone",6)
```

以下の例は、すべての遅延バインドビューの列メタデータをテーブル形式で返します。

```
select * from pg_get_late_binding_view_cols() cols(view_schema name, view_name name,
 col_name name, col_type varchar, col_num int);
view_schema | view_name | col_name   | col_type                    | col_num
------------+-----------+------------+-----------------------------+--------
public      | sales_lbv | salesid    | integer                     |       1
public      | sales_lbv | listid     | integer                     |       2
public      | sales_lbv | sellerid   | integer                     |       3
public      | sales_lbv | buyerid    | integer                     |       4
public      | sales_lbv | eventid    | integer                     |       5
public      | sales_lbv | dateid     | smallint                    |       6
public      | sales_lbv | qtysold    | smallint                    |       7
public      | sales_lbv | pricepaid  | numeric(8,2)                |       8
public      | sales_lbv | commission | numeric(8,2)                |       9
public      | sales_lbv | saletime   | timestamp without time zone |      10
public      | event_lbv | eventid    | integer                     |       1
```

```
public       | event_lbv | venueid    | smallint                    |       2
public       | event_lbv | catid      | smallint                    |       3
public       | event_lbv | dateid     | smallint                    |       4
public       | event_lbv | eventname  | character varying(200)      |       5
public       | event_lbv | starttime  | timestamp without time zone |       6
```

PG_GET_SESSION_ROLES

現在ログインしているユーザーのセッションロールを返します。ユーザーのセッションロールは、ログインしたユーザーの ID プロバイダー (IdP) によって定義されるグループです。例えば、Microsoft Azure Active Directory (Azure AD) のような ID プロバイダー (IdP) は、ユーザーの ID を検証し、そのユーザーが所属する外部グループをユーザーログインプロセス中に提供します。これらの外部グループは Amazon Redshift ロールに変換され、現在のセッション中に使用できます。これらのロールはセッションロールと呼ばれます。管理者は、他の Amazon Redshift ロールと同様の権限をセッションロールに付与できます。ロールの使用の詳細については、「ロールベースのアクセスコントロール (RBAC)」を参照してください。ID プロバイダー (IdP) による ID 管理の詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift 用のネイティブ ID プロバイダー (IdP) フェデレーション」を参照してください。

Amazon Redshift カタログで定義されているロールを表示するには、システムビュー SVV_ROLES をクエリします。

構文

```
pg_get_session_roles()
```

戻り型

2 つの値で構成される行のセット。最初の値は 2 つの部分からなり、`idp-namespace:role-name` を含むコロン (:) で区切られます。`idp-namespace` は、ID プロバイダー (IdP) の名前空間です。`role-name` は、ID プロバイダー (IdP) の外部グループの名前です。2 番目の値には、ロール識別子である `role-id` が含まれます。

使用に関する注意事項

`PG_GET_SESSION_ROLES` 関数は、返されたセッションロールごとに 1 行を返します。

例

次の例では、Azure Active Directory IdP からロールごとに 1 つの行が返されます。返された列は、列 name および roleid と共に sess_roles にキャストされます。各 name は、Azure Active Directory 名前空間と Azure Active Directory 内のグループ名で構成されます。

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                    roleid
--------------------------------
my_aad:test_group_1    106204
my_aad:test_group_2    106205
my_aad:test_group_3    106206
my_aad:test_group_4    106207
my_aad:test_group_5    106208
```

次の例では、現在ログインしている IAM ユーザーがメンバーになっている IAM グループごとに 1 つの行が返されます。返された列は、列 name および roleid と共に sess_roles にキャストされます。各 name は、IAM 名前空間と IAM グループ名で構成されます。

```
SELECT * FROM pg_get_session_roles() AS sess_roles(name name, roleid integer);

name                    roleid
--------------------------------
IAM:myGroup            110332
```

PG_LAST_COPY_COUNT

現在のセッションで最後に実行された COPY コマンドでロードされた行数を返します。PG_LAST_COPY_COUNT は、ロードが失敗した場合でも、ロードプロセスを開始した最後の COPY 操作のクエリ ID である最後の COPY ID で更新されます。COPY コマンドがロードプロセスを開始すると、クエリ ID と COPY ID が更新されます。

構文エラーまたは不十分な権限のために COPY が失敗した場合、COPY ID は更新されず、PG_LAST_COPY_COUNT は前の COPY のカウントを返します。現在のセッションで COPY コマンドが実行されなかった場合、または最後の COPY がロード中に失敗した場合、PG_LAST_COPY_COUNT は 0 を返します。詳細については、「PG_LAST_COPY_ID」を参照してください。

構文

```
pg_last_copy_count()
```

戻り型

BIGINT を返します。

例

次のクエリは、現在のセッションで最後に実行された COPY コマンドによってロードされた行数を返します。

```
select pg_last_copy_count();

pg_last_copy_count
--------------------
              192497
(1 row)
```

PG_LAST_COPY_ID

現在のセッションで最近完了した、COPY コマンドのクエリ ID を返します。現在のセッションで実行された COPY コマンドが存在しない場合、PG_LAST_COPY_ID は -1 を返します。

PG_LAST_COPY_ID の値は、COPY コマンドがロードプロセスを開始するときに更新されます。ロードデータが無効なために COPY 操作が失敗した場合、COPY ID は更新されます。このため、STL_LOAD_ERRORS テーブルに対するクエリを実行する時に、PG_LAST_COPY_ID を使用できます。COPY トランザクションがロールバックされる場合、COPY ID は更新されません。

構文エラー、アクセスエラー、無効な認証情報、または不十分な権限など、ロードプロセスが開始する前に発生するエラーにより COPY コマンドが失敗すると、COPY ID は更新されません。接続が成功した後、データをロードする前に始まる圧縮を分析するステップの間に、COPY が失敗すると、COPY ID は更新されません。

構文

```
pg_last_copy_id()
```

戻り型

整数を返します。

例

次のクエリは、現在のセッションで最後に実行された COPY コマンドのクエリ ID を返します。

```
select pg_last_copy_id();

pg_last_copy_id
---------------
          5437
(1 row)
```

次のクエリは、STL_LOAD_ERRORS を STL_LOADERROR_DETAIL に結合して、現在のセッションで最後に実行されたロード中に発生したエラーの詳細を表示します。

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |                    err_reason
-------+-------------------+------+----------
+-------------------------------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or unsupported
 UTF8 code
    558| allusers_pipe.txt |  251 | Walter   | String contains invalid or unsupported
 UTF8 code
```

PG_LAST_UNLOAD_ID

現在のセッションで直近に完了された UNLOAD コマンドのクエリ ID を返します。現在のセッションで実行された UNLOAD コマンドが存在しない場合、PG_LAST_UNLOAD_ID は -1 を返します。

PG_LAST_UNLOAD_ID の値は、UNLOAD コマンドがロードプロセスを開始するときに更新されます。ロードデータが無効であるために UNLOAD が失敗した場合は、UNLOAD ID が更新されるた

め、更新された UNLOAD ID を以後の調査に使用できます。UNLOAD トランザクションがロールバックされた場合、UNLOAD ID は更新されません。

ロードプロセスの開始前に発生した構文エラー、アクセスエラー、無効な認証情報、不十分な権限などのエラーにより UNLOAD コマンドが失敗した場合、UNLOAD ID は更新されません。

構文

```
PG_LAST_UNLOAD_ID()
```

戻り型

整数を返します。

例

次のクエリは、現在のセッションで直近に実行された UNLOAD コマンドのクエリ ID を返します。

```
select PG_LAST_UNLOAD_ID();

PG_LAST_UNLOAD_ID
---------------
          5437
(1 row)
```

PG_LAST_QUERY_ID

現在のセッションで最近完了した、クエリのクエリ ID を返します。現在のセッションで実行されたクエリが存在しない場合、PG_LAST_QUERY_ID は -1 を返します。PG_LAST_QUERY_ID は、リーダーノードでのみ実行されるクエリについては、クエリ ID を返しません。詳細については、「リーダーノード専用関数」を参照してください。

構文

```
pg_last_query_id()
```

戻り型

整数を返します。

例

次のクエリは、現在のセッションで最後に完了されたクエリの ID を返します。

```
select pg_last_query_id();
```

結果は、以下の通りです。

```
pg_last_query_id
----------------
           5437
(1 row)
```

次のクエリは、現在のセッションで最後に完了されたクエリの ID とテキストを返します。

```
select query, trim(querytxt) as sqlquery
from stl_query
where query = pg_last_query_id();
```

結果は、以下のとおりです。

```
query | sqlquery
------+--------------------------------------------------
 5437 | select name, loadtime from stl_file_scan where loadtime > 1000000;
(1 rows)
```

PG_LAST_UNLOAD_COUNT

現在のセッションで最後に完了された UNLOAD コマンドによりアンロードされた行数を返します。PG_LAST_UNLOAD_COUNT は、操作に失敗した場合でも、最後の UNLOAD のクエリ ID で更新されます。UNLOAD が完了するとクエリ ID が更新されます。構文エラーまたは不十分な権限のために UNLOAD が失敗した場合、PG_LAST_UNLOAD_COUNT は前の UNLOAD のカウントを返します。現在のセッションで UNLOAD コマンドが実行されなかった場合や、最後のアンロードのオペレーション中に UNLOAD が失敗した場合、PG_LAST_UNLOAD_COUNT は 0 を返します。

構文

```
pg_last_unload_count()
```

戻り型

BIGINT を返します。

例

次のクエリは、現在のセッションで最後に実行された UNLOAD コマンドによってアンロードされた行数を返します。

```
select pg_last_unload_count();

pg_last_unload_count
--------------------
             192497
(1 row)
```

SLICE_NUM 関数

行のデータが存在するクラスター内のスライス番号に対応する整数を返します。SLICE_NUM にはパラメータはありません。

構文

```
SLICE_NUM()
```

戻り型

SLICE_NUM 関数は整数を返します。

例

次の例では、EVENTS テーブル内の最初の 10 個の EVENT 行のデータを含むスライスを示します。

```
select distinct eventid, slice_num() from event order by eventid limit 10;

 eventid | slice_num
---------+-----------
       1 |         1
       2 |         2
       3 |         3
       4 |         0
       5 |         1
       6 |         2
       7 |         3
       8 |         0
       9 |         1
```

```
      10 |          2
(10 rows)
```

次の例では、コード (10000) を返すことで、FROM ステートメントを含まないクエリがリーダーノードで実行されていることを示しています。

```
select slice_num();
slice_num
-----------
10000
(1 row)
```

USER

CURRENT_USER のシノニム。「CURRENT_USER」を参照してください。

ROLE_IS_MEMBER_OF

ロールが別のロールのメンバーである場合は true を返します。スーパーユーザーは、すべてのロールのメンバーシップを確認できます。ACCESS SYSTEM TABLE のアクセス許可を持つ通常のユーザーは、すべてのユーザーのメンバーシップを確認できます。そうでない場合、通常のユーザーが確認できるのはアクセス権のあるロールのみです。提供されたロールが存在しない場合、または現在のユーザーがロールへのアクセス権を持っていない場合、Amazon Redshift はエラーになります。

構文

```
role_is_member_of( role_name,  granted_role_name)
```

引数

role_name

ロールの名前。

granted_role_name

付与されたロールの名前。

戻り型

BOOLEAN を返します。

例

次のクエリでは、ロールが role1 または role2 のメンバーではないことを確認します。

```
SELECT role_is_member_of('role1', 'role2');

 role_is_member_of
-------------------
             False
```

USER_IS_MEMBER_OF

ユーザーがロールまたはグループのメンバーである場合は true を返します。スーパーユーザーは、すべてのユーザーのメンバーシップを確認できます。sys:secadmin または sys:superuser ロールのメンバーである通常のユーザーは、すべてのユーザーのメンバーシップを確認できます。そうでない場合、通常のユーザーが確認できるのは自分自身のメンバーシップのみです。提供されたアイデンティティが存在しない場合、または現在のユーザーがロールへのアクセス権を持っていない場合、Amazon Redshift はエラーを送信します。

構文

```
user_is_member_of( user_name,  role_name | group_name)
```

引数

user_name

　ユーザーの名前。

role_name

　ロールの名前。

group_name

　グループの名前。

戻り型

BOOLEAN を返します。

例

次のクエリでは、ユーザーが role1 のメンバーでないことを確認します。

```
SELECT user_is_member_of('reguser', 'role1');

 user_is_member_of
-------------------
           False
```

VERSION

VERSION 関数は、現在インストールされているリリースに関する詳細を返し、Amazon Redshift の特定のバージョンに関する情報が最後に付加されます。

> **Note**
> これはリーダーノード関数です。この関数は、ユーザー作成テーブル、STL または STV システムテーブル、SVV または SVL システムビューを参照する場合、エラーを返します。

構文

```
VERSION()
```

戻り型

CHAR 型または VARCHAR 型の文字列を返します。

例

次の例は、現在のクラスターのクラスターバージョン情報を示しています。

```
select version();
```

```
 version
------------------------------------------------------------------------------------------------

 PostgreSQL 8.0.2 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 3.4.2 20041017 (Red
 Hat 3.4.2-6.fc3), Redshift 1.0.12103
```

ここで `1.0.12103` はクラスターのバージョン番号です。

> **Note**
>
> クラスターを最新のクラスターバージョンに強制的に更新するには、メンテナンスウィンドウを調整します。

予約語

以下は、Amazon Redshift の予約語の一覧です。予約語は、区切られた識別子 (二重引用符) とともに使用できます。

> **Note**
>
> START と CONNECT は予約語ではありませんが、ランタイムに失敗しないように、クエリのテーブルエイリアスとして START と CONNECT を使用する場合は、区切られた識別子または AS を使用してください。

詳細については、「名前と識別子」を参照してください。

```
AES128
AES256
ALL
ALLOWOVERWRITE
ANALYSE
ANALYZE
AND
ANY
ARRAY
AS
ASC
AUTHORIZATION
AZ64
BACKUP
BETWEEN
BINARY
BLANKSASNULL
BOTH
BYTEDICT
```

```
BZIP2
CASE
CAST
CHECK
COLLATE
COLUMN
CONSTRAINT
CREATE
CREDENTIALS
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURRENT_USER_ID
DEFAULT
DEFERRABLE
DEFLATE
DEFRAG
DELTA
DELTA32K
DESC
DISABLE
DISTINCT
DO
ELSE
EMPTYASNULL
ENABLE
ENCODE
ENCRYPT
ENCRYPTION
END
EXCEPT
EXPLICIT
FALSE
FOR
FOREIGN
FREEZE
FROM
FULL
GLOBALDICT256
GLOBALDICT64K
GRANT
GROUP
```

```
GZIP
HAVING
IDENTITY
IGNORE
ILIKE
IN
INITIALLY
INNER
INTERSECT
INTERVAL
INTO
IS
ISNULL
JOIN
LEADING
LEFT
LIKE
LIMIT
LOCALTIME
LOCALTIMESTAMP
LUN
LUNS
LZO
LZOP
MINUS
MOSTLY16
MOSTLY32
MOSTLY8
NATURAL
NEW
NOT
NOTNULL
NULL
NULLS
OFF
OFFLINE
OFFSET
OID
OLD
ON
ONLY
OPEN
OR
ORDER
```

```
OUTER
OVERLAPS
PARALLEL
PARTITION
PERCENT
PERMISSIONS
PIVOT
PLACING
PRIMARY
RAW
READRATIO
RECOVER
REFERENCES
REJECTLOG
RESORT
RESPECT
RESTORE
RIGHT
SELECT
SESSION_USER
SIMILAR
SNAPSHOT
SOME
SYSDATE
SYSTEM
TABLE
TAG
TDES
TEXT255
TEXT32K
THEN
TIMESTAMP
TO
TOP
TRAILING
TRUE
TRUNCATECOLUMNS
UNION
UNIQUE
UNNEST
UNPIVOT
USER
USING
VERBOSE
```

```
WALLET
WHEN
WHERE
WITH
WITHOUT
```

システムテーブルとビューのリファレンス

Amazon Redshift には、システムの動作に関する情報を含む多くのシステムテーブルとビューがあります。これらのシステムテーブルとビューには、その他のデータベーステーブルと同じ方法でクエリを実行できます。このセクションでは、いくつかのサンプルシステムテーブルクエリを示し、以下について説明します。

- 異なるタイプのシステムテーブルとビューが生成される方法
- これらのテーブルから取得できる情報のタイプ
- Amazon Redshift システムテーブルをカタログテーブルに結合する方法
- システムテーブルのログファイルの増大を管理する方法

一部のシステムテーブルは、診断目的のため AWS スタッフのみが使用できます。以下のセクションでは、システム管理者またはその他のデータベースユーザーが有益な情報を取得するためにクエリを実行できるシステムテーブルについて説明します。

> **Note**
>
> システムテーブルは自動または手動クラスターバックアップ（スナップショット）には含まれません。STL システムビューは 7 日間のログ履歴を保持します。ログの保持にお客様によるアクションは不要ですが、ログデータを 7 日間以上保持する場合は、ログを定期的に他のテーブルにコピーするか、Amazon S3 にアンロードする必要があります。

トピック

- システムモニタリング (プロビジョニングのみ)
- システムカタログテーブル

システムテーブルとビューのタイプ

システムテーブルとビューには、以下の複数の種類があります。

- SVV ビューには、一時的な STV テーブルへの参照を含むデータベースオブジェクトに関する情報が含まれます。
- SYS ビューは、プロビジョニングされたクラスターとサーバーレスワークグループのクエリとワークロードの使用状況をモニタリングするために使用します。
- STL ビューは、システムの履歴を提供するためにディスクに保持されたログから生成されます。
- STV テーブルとは、現在のシステムデータのスナップショットを含む仮想システムテーブルです これらのテーブルは一時的なメモリ内データに基づいていて、ディスクベースのログまたは通常のテーブルには保持されません。
- SVCS ビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。
- SVL ビューは、メインクラスターのクエリに関する詳細を表示します。

システムテーブルとビューは、通常のテーブルと同じ整合性モデルを使用しません。システムテーブルとビュー (特に STV テーブルと SVV ビュー) にクエリを実行するときは、これについて注意するのことが重要です。たとえば、通常の t1 テーブルと c1 列がある場合、次のクエリが行を返さないことが予測されます。

```
select * from t1
where c1 > (select max(c1) from t1)
```

ただし、システムテーブルに対する次のクエリは、行を返す場合があります。

```
select * from stv_exec_state
where currenttime > (select max(currenttime) from stv_exec_state)
```

このクエリが行を返すことがある理由は、currenttime が一時的で、クエリの 2 つの参照が評価時に同じ値を返さない可能性があるためです。

一方、次のクエリは行を返さないと予想されます。

```
select * from stv_exec_state
where currenttime = (select max(currenttime) from stv_exec_state)
```

システムテーブルとビューのデータの可視性

システムテーブルとビューには、データについて 2 つのクラスの可視性 (ユーザーが表示可能、スーパーユーザーが表示可能) があります。

スーパーユーザー特権を持っているユーザーのみが、スーパーユーザーが表示可能なカテゴリのテーブルのデータを表示できます。通常のユーザーは、ユーザーが表示可能なテーブルのデータを表示できます。スーパーユーザーが表示できるテーブルに通常のユーザーがアクセスできるようにするには、テーブルに対する SELECT 権限を通常のユーザーに付与します。詳細については、「GRANT」を参照してください。

デフォルトでは、ユーザーが表示可能なテーブルの大部分で、別のユーザーによって生成された行は、通常のユーザーには表示されません。通常のユーザーに SYSLOG ACCESS UNRESTRICTED を付与すると、ユーザーが表示できるテーブルのすべての行 (別のユーザーが生成した行を含む) を表示できます。詳細については、「ALTER USER」または「CREATE USER」を参照してください。SVV_TRANSACTIONS のすべての行は、すべてのユーザーが表示可能です。データの可視性の詳細については、AWS re:Post ナレッジベースの記事「Amazon Redshift データベースの通常のユーザーに、自分のクラスターの他のユーザーからのシステムテーブルのデータを見る許可を与えるにはどうすればよいですか?」を参照してください。

メタデータビューの場合、Amazon Redshift は SYSLOG ACCESS UNRESTRICTED が許可されているユーザーには表示を許可しません。

> Note
>
> システムテーブルに対する無制限のアクセス権限を付与されたユーザーは、別のユーザーが生成したデータへの可視性が提供されます。たとえば、STL_QUERY と STL_QUERY_TEXT には INSERT、UPDATE、および DELETE ステートメントのフルテキストが含まれており、ユーザーが生成した機密データがこれらに含まれている可能性があります。

スーパーユーザーは、すべてのテーブルのすべての行を表示できます。通常のユーザーに、スーパーユーザーが表示可能なテーブルのアクセス権を付与するには、そのテーブルに対する SELECT 権限の GRANT を、通常のユーザーに対して実行します。

システム生成クエリのフィルタ処理

通常、SVL_QUERY_SUMMARY、SVL_QLOG など、クエリに関連するシステムテーブルとビューには、Amazon Redshift がデータベースのステータスを監視するために使用する、自動的に生成された多数のステートメントが含まれます。これらのシステム生成クエリはスーパーユーザーに表示されますが、ほとんど役立ちません。`userid` 列を使用するシステムテーブルまたはシステムビューからの選択時にそれらのクエリを除外するには、条件 `userid > 1` を WHERE 句に追加します。例:

```
select * from svl_query_summary where userid > 1
```

プロビジョニング専用クエリから SYS モニタリングビュークエリへの移行

プロビジョニングされたクラスターを Amazon Redshift Serverless に移行する

プロビジョニングされたクラスターを Amazon Redshift Serverless に移行すると、プロビジョニングされたクラスターのデータのみを保存する以下のシステムビューがクエリで使用される場合があります。

- すべての STL ビュー
- すべての STV ビュー
- すべての SVCS ビュー
- すべての SVL ビュー
- 一部の SVV ビュー
  - Amazon Redshift Serverless でサポートされていない SVV ビューの完全なリストについては、「Amazon Redshift 管理ガイド」の「Amazon Redshift Serverless でのクエリとワークロードのモニタリング」の下部にあるリストを参照してください。

クエリを引き続き使用するには、SYS モニタリングビューで定義されている列のうち、プロビジョニング専用ビューの列に対応するものを使用するように、クエリを再調整してください。プロビジョニング専用ビューと SYS モニタリングビュー間のマッピング関係を確認するには、「SYS モニタリングビューに移行するためのシステムビューマッピング」を参照してください。

プロビジョニングされたクラスター上でのクエリの更新

Amazon Redshift Serverless に移行しない場合でも、既存のクエリを更新できます。SYS モニタリングビューは、使いやすく、複雑さを軽減するように設計されており、効果的なモニタリングとトラブルシューティングに役立つさまざまなメトリクスを提供します。複数のプロビジョニング専用ビューの情報を統合する SYS_QUERY_HISTORY や SYS_QUERY_DETAIL などの SYS ビューを使用すると、クエリを効率化できます。

SYS モニタリングビューを使用したクエリ識別子の改善

SYS_QUERY_HISTORY や SYS_QUERY_DETAIL などの SYS モニタリングビューには、ユーザーのクエリの識別子を保持する query_id 列が含まれています。同様に、STL_QUERY や SVL_QLOG などのプロビジョニング専用ビューにはクエリ列が含まれ、クエリ識別子も保持されます。ただし、SYS システムビューに記録されるクエリ識別子は、プロビジョニング専用ビューに記録されるクエリ識別子とは異なります。

SYS ビューの query_id 列の値とプロビジョニング専用ビューのクエリ列の値の違いは次のとおりです。

- SYS ビューでは、ユーザーが送信したクエリが query_id 列に元の形式で記録されます。Amazon Redshift オプティマイザは、パフォーマンスを向上させるためにそれらを子クエリに分割する場合がありますが、実行した 1 つのクエリでは SYS_QUERY_HISTORY に 1 つの行しか含まれません。個々の子クエリを確認したい場合は、SYS_QUERY_DETAIL で見つけることができます。
- プロビジョニング専用ビューでは、クエリ列に子クエリレベルでクエリが記録されます。Amazon Redshift オプティマイザが元のクエリを複数の子クエリに書き換えた場合、実行する 1 つのクエリに対して、異なるクエリ識別子の値を持つ複数の行がSTL_QUERY に存在します。

モニタリングクエリと診断クエリをプロビジョニング専用ビューから SYS ビューに移行するときは、この違いを考慮し、適宜クエリを編集します。Amazon Redshift がクエリを処理する方法の詳細については、「クエリプランと実行ワークフロー」を参照してください。

例

Amazon Redshift でクエリを記録する方法が、プロビジョニング専用ビューと SYS モニタリングビューで異なる例については、次のサンプルクエリを参照してください。これは Amazon Redshift で実行する場合と同じように記述されたクエリです。

```
SELECT
```

```
  s_name
  , COUNT(*) AS numwait
FROM
  supplier,
  lineitem l1,
  orders,
  nation
WHERE    s_suppkey = l1.l_suppkey
         AND o_orderkey = l1.l_orderkey
         AND o_orderstatus = 'F'
         AND l1.l_receiptdate > l1.l_commitdate
         AND EXISTS (SELECT
                         *
                     FROM
                       lineitem l2
                     WHERE  l2.l_orderkey = l1.l_orderkey
                            AND l2.l_suppkey <> l1.l_suppkey )
         AND NOT EXISTS (SELECT
                             *
                         FROM
                           lineitem l3
                         WHERE  l3.l_orderkey = l1.l_orderkey
                                AND l3.l_suppkey <> l1.l_suppkey
                                AND l3.l_receiptdate > l3.l_commitdate )
         AND s_nationkey = n_nationkey
         AND n_name = 'UNITED STATES'
GROUP BY
  s_name
ORDER BY
  numwait DESC
  , s_name LIMIT 100;
```

内部で Amazon Redshift クエリオプティマイザは、ユーザーが送信した上記のクエリを 5 つの子クエリに書き換えます。

最初の子クエリは、サブクエリをマテリアライズするための一時テーブルを作成します。

```
CREATE TEMP TABLE volt_tt_606590308b512(l_orderkey
                                        , l_suppkey
                                        , s_name   ) AS SELECT
                                                        l1.l_orderkey
                                                        , l1.l_suppkey
                                                        , public.supplier.s_name
```

```
                                                              FROM
                                                                public.lineitem AS l1,
                                                                public.nation,
                                                                public.orders,
                                                                public.supplier
                                                              WHERE  l1.l_commitdate <
 l1.l_receiptdate
                                                                     AND l1.l_orderkey =
 public.orders.o_orderkey
                                                                     AND l1.l_suppkey =
 public.supplier.s_suppkey
                                                                     AND public.nation.n_name
 = 'UNITED STATES'::CHAR(8)
                                                                     AND
 public.nation.n_nationkey = public.supplier.s_nationkey
                                                                     AND
 public.orders.o_orderstatus = 'F'::CHAR(1);
```

2 番目の子クエリは、一時テーブルから統計情報を収集します。

```
padb_fetch_sample: select count(*) from volt_tt_606590308b512;
```

3 番目の子クエリは、上記で作成した一時テーブルを参照して、別のサブクエリをマテリアライズするための別の一時テーブルを作成します。

```
CREATE TEMP TABLE volt_tt_606590308c2ef(l_orderkey
                                        , l_suppkey) AS (SELECT

 volt_tt_606590308b512.l_orderkey
                                                                 ,
 volt_tt_606590308b512.l_suppkey
                                                               FROM
                                                                 public.lineitem AS l2,
                                                                 volt_tt_606590308b512
                                                               WHERE  l2.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                                      AND l2.l_orderkey =
 volt_tt_606590308b512.l_orderkey)
                                                                      EXCEPT distinct (SELECT
 volt_tt_606590308b512.l_orderkey, volt_tt_606590308b512.l_suppkey
                                                                      FROM public.lineitem AS
 l3, volt_tt_606590308b512
```

```
                                                              WHERE l3.l_commitdate <
 l3.l_receiptdate
                                                                AND l3.l_suppkey <>
 volt_tt_606590308b512.l_suppkey
                                                                AND l3.l_orderkey =
 volt_tt_606590308b512.l_orderkey);
```

4 番目の子クエリは、一時テーブルの統計を再度収集します。

```
padb_fetch_sample: select count(*) from volt_tt_606590308c2ef
```

最後の子クエリは、上記で作成した一時テーブルを使用して出力を生成します。

```
SELECT
  volt_tt_606590308b512.s_name AS s_name
  , COUNT(*) AS numwait
FROM
  volt_tt_606590308b512,
  volt_tt_606590308c2ef
WHERE    volt_tt_606590308b512.l_orderkey = volt_tt_606590308c2ef.l_orderkey
         AND volt_tt_606590308b512.l_suppkey = volt_tt_606590308c2ef.l_suppkey
GROUP BY
  1
ORDER BY
  2 DESC
  , 1 ASC LIMIT 100;
```

プロビジョニング専用システムビュー STL_QUERY では、Amazon Redshift は次のように 5 つの行を子クエリレベルで記録します。

```
SELECT userid, xid, pid, query, querytxt::varchar(100);
FROM stl_query
WHERE xid = 48237350
ORDER BY xid, starttime;

 userid |   xid    |    pid     |  query   |
    querytxt
--------+----------+------------+----------
+------------------------------------------------------------------------------------------------------
    101 | 48237350 | 1073840810 | 12058151 | CREATE TEMP TABLE
 volt_tt_606590308b512(l_orderkey, l_suppkey, s_name) AS SELECT l1.l_orderkey, l1.l
```

```
    101 | 48237350 | 1073840810 | 12058152 | padb_fetch_sample: select count(*) from
 volt_tt_606590308b512
    101 | 48237350 | 1073840810 | 12058156 | CREATE TEMP TABLE
 volt_tt_606590308c2ef(l_orderkey, l_suppkey) AS (SELECT volt_tt_606590308b512.l_or
    101 | 48237350 | 1073840810 | 12058168 | padb_fetch_sample: select count(*) from
 volt_tt_606590308c2ef
    101 | 48237350 | 1073840810 | 12058170 | SELECT s_name , COUNT(*) AS numwait FROM
 supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
(5 rows)
```

SYS モニタリングビュー SYS_QUERY_HISTORY では、Amazon Redshift は次のようにクエリを記録します。

```
SELECT user_id, transaction_id, session_id, query_id, query_text::varchar(100)
FROM sys_query_history
WHERE transaction_id = 48237350
ORDER BY start_time;

 user_id | transaction_id | session_id | query_id |
         query_text
---------+----------------+------------+----------
+------------------------------------------------------------------------------------
     101 |       48237350 | 1073840810 | 12058149 | SELECT s_name , COUNT(*) AS numwait
 FROM supplier, lineitem l1, orders, nation WHERE s_suppkey = l1.
```

SYS_QUERY_DETAIL では、SYS_QUERY_HISTORY の query_id 値を使用して子クエリレベルの詳細を検索できます。child_query_sequence 列には、子クエリの実行順序が表示されます。SYS_QUERY_DETAIL の列の詳細については、「SYS_QUERY_DETAIL」を参照してください。

```
select user_id,
       query_id,
       child_query_sequence,
       stream_id,
       segment_id,
       step_id,
       start_time,
       end_time,
       duration,
       blocks_read,
       blocks_write,
       local_read_io,
```

```
       remote_read_io,
       data_skewness,
       time_skewness,
       is_active,
       spilled_block_local_disk,
       spilled_block_remote_disk
from sys_query_detail
where query_id = 12058149
      and step_id = -1
order by query_id,
         child_query_sequence,
         stream_id,
         segment_id,
         step_id;

 user_id | query_id | child_query_sequence | stream_id | segment_id | step_id |
       start_time          |          end_time          | duration | blocks_read |
 blocks_write | local_read_io | remote_read_io | data_skewness | time_skewness |
 is_active | spilled_block_local_disk | spilled_block_remote_disk
---------+----------+----------------------+-----------+------------+---------
+----------------------------+----------------------------+----------+-------------
+--------------+---------------+----------------+---------------+---------------
+-----------+--------------------------+--------------------------
     101 | 12058149 |                    1 |         0 |          0 |      -1 |
 2023-09-27 15:40:38.512415 | 2023-09-27 15:40:38.533333 |    20918 |           0 |
        0 |             0 |              0 |             0 |            44 | f
 |                        0 |                         0
     101 | 12058149 |                    1 |         1 |          1 |      -1 |
 2023-09-27 15:40:39.931437 | 2023-09-27 15:40:39.972826 |    41389 |          12 |
        0 |            12 |              0 |             0 |            77 | f
 |                        0 |                         0
     101 | 12058149 |                    1 |         2 |          2 |      -1 |
 2023-09-27 15:40:40.584412 | 2023-09-27 15:40:40.613982 |    29570 |          32 |
        0 |            32 |              0 |             0 |            25 | f
 |                        0 |                         0
     101 | 12058149 |                    1 |         2 |          3 |      -1 |
 2023-09-27 15:40:40.582038 | 2023-09-27 15:40:40.615758 |    33720 |           0 |
        0 |             0 |              0 |             0 |             1 | f
 |                        0 |                         0
     101 | 12058149 |                    1 |         3 |          4 |      -1 |
 2023-09-27 15:40:46.668766 | 2023-09-27 15:40:46.705456 |    36690 |          24 |
        0 |            15 |              0 |             0 |            17 | f
 |                        0 |                         0
```

```
    101 | 12058149 |                    1 |          4 |          5 |       -1 |
 2023-09-27 15:40:46.707209 | 2023-09-27 15:40:46.709176 |     1967 |            0 |
        0 |            0 |                0 |               0 |           18 | f
 |                       0 |                          0
    101 | 12058149 |                    1 |          4 |          6 |       -1 |
 2023-09-27 15:40:46.70656  | 2023-09-27 15:40:46.71289  |     6330 |            0 |
        0 |            0 |                0 |               0 |            0 | f
 |                       0 |                          0
    101 | 12058149 |                    1 |          5 |          7 |       -1 |
 2023-09-27 15:40:46.71405  | 2023-09-27 15:40:46.714343 |      293 |            0 |
        0 |            0 |                0 |               0 |            0 | f
 |                       0 |                          0
    101 | 12058149 |                    2 |          0 |          0 |       -1 |
 2023-09-27 15:40:52.083907 | 2023-09-27 15:40:52.087854 |     3947 |            0 |
        0 |            0 |                0 |               0 |           35 | f
 |                       0 |                          0
    101 | 12058149 |                    2 |          1 |          1 |       -1 |
 2023-09-27 15:40:52.089632 | 2023-09-27 15:40:52.091129 |     1497 |            0 |
        0 |            0 |                0 |               0 |           11 | f
 |                       0 |                          0
    101 | 12058149 |                    2 |          1 |          2 |       -1 |
 2023-09-27 15:40:52.089008 | 2023-09-27 15:40:52.091306 |     2298 |            0 |
        0 |            0 |                0 |               0 |            0 | f
 |                       0 |                          0
    101 | 12058149 |                    3 |          0 |          0 |       -1 |
 2023-09-27 15:40:56.882013 | 2023-09-27 15:40:56.897282 |    15269 |            0 |
        0 |            0 |                0 |               0 |           29 | f
 |                       0 |                          0
    101 | 12058149 |                    3 |          1 |          1 |       -1 |
 2023-09-27 15:40:59.718554 | 2023-09-27 15:40:59.722789 |     4235 |            0 |
        0 |            0 |                0 |               0 |           13 | f
 |                       0 |                          0
    101 | 12058149 |                    3 |          2 |          2 |       -1 |
 2023-09-27 15:40:59.800382 | 2023-09-27 15:40:59.807388 |     7006 |            0 |
        0 |            0 |                0 |               0 |           58 | f
 |                       0 |                          0
    101 | 12058149 |                    3 |          3 |          3 |       -1 |
 2023-09-27 15:41:06.488685 | 2023-09-27 15:41:06.493825 |     5140 |            0 |
        0 |            0 |                0 |               0 |           56 | f
 |                       0 |                          0
    101 | 12058149 |                    3 |          3 |          4 |       -1 |
 2023-09-27 15:41:06.486206 | 2023-09-27 15:41:06.497756 |    11550 |            0 |
        0 |            0 |                0 |               0 |            2 | f
 |                       0 |                          0
```

```
    101 | 12058149 |                    3 |          4 |          5 |       -1 |
 2023-09-27 15:41:06.499201 | 2023-09-27 15:41:06.500851 |      1650 |            0 |
        0 |              0 |                0 |               0 |            15 | f
 |                        0 |                          0
    101 | 12058149 |                    3 |          4 |          6 |       -1 |
 2023-09-27 15:41:06.498609 | 2023-09-27 15:41:06.500949 |      2340 |            0 |
        0 |              0 |                0 |               0 |             0 | f
 |                        0 |                          0
    101 | 12058149 |                    3 |          5 |          7 |       -1 |
 2023-09-27 15:41:06.502945 | 2023-09-27 15:41:06.503282 |       337 |            0 |
        0 |              0 |                0 |               0 |             0 | f
 |                        0 |                          0
    101 | 12058149 |                    4 |          0 |          0 |       -1 |
 2023-09-27 15:41:06.62899  | 2023-09-27 15:41:06.631452 |      2462 |            0 |
        0 |              0 |                0 |               0 |            22 | f
 |                        0 |                          0
    101 | 12058149 |                    4 |          1 |          1 |       -1 |
 2023-09-27 15:41:06.632313 | 2023-09-27 15:41:06.63391  |      1597 |            0 |
        0 |              0 |                0 |               0 |            20 | f
 |                        0 |                          0
    101 | 12058149 |                    4 |          1 |          2 |       -1 |
 2023-09-27 15:41:06.631726 | 2023-09-27 15:41:06.633813 |      2087 |            0 |
        0 |              0 |                0 |               0 |             0 | f
 |                        0 |                          0
    101 | 12058149 |                    5 |          0 |          0 |       -1 |
 2023-09-27 15:41:12.571974 | 2023-09-27 15:41:12.584234 |     12260 |            0 |
        0 |              0 |                0 |               0 |            39 | f
 |                        0 |                          0
    101 | 12058149 |                    5 |          0 |          1 |       -1 |
 2023-09-27 15:41:12.569815 | 2023-09-27 15:41:12.585391 |     15576 |            0 |
        0 |              0 |                0 |               0 |             4 | f
 |                        0 |                          0
    101 | 12058149 |                    5 |          1 |          2 |       -1 |
 2023-09-27 15:41:13.758513 | 2023-09-27 15:41:13.76401  |      5497 |            0 |
        0 |              0 |                0 |               0 |            39 | f
 |                        0 |                          0
    101 | 12058149 |                    5 |          1 |          3 |       -1 |
 2023-09-27 15:41:13.749    | 2023-09-27 15:41:13.772987 |     23987 |            0 |
        0 |              0 |                0 |               0 |            32 | f
 |                        0 |                          0
    101 | 12058149 |                    5 |          2 |          4 |       -1 |
 2023-09-27 15:41:13.799526 | 2023-09-27 15:41:13.813506 |     13980 |            0 |
        0 |              0 |                0 |               0 |            62 | f
 |                        0 |                          0
```

```
     101 | 12058149 |                    5 |          2 |            5 |        -1 |
  2023-09-27 15:41:13.798823 | 2023-09-27 15:41:13.813651 |     14828 |            0 |
         0 |              0 |                0 |               0 |              0 | f
  |                        0 |                           0
 (28 rows)
```

システムテーブルクエリ、プロセス、セッション ID

システムテーブルに表示されるクエリ ID、プロセス ID、セッション ID を分析するときは、以下の点に注意してください。

- クエリ ID の値 (query_id や query などの列内) は、長期に再利用できます。
- プロセス ID またはセッション ID の値 (process_id、pid、session_id などの列内) は、長期に再利用できます。
- トランザクション ID の値 (transaction_id や xid などの列内) は一意です。

SVV メタデータビュー

SVV ビューは、データベースオブジェクトに関する情報を含む Amazon Redshift のシステムビューです。

> Note
>
> Amazon Redshift では、何らかの理由でデータベースの応答が失敗した場合、エラーではなく警告を報告します。Amazon Redshift は、データ共有内のオブジェクトをクエリしているときは ERROR メッセージを送信しません。

トピック

- SVV_ACTIVE_CURSORS
- SVV_ALL_COLUMNS
- SVV_ALL_SCHEMAS
- SVV_ALL_TABLES
- SVV_ALTER_TABLE_RECOMMENDATIONS
- SVV_ATTACHED_MASKING_POLICY
- SVV_COLUMNS

- SVV_REDSHIFT_DATABASES
- SVV_REDSHIFT_FUNCTIONS
- SVV_REDSHIFT_SCHEMA_QUOTA
- SVV_REDSHIFT_SCHEMAS
- SVV_REDSHIFT_TABLES
- SVV_RELATION_PRIVILEGES
- SVV_RLS_APPLIED_POLICY
- SVV_RLS_ATTACHED_POLICY
- SVV_RLS_POLICY
- SVV_RLS_RELATION
- SVV_ROLE_GRANTS
- SVV_ROLES
- SVV_SCHEMA_PRIVILEGES
- SVV_SCHEMA_QUOTA_STATE
- SVV_SYSTEM_PRIVILEGES
- SVV_TABLE_INFO
- SVV_TABLES
- SVV_TRANSACTIONS
- SVV_USER_GRANTS
- SVV_USER_INFO
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SVV_ACTIVE_CURSORS

SVV_ACTIVE_CURSORS は、現在開いているカーソルの詳細を表示します。詳細については、「DECLARE」を参照してください。

SVV_ACTIVE_CURSORS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。ユーザーは、自身が開いたカーソルのみを表示できます。スーパーユーザーはすべてのカーソルを表示できます。

テーブルの列

列名	データ型	説明
user_id	integer	カーソルを作成したユーザーの ID。
cursor_name	varCHAR(128)	カーソルの名前。
transaction_id	bigint(128)	トランザクションの ID。
session_id	integer	アクティブなカーソルがあるプロセスの ID。
declare_time	timestamp	カーソルを宣言した時刻。
total_bytes	bigint	カーソル結果セットのサイズ (バイト単位)。
total_rows	bigint	カーソル結果セット内の行数。
fetched_rows	bigint	カーソル結果セットから現在取得されている行数。
cursor_storage_limit_used_percent	integer	カーソルが現在使用しているディスク容量の割合。

SVV_ALL_COLUMNS

SVV_ALL_COLUMNS に示されているように、SVV_ALL_COLUMNS を使用して、Amazon Redshift テーブルの列のユニオンと、すべての外部テーブルからのすべての外部列の統合リストを表示します Amazon Redshift 列の詳細については、「SVV_REDSHIFT_COLUMNS」を参照してください。

SVV_ALL_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	データベースの名前。
schema_name	varCHAR(128)	スキーマの名前。
table_name	varCHAR(128)	テーブルの名前。
column_name	varCHAR(128)	列の名前。
ordinal_position	integer	テーブルの列の位置。
column_default	varCHAR(4000)	列のデフォルト値。
is_nullable	varCHAR(3)	列が NULL であるかを示す値。指定できる値は yes と no です。
data_type	varCHAR(128)	列のデータ型。
character_maximum_length	integer	列で許容される文字の最大数。
numeric_precision	integer	数値の精度。
numeric_scale	integer	スケールの数値。
remarks	varCHAR(256)	解説。

サンプルクエリ

次の例では、SVV_ALL_COLUMNS の出力を返します。

```
SELECT *
FROM svv_all_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    SCHEMA_NAME
```

```
LIMIT 5;

 database_name | schema_name |      table_name        | column_name | ordinal_position
 | column_default | is_nullable | data_type | character_maximum_length |
 numeric_precision | numeric_scale | remarks
 --------------+-------------+----------------------+------------+-----------------
+----------------+------------+----------+-------------------------
+-------------------+--------------+---------
   tickit_db    |    public    | tickit_sales_redshift |    buyerid   |        4          |
                 |      NO     |  integer  |                           |        32
  |       0        |
   tickit_db    |    public    | tickit_sales_redshift | commission  |        9          |
                 |     YES     |  numeric  |                           |         8
  | 2          |
   tickit_db    |    public    | tickit_sales_redshift |    dateid    |        7          |
                 |      NO     |  smallint |                           |        16
  |       0        |
   tickit_db    |    public    | tickit_sales_redshift |   eventid    |        5          |
                 |      NO     |  integer  |                           |        32
  |       0        |
   tickit_db    |    public    | tickit_sales_redshift |    listid    |        2          |
                 |      NO     |  integer  |                           |        32
  |       0        |
```

SVV_ALL_SCHEMAS

SVV_REDSHIFT_SCHEMAS に示すように、SVV_ALL_SCHEMAS を使用して、Amazon Redshift スキーマのユニオン、およびすべてのデータベースからのすべての外部スキーマの統合リストを表示します。Amazon Redshift スキーマの詳細については、「SVV_REDSHIFT_SCHEMAS」を参照してください。

SVV_ALL_SCHEMAS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	スキーマが存在するデータベースの名前。

列名	データ型	説明
schema_name	varCHAR(128)	スキーマの名前。
schema_owner	integer	スキーマの所有者のユーザー ID。ユーザー ID の詳細については、「PG_USER_INFO」を参照してください。
schema_type	varCHAR(128)	スキーマのタイプ。指定できる値は、外部スキーマ、ローカルスキーマ、および共有スキーマです。
schema_acl	varCHAR(128)	スキーマに指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。
source_database	varCHAR(128)	外部スキーマ用のソースデータベースの名前。
schema_option	varCHAR(256)	スキーマのオプション。これは外部スキーマの属性です。

サンプルクエリ

次の例では、SVV_ALL_SCHEMAS の出力を返します。

```
SELECT *
FROM svv_all_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;


 database_name |    schema_name     | schema_owner | schema_type | schema_acl |
 source_database | schema_option
---------------+--------------------+--------------+-------------+------------
+-----------------+---------------
```

```
    tickit_db    |        public         |        1        |   shared     |             |
            |
```

SVV_ALL_TABLES

SVV_REDSHIFT_TABLES に示すように、SVV_ALL_TABLES を使用して、Amazon Redshift テーブルのユニオン、およびすべての外部スキーマからのすべての外部テーブルの統合リストを表示します。Amazon Redshift テーブルの詳細については、「SVV_REDSHIFT_TABLES」を参照してください。

SVV_ALL_TABLES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	テーブルが存在するデータベースの名前。
schema_name	varCHAR(128)	テーブルのスキーマ名。
table_name	varCHAR(128)	テーブルの名前。
table_acl	varCHAR(128)	テーブルに指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。
table_type	varCHAR(128)	テーブルの種類。指定できる値は、ビュー、ベーステーブル、外部テーブル、共有テーブルです。
remarks	varCHAR(256)	解説。

サンプルクエリ

次の例では、SVV_ALL_TABLES の出力を返します。

```
SELECT *
FROM svv_all_tables
WHERE database_name = 'tickit_db'
ORDER BY TABLE_NAME,
    SCHEMA_NAME
LIMIT 5;

 database_name | schema_name |         table_name         | table_type | table_acl |
 remarks
---------------+-------------+--------------------------+------------+-----------
+---------
   tickit_db    |    public    | tickit_category_redshift |    TABLE    |            |
   tickit_db    |    public    |   tickit_date_redshift   |    TABLE    |            |
   tickit_db    |    public    |   tickit_event_redshift  |    TABLE    |            |
   tickit_db    |    public    | tickit_listing_redshift  |    TABLE    |            |
   tickit_db    |    public    |   tickit_sales_redshift  |    TABLE    |            |
```

table_acl の値が NULL の場合、対応するテーブルに明示的に付与されたアクセス権限はありません。

SVV_ALTER_TABLE_RECOMMENDATIONS

テーブルで現在の Amazon Redshift Advisor レコメンデーションを記録します。このビューには、自動最適化用に定義されているかどうかにかかわらず、すべてのテーブルのレコメンデーションが表示されます。テーブルが自動最適化用に定義されているかどうかを確認するには、「SVV_TABLE_INFO」を参照してください。エントリは、現在のセッションのデータベースに表示されているテーブルに対してのみ表示されます。(Amazon Redshift またはユーザーのいずれかによって) レコメンデーションが適用されると、ビューに表示されなくなります。

SVV_ALTER_TABLE_RECOMMENDATIONS は、スーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
type	character (30)	レコメンデーションのタイプ。可能な値は、distkey と sortkey です。
データベース	character (128)	データベース名。
table_id	integer	テーブル識別子。
group_id	integer	レコメンデーションのグループ番号。最大の利点を得るには、グループ内のすべての推奨事項を適用する必要があります。可能な値は、ソートキーのレコメンデーションの場合は -1、ディストリビューションキーのレコメンデーションの場合は 0 より大きい値です。
ddl	character (1024)	レコメンデーションを適用するために実行する必要がある SQL ステートメント。
auto_eligible	character (1)	この値は、レコメンデーションが Amazon Redshift を自動的に実行する資格があるかどうかを示します。この値が t の場合は true、f の場合は false を示します。

サンプルクエリ

次の例では、結果の行に、ディストリビューションキーおよびソートキーの推奨事項が示されています。また、行には、レコメンデーションが Amazon Redshift で自動的に適用される資格があるかどうかも表示されます。

```
select type, database, table_id, group_id, ddl, auto_eligible
from svv_alter_table_recommendations;
```

```
 type       | database | table_id | group_id | ddl

                      | auto_eligible
```

```
diststyle | db0      | 117884   | 2         | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117892   | 2         | ALTER TABLE "sch"."dp21235_tbl_1" ALTER
DISTSTYLE KEY DISTKEY "c0"
                     | f
diststyle | db0      | 117885   | 1         | ALTER TABLE "sch"."catalog_returns"
ALTER DISTSTYLE KEY DISTKEY "cr_sold_date_sk", ALTER COMPOUND SORTKEY
("cr_sold_date_sk","cr_returned_time_sk") | t
sortkey   | db0      | 117890   | -1        | ALTER TABLE "sch"."customer_addresses"
ALTER COMPOUND SORTKEY ("ca_address_sk")
                      | t
```

SVV_ATTACHED_MASKING_POLICY

SVV_ATTACHED_MASKING_POLICY を使用して、現在接続されているデータベースに、ポリシーがアタッチされているすべてのリレーションとロール/ユーザーを表示します。

SVV_ATTACHED_MASKING_POLICY を閲覧できるのは、スーパーユーザーおよび `sys:secadmin` ロールを持つユーザーだけです。通常のユーザーには 0 行が表示されます。

テーブルの列

列名	データ型	説明
policy_name	text	テーブルにアタッチされているマスキングポリシーの名前。
schema_name	text	ポリシーがアタッチされているテーブルのスキーマ。
table_name	text	ポリシーがアタッチされているテーブルの名前。
table_type	text	ポリシーがアタッチされているテーブルのタイプ。
grantor	text	このポリシーをアタッチしたユーザーの名前。

列名	データ型	説明
grantee	text	ポリシーがアタッチされているユーザー/ロールの名前。
grantee_type	text	被付与者のタイプ。ロール、ユーザー、パブリックのいずれでもかまいません。
priority	整数	アタッチされたポリシーの優先順位。
input_columns	text	アタッチされたポリシーの入力列属性。
output_columns	text	アタッチされたポリシーの出力列属性。

内部関数

SVV_ATTACHED_MASKING_POLICY は以下の内部関数をサポートしています。

mask_get_policy_for_role_on_column

特定の列/ロールペアに適用される最も優先度の高いポリシーを取得します。

構文

```
mask_get_policy_for_role_on_column
                         (relschema,
                         relname,
                         colname,
                         rolename);
```

パラメータ

relschema

ポリシーがアタッチされているスキーマの名前。

relname

ポリシーがアタッチされているテーブルの名前。

colname

ポリシーがアタッチされている列の名前。

rolename

ポリシーがアタッチされているロールの名前。

mask_get_policy_for_user_on_column

特定の列/ユーザーのペアに適用される最も優先度の高いポリシーを取得します。

構文

```
mask_get_policy_for_user_on_column
                            (relschema,
                            relname,
                            colname,
                            username);
```

パラメータ

relschema

ポリシーがアタッチされているスキーマの名前。

relname

ポリシーがアタッチされているテーブルの名前。

colname

ポリシーがアタッチされている列の名前。

rolename

このポリシーをアタッチしたユーザーの名前。

SVV_COLUMNS

SVV_COLUMNS を使用して、遅延ビューを含む、ローカルおよび外部のテーブルの列とビューに関するカタログ情報を表示します。

SVV_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVV_COLUMNS ビューは システムカタログテーブル (PG プレフィックスを持つテーブル) および SVV_EXTERNAL_COLUMNS システムビューからのテーブルメタデータと結合します。システムカタログテーブルは、Amazon Redshift データベースのテーブルを説明します。SVV_EXTERNAL_COLUMNS は Amazon Redshift Spectrum とともに使用する外部テーブルを説明します。

すべてのユーザーがシステムカタログのテーブルからのすべての行を表示できます。通常のユーザーは、アクセス権限が付与されている外部テーブル専用の SVV_EXTERNAL_COLUMNS ビューから列の定義を見ることができます。通常のユーザーはシステムカタログテーブルでテーブルメタデータを見ることができますが、テーブルを所有している場合またはアクセス権限が付与されている場合に限り、ユーザー定義のテーブルからデータを選択できます。

テーブルの列

列名	データ型	説明
table_catalog	text	テーブルがあるカタログの名前。
table_schema	text	テーブルのスキーマ名。
table_name	text	テーブルの名前。
column_name	text	列の名前。
ordinal_position	int	テーブルの列の位置。
column_default	text	列のデフォルト値。
is_nullable	text	列が NULL であるかを示す値。

列名	データ型	説明
data_type	text	列のデータ型。
character_maximum_length	int	列で許容される文字の最大数。
numeric_precision	int	数値の精度。data_type 列が数値の場合、この列は値全体の有効桁数を返します。
numeric_precision_radix	int	数値の精度基数。data_type 列が数値の場合、この列は numeric_precision 列と numeric_scale 列の基底を返します。
numeric_scale	int	スケールの数値。data_type 列が数値の場合、この列は 10 進値の有効桁数を返します。
datetime_precision	int	日時の精度。
interval_type	text	間隔のタイプ。
interval_precision	text	間隔の精度。
character_set_catalog	text	キャラクタセットカタログ。
character_set_schema	text	キャラクタセットスキーマ。
character_set_name	text	キャラクタセット名。
collation_catalog	text	照合カタログ。
collation_schema	text	照合スキーマ。
collation_name	text	照合名。
domain_name	text	ドメイン名。

列名	データ型	説明
remarks	text	解説。

SVV_COLUMN_PRIVILEGES

SVV_COLUMN_PRIVILEGES を使用して、現在のデータベース内のユーザー、ロール、およびグループに明示的に付与されている列のアクセス許可を表示します。

SVV_COLUMN_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
namespace_name	text	指定されたリレーションが存在する名前空間の名前。
relation_name	text	リレーションの名前。
column_name	text	列の名前。
privilege_type	text	アクセス許可の種類。指定できる値は SELECT または UPDATE です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。

列名	データ型	説明
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。

サンプルクエリ

次の例では、SVV_COLUMN_PRIVILEGES の結果を示します。

```
SELECT
 namespace_name,relation_name,COLUMN_NAME,privilege_type,identity_name,identity_type
FROM svv_column_privileges WHERE relation_name = 'lineitem';

 namespace_name | relation_name | column_name | privilege_type | identity_name |
 identity_type
----------------+---------------+-------------+----------------+---------------
+----------------
    public      |   lineitem    | l_orderkey  |     SELECT     |    reguser    |
 user
    public      |   lineitem    | l_orderkey  |     SELECT     |     role1     |
 role
    public      |   lineitem    | l_partkey   |     SELECT     |    reguser    |
 user
    public      |   lineitem    | l_partkey   |     SELECT     |     role1     |
 role
```

SVV_DATABASE_PRIVILEGES

SVV_DATABASE_PRIVILEGES を使用して、Amazon Redshift クラスターのユーザー、ロール、およびグループに明示的に付与されたデータベースのアクセス許可を表示します。

SVV_DATABASE_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
database_name	text	データベースの名前。
privilege_type	text	アクセス許可の種類。データベースの privilege_scope を使用するアクセス許可の場合、指定できる値は USAGE、CREATE、TEMPORARY、TEMP、ALTER です。データベース以外の privilege_scope 値の場合、指定できる値には、アクセス許可のスコープで使用できる任意のアクセス許可タイプが含まれます。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。
privilege_scope	text	privilege_type で指定されたアクセス許可の範囲。可能な値は以下のとおりです。 • DATABASE • SCHEMAS • TABLES • FUNCTIONS • LANGUAGES 範囲指定されたアクセス許可の詳細については、「スコープ設定アクセス許可」を参照してください。

サンプルクエリ

次の例では、SVV_DATABASE_PRIVILEGES の結果を示します。

```
SELECT database_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_database_privileges
WHERE database_name = 'test_db';

 database_name | privilege_type | identity_name | identity_type | admin_option
---------------+----------------+---------------+---------------+--------------
     test_db   |     CREATE     |     reguser   |      user     |     False
     test_db   |     CREATE     |      role1    |      role     |     False
     test_db   |     TEMP       |      public   |      public   |     False
     test_db   |     TEMP       |      role1    |      role     |     False
```

SVV_DATASHARE_PRIVILEGES

SVV_DATASHARE_PRIVILEGES を使用して、Amazon Redshift クラスターのユーザー、ロール、およびグループに明示的に付与されたデータ共有のアクセス許可を表示します。

SVV_DATASHARE_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
datashare_name	text	データ共有の名前。
privilege_type	text	アクセス許可の種類。指定できる値は ALTER または SHARE です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。

列名	データ型	説明
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_DATASHARE_PRIVILEGES の結果を示します。

```
SELECT datashare_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_datashare_privileges
WHERE datashare_name = 'demo_share';

 datashare_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
   demo_share   |     ALTER      |    superuser   |     user      |   False
   demo_share   |     ALTER      |    reguser     |     user      |   False
```

SVV_DATASHARES

SVV_DATASHARES を使用して、クラスター上に作成されたデータ共有およびクラスターと共有されているデータ共有のリストを表示します。

SVV_DATASHARES は以下のユーザーに表示されます。

- スーパーユーザー
- データ共有の所有者
- データ共有に対する ALTER または USAGE アクセス許可を持つユーザー

他のユーザーは行を確認できません。ALTER および USAGE アクセス許可に関する詳細は、「GRANT」を参照してください。

テーブルの列

列名	データ型	説明
share_name	varCHAR(128)	データ共有の名前。
share_id	integer	データ共有の ID。
share_owner	integer	データ共有の所有者。
source_database	varCHAR(128)	このデータ共有のソースデータベース。
consumer_database	varCHAR(128)	このデータ共有から作成されたコンシューマデータベース。
share_type	varCHAR(8)	データ共有のタイプ。指定できる値は、INBOUND と OUTBOUND です。
createdate	タイムゾーンなしのタイムスタンプ	データ共有が作成された日付。
is_publicaccessible	boolean	データ共有を公的にアクセス可能なクラスターと共有できるかどうかを指定するプロパティ。
share_acl	varCHAR(256)	データ共有に指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。
producer_account	varCHAR(16)	データ共有プロデューサのアカウント ID。

列名	データ型	説明
producer_namespace	varCHAR(64)	データ共有プロデューサクラスターの一意のクラスター識別子。
managed_by	varCHAR(64)	データ共有の管理を行う AWS サービスを指定するプロパティ。

使用に関する注意事項

追加のメタデータの取得 — share_owner 列で返された整数を使用して、SVL_USER_INFO の usesysid と結合してデータ共有所有者に関するデータを取得できます。これには、名前と追加のプロパティが含まれます。

サンプルクエリ

次の例では、SVV_DATASHARES の出力を返します。

```
SELECT share_owner, source_database, share_type, is_publicaccessible
FROM svv_datashares
WHERE share_name LIKE 'tickit_datashare%'
AND source_database = 'dev';

  share_owner | source_database | share_type  | is_publicaccessible
--------------+-----------------+-------------+----------------------
     100      |       dev       |   OUTBOUND  |        True
(1 rows)
```

次の例では、アウトバウンドデータ共有のために、SVV_DATASHARES の出力を返します。

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'OUTBOUND';

   share_name    | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account|         producer_namespace
   | managed_by
```

```
----------------+-------------+-----------------+-------------------
+------------+--------------------+----------+-----------------
+-------------------------------------+------------
    salesshare  |      1      |       dev       |                    |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |

 marketingshare |      1      |       dev       |                    |  OUTBOUND  |
   True         |           |   123456789012  | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

次の例では、インバウンドデータ共有のために SVV_DATASHARES の出力を返します。

```
SELECT share_name, share_owner, btrim(source_database), btrim(consumer_database),
 share_type, is_publicaccessible, share_acl, btrim(producer_account),
 btrim(producer_namespace), btrim(managed_by) FROM svv_datashares WHERE share_type =
 'INBOUND';

  share_name    | share_owner | source_database | consumer_database | share_type |
 is_publicaccessible | share_acl | producer_account |         producer_namespace
    | managed_by
----------------+-------------+-----------------+-------------------
+------------+--------------------+----------+------------------
+-------------------------------------+------------
  salesshare    |             |                 |                    |  INBOUND   |
   False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d
 |
 marketingshare |             |                 |                    |  INBOUND   |
  False         |           |  123456789012    | 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
 ADX
```

SVV_DATASHARE_CONSUMERS

SVV_DATASHARE_CONSUMERS を使用して、クラスター上に作成されたデータ共有コンシューマーのリストを表示します。

SVV_DATASHARE_CONSUMER は、次のユーザーに表示されます。

- スーパーユーザー
- データ共有の所有者
- データ共有に対する ALTER または USAGE アクセス許可を持つユーザー

他のユーザーは行を確認できません。ALTER および USAGE アクセス許可に関する詳細は、「GRANT」を参照してください。

テーブルの列

列名	データ型	説明
share_name	varCHAR(128)	データ共有の名前。
consumer_account	varCHAR(16)	データ共有コンシューマーのアカウント ID。
consumer_namespace	varCHAR(64)	データ共有コンシューマークラスターの一意のクラスター識別子。
share_date	タイムゾーンなしのタイムスタンプ	データ共有が共有された日付。

サンプルクエリ

次の例では、SVV_DATASHARE_CONSUMERS の出力を返します。

```
SELECT count(*)
FROM svv_datashare_consumers
WHERE share_name LIKE 'tickit_datashare%';

1
```

SVV_DATASHARE_OBJECTS

SVV_DATASHARE_OBJECTS を使用して、クラスター上で作成された、またはクラスターと共有されたすべてのデータ共有内のオブジェクトのリストを表示します。

SVV_DATASHARE_OBJECTS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

データ共有のリストを表示する方法については、「SVV_DATASHARES」を参照してください。

テーブルの列

列名	データ型	説明
share_type	varCHAR(8)	指定されたデータ共有のタイプ。指定できる値は、OUTBOUND と INBOUND です。
share_name	varCHAR(128)	データ共有の名前。
object_type	varCHAR(64)	指定されたオブジェクトのタイプ。使用可能な値は、スキーマ、テーブル、ビュー、遅延バインディングビュー、マテリアライズドビュー、および関数です。
object_name	varCHAR(512)	オブジェクトの名前。オブジェクト名は、schema1.t1 などのスキーマ名を含むように拡張されます。
producer_account	varCHAR(16)	データ共有プロデューサのアカウント ID。
producer_namespace	varCHAR(64)	データ共有プロデューサクラスターの一意のクラスター識別子。
include_new	boolean	指定したスキーマで作成される将来のテーブル、ビュー、または SQL ユーザー定義関数 (UDF) を、データ共有に追加するかどうかを指定するプロパティ。このパラメータは、OUBUND データ共有にのみ関連し、またデータシェア内の

列名	データ型	説明
		スキーマタイプに対してのみ意味を持ちます。

サンプルクエリ

次の例では、SVV_DATASHARE_OBJECTS の出力が返されます。

```
SELECT share_type,
    btrim(share_name)::varchar(16) AS share_name,
    object_type,
    object_name
FROM svv_datashare_objects
WHERE share_name LIKE 'tickit_datashare%'
AND object_name LIKE '%tickit%'
ORDER BY object_name
LIMIT 5;

 share_type |     share_name     | object_type |           object_name
------------+--------------------+-------------+---------------------------------
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_category_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_date_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_event_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_listing_redshift
 OUTBOUND   |  tickit_datashare  |    table    |  public.tickit_sales_redshift
```

```
SELECT * FROM SVV_DATASHARE_OBJECTS WHERE share_name like 'sales%';

share_type | share_name | object_type | object_name  | producer_account |
 producer_namespace           | include_new
-----------+------------+-------------+--------------+------------------
+--------------------------------------+-------------
 OUTBOUND  | salesshare | schema      | public       | 123456789012     |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |      t
 OUTBOUND  | salesshare | table       | public.sales | 123456789012     |
 13b8833d-17c6-4f16-8fe4-1a018f5ed00d |
```

SVV_DEFAULT_PRIVILEGES

SVV_DEFAULT_PRIVILEGES を使用して、Amazon Redshift クラスターでユーザーがアクセスできるデフォルト権限を表示します。

SVV_DEFAULT_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、自分に付与されたデフォルトのアクセス許可のみを表示できます。

テーブルの列

列名	データ型	説明
schema_name	text	スキーマの名前。
object_type	text	オブジェクトのタイプ。使用できる値は、RELATION、FUNCTION、または PROCEDURE です。
owner_id	integer	所有者 ID。指定できる値はユーザー ID です。
owner_name	text	所有者の名前。
owner_type	text	所有者のタイプ。指定できる値はユーザーです。
privilege_type	text	権限のタイプ。指定できる値は、INSERT、SELECT、UPDATE、DELETE、RULE、REFERENCES TRIGGER、DROP、EXECUTE です。
grantee_id	integer	被付与者 ID。指定できる値は、ユーザー ID、ロール ID、グループ ID です。
grantee_type	text	被付与者のタイプ。指定できる値は、ユーザー、ロール、パブリックです。

列名	データ型	説明
admin_option	ブール値	ユーザーが他のユーザーおよびロールに許可を付与できるかどうかを示す値。ロールおよびグループのタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_DEFAULT_PRIVILEGES の出力を返します。

```
SELECT * from svv_default_privileges;

 schema_name |    object_type     | owner_id | owner_name | owner_type | privilege_type
 | grantee_id | grantee_name | grantee_type | admin_option
-------------+-------------------+--------- +------------+------------+----------------
+------------+--------------+--------------+-------------+
   public    |      RELATION      |    106   |     u1     |    user    |     UPDATE
 |      107    |      u2      |     user     |      f      |
   public    |      RELATION      |    106   |     u1     |    user    |     SELECT
 |      107    |      u2      |     user     |      f      |
```

SVV_DISKUSAGE

Amazon Redshift は STV_TBL_PERM テーブルと STV_BLOCKLIST テーブルを結合して、SVV_DISKUSAGE システムビューを作成します。SVV_DISKUSAGE ビューにはデータベースのテーブルに対するデータ割り当てに関する情報が含まれます。

次の例で示されているように集計クエリを SVV_DISKUSAGE と一緒に使用すると、データベースあたり、テーブルあたり、スライスあたり、列あたりに割り当てられたディスクブロックの数が算出されます。各データブロックのサイズは 1 MB です。または STV_PARTITIONS を使用して、ディスク利用に関する概要を見ることができます。

SVV_DISKUSAGE はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
db_id	integer	データベース ID。
name	character (72)	テーブル名。
slice	integer	テーブルに割り当てられたデータスライス。
col	integer	列のゼロベースインデックス。作成するテーブルにはすべて、3 つの非表示列 (INSERT_XID、DELETE_XID、ROW_ID (OID)) が追加されます。例えばユーザー定義列が 3 つあるテーブルには、実際には 6 つの列が含まれます。ユーザー定義列の初期番号は 0、1、2 です。その場合、INSERT_XID、DELETE_XID、および ROW_ID 列はそれぞれ、3、4、および 5 となります。
tbl	integer	テーブル ID。
blocknum	integer	データブロックの ID。
num_values	integer	ブロックに含まれる値の数。
minvalue	bigint	ブロックに含まれる最小値。
maxvalue	bigint	ブロックに含まれる最大値。
sb_pos	integer	ディスクのスーパーブロックの位置の内部識別子。
pinned	integer	事前ロードの一環としてブロックをメモリにピンするかどうかを示します。0 = false、1 = true。デフォルトは false です。
on_disk	integer	ブロックを自動的にディスクに保存するかどうかを示します。0 = false、1 = true。デフォルトは false です。
modified	integer	ブロックが修正されたかどうかを示します。0 = false、1 = true。デフォルトは false です。

列名	データ型	説明
hdr_modified	integer	ブロックヘッダーが修正されたかどうかを示します。0 = false、1 = true。デフォルトは false です。
unsorted	integer	ブロックが未ソートかどうかを示します。0 = false、1 = true。デフォルトは true です。
tombstone	integer	内部使用を目的とします。
preferred_diskno	integer	ディスクにエラーがない場合に、ブロックが存在すべきディスクの番号。ディスクが修正されると、ブロックがそのディスクに戻されます。
temporary	integer	ブロックに、一時テーブルやクエリの中間結果などの一時データが含まれているかどうかを示します。0 = false、1 = true。デフォルトは false です。
newblock	integer	ブロックが新しい (true) かまたは一度もディスクにコミットされていない (false) かを示します。0 = false、1 = true。

サンプルクエリ

SVV_DISKUSAGE には割り当て済みディスクブロックにつき 1 つの行が含まれるため、すべての行を選択するクエリを実行すると非常に多数の行が返される可能性があります。SVV_DISKUSAGE を使用した集計クエリのみを使用することをお勧めします。

USERS テーブルの列 6 (EMAIL 列) に対してこれまでに割り当てられたブロックの最も大きな数を返します。

```
select db_id, trim(name) as tablename, max(blocknum)
from svv_diskusage
where name='users' and col=6
group by db_id, name;

db_id   | tablename | max
--------+-----------+-----
175857 | users     |   2
```

```
(1 row)
```

次のクエリは、SALESNEW という名前の 10 列の大型テーブルのすべての列と似た結果を返します。(列 10 から 12 に対応する最後の 3 行は、非表示のメタデータ列用です。)

```
select db_id, trim(name) as tablename, col, tbl, max(blocknum)
from svv_diskusage
where name='salesnew'
group by db_id, name, col, tbl
order by db_id, name, col, tbl;

db_id  | tablename  | col |  tbl   | max
--------+------------+-----+--------+-----
175857 | salesnew   |   0 | 187605 | 154
175857 | salesnew   |   1 | 187605 | 154
175857 | salesnew   |   2 | 187605 | 154
175857 | salesnew   |   3 | 187605 | 154
175857 | salesnew   |   4 | 187605 | 154
175857 | salesnew   |   5 | 187605 |  79
175857 | salesnew   |   6 | 187605 |  79
175857 | salesnew   |   7 | 187605 | 302
175857 | salesnew   |   8 | 187605 | 302
175857 | salesnew   |   9 | 187605 | 302
175857 | salesnew   |  10 | 187605 |   3
175857 | salesnew   |  11 | 187605 |   2
175857 | salesnew   |  12 | 187605 | 296
(13 rows)
```

SVV_EXTERNAL_COLUMNS

SVV_EXTERNAL_COLUMNS を使用して外部テーブルの列の詳細を表示します。また、クロスデータベースクエリにも SVV_EXTERNAL_COLUMNS を使用して、ユーザーがアクセスできる接続されていないデータベース上のテーブルですべての列の詳細を表示します。

SVV_EXTERNAL_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
redshift_database_name	text	ローカルのAmazon Redshift データベースの名前。
schemaname	text	外部テーブルの Amazon Redshift 外部スキーマの名前。
tablename	text	外部テーブルの名前。
columnname	text	列の名前。
external_type	text	列のデータ型。
columnnum	integer	1 から始まる外部列番号。
part_key	integer	列がパーティションキーの場合はキーの順序。列がパーティションでない場合、値は `0` です。
is_nullable	text	列が Null 許容型であるかどうかを定義します。一部の値は `true`、`false`、または情報がないことを表す空の文字列です。

SVV_EXTERNAL_DATABASES

SVV_EXTERNAL_DATABASES を使用して外部データベースの詳細を表示します。

SVV_EXTERNAL_DATABASES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「[システムテーブルとビューのデータの可視性](#)」を参照してください。

テーブルの列

列名	データ型	説明
eskind	integer	データベースの外部カタログのタイプ。**1** はデータカタログ、**2** は Hive メタストアを示します。
esoptions	text	データベースが存在するカタログの詳細。
databasename	text	外部カタログのデータベースの名前。
location	text	データベースの場所。
parameters	text	データベースのパラメータ。

SVV_EXTERNAL_PARTITIONS

SVV_EXTERNAL_PARTITIONS を使用して外部テーブルのパーティションの詳細を表示します。

SVV_EXTERNAL_PARTITIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
schemaname	text	指定のパーティションを持つ外部テーブルの Amazon Redshift 外部スキーマの名前。
tablename	text	外部テーブルの名前。
値	text	パーティションの値。

列名	データ型	説明
location	text	パーティションの場所。列サイズは 128 文字に制限されています。それより長い値は切り詰められます。
input_format	text	入力形式。
output_format	text	出力形式。
serialization_lib	text	シリアル化ライブラリ。
serde_parameters	text	SerDe パラメータ。
compressed	integer	パーティションが圧縮されているかどうかを示す値。**1** は圧縮されていることを、**0** は圧縮されていないことを示します。
parameters	text	パーティションのプロパティ。

SVV_EXTERNAL_SCHEMAS

SVV_EXTERNAL_SCHEMAS を使用して、外部スキーマの情報を表示します。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

SVV_EXTERNAL_SCHEMAS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
esoid	oid	外部スキーマ ID。
eskind	smallint	外部スキーマの外部カタログのタイプ: 1 はデータカタログ、2 は Hive メタストア、3 は Aurora PostgreSQL または Amazon RDS PostgreSQL への横串検索、4 はローカル Amazon Redshift データベースのスキーマ、5 はリモート Amazon Redshift データベースのスキーマ、6 はシステム

列名	データ型	説明
		テーブルのスキーマ、8 はリモート MySQL データベースのスキーマ、9 は Amazon Kinesis Data Streams のスキーマ、10 は Apache Kafka データストリーム向けの Amazon マネージドストリーミングを示します。
schemaname	name	外部スキーマ名。
esowner	integer	外部スキーマ所有者のユーザー ID です。
databasename	text	外部データベース名。
esoptions	text	外部スキーマのオプション。

例

以下の例に示しているのは、外部スキーマの詳細です。

```
select * from svv_external_schemas;

esoid  | eskind | schemaname | esowner | databasename | esoptions

-------+--------+------------+---------+--------------
+-------------------------------------------------------------
100133 |      1 | spectrum   |     100 | redshift     |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

SVV_EXTERNAL_TABLES

SVV_EXTERNAL_TABLES を使用して外部テーブルの詳細を表示します。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。データベース間クエリにも SVV_EXTERNAL_TABLES を使用して、ユーザーがアクセスできる接続されていないデータベース上のすべてのテーブルのメタデータを表示します。

SVV_EXTERNAL_TABLES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
redshift_database_name	text	ローカルのAmazon Redshift データベースの名前。
schemaname	text	外部テーブルの Amazon Redshift 外部スキーマの名前 。
tablename	text	外部テーブルの名前。
tabletype	text	テーブルの種類。一部の値は、TABLE、VIEW、MA TERIALIZED VIEW、または情報がないことを表す空の文字列 " " です。
location	text	テーブルの場所。
input_format	text	入力形式
output_format	text	出力形式。
serialization_lib	text	シリアル化ライブラリ。
serde_parameters	text	SerDe パラメータ。
compressed	integer	テーブルが圧縮されているかどうかを示す値。`1` は圧縮されていることを、`0` は圧縮されていないことを示します。
parameters	text	テーブルのプロパティ。

例

次の例は、svv_external_tables とフェデレーテッドクエリで述語に外部スキーマを使用した場合の詳細を示しています。

```
select schemaname, tablename from svv_external_tables where schemaname = 'apg_tpch';
schemaname  | tablename
------------+-----------
apg_tpch    | customer
apg_tpch    | lineitem
apg_tpch    | nation
apg_tpch    | orders
apg_tpch    | part
apg_tpch    | partsupp
apg_tpch    | region
apg_tpch    | supplier
(8 rows)
```

SVV_FUNCTION_PRIVILEGES

SVV_FUNCTION_PRIVILEGES を使用して、現在のデータベース内のユーザー、ロール、およびグループに明示的に付与されている関数のアクセス許可を表示します。

SVV_FUNCTION_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
namespace_name	text	指定された関数が存在する名前空間の名前。
function_name	text	関数の名前

列名	データ型	説明
argument_types	text	関数の入力引数の型を表す文字列。
privilege_type	text	アクセス許可の種類。指定できる値は EXECUTE です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_FUNCTION_PRIVILEGES の結果を示します。

```
SELECT
 namespace_name,function_name,argument_types,privilege_type,identity_name,identity_type,admin_o
 FROM svv_function_privileges
WHERE identity_name IN ('role1', 'reguser');

 namespace_name | function_name |       argument_types       | privilege_type |
 identity_name | identity_type | admin_option
----------------+---------------+----------------------------+----------------
+-----------------+---------------+--------------
    public      | test_func1    | integer                    |    EXECUTE     |
 role1      |     role      |  False
    public      | test_func2    | integer, character varying |    EXECUTE     |
 reguser     |     user      |  False
```

SVV_GEOGRAPHY_COLUMNS

データウェアハウス内の GEOGRAPHY 列のリストを表示するには、SVV_GEOGRAPHY_COLUMNS を使用してください。この列のリストには、データ共有の列が含まれます。

SVV_GEOGRAPHY_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
f_table_catalog	varCHAR(128)	GEOGRAPHY 列を含むテーブルが存在するデータベースの名前。
f_table_schema	varCHAR(128)	GEOGRAPHY 列を含むテーブルが存在するスキーマの名前。
f_table_name	varCHAR(128)	GEOGRAPHY 列が存在するテーブルの名前。
f_geography_column	varCHAR(128)	GEOGRAPHY 列の名前。
coord_dimension	integer	GEOGRAPHY データのディメンションの数。
srid	integer	GEOGRAPHY データの空間リファレンスシステム識別子 (SRID)。
type	varCHAR(128)	空間地理データ型名。

サンプルクエリ

次の例では、SVV_GEOGRAPHY_COLUMNS の結果を示します。

```
SELECT * FROM svv_geography_columns;

f_table_catalog   | f_table_schema   | f_table_name   | f_geography_column   |
 coord_dimension | srid |   type
-----------------+-----------------+---------------+---------------------
+-----------------+------+--------------
dev               | public           | spatial_test   | test_geography       | 2
    | 0     | GEOGRAPHY
```

SVV_GEOMETRY_COLUMNS

データウェアハウス内の GEOMETRY 列のリストを表示するには、SVV_GEOMETRY_COLUMNS を使用してください。この列のリストには、データ共有の列が含まれます。

SVV_GEOMETRY_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
f_table_catalog	varCHAR(128)	列を含むテーブルが存在するデータベースの名前。
f_table_schema	varCHAR(128)	GEOMETRY 列を含むテーブルが存在するスキーマの名前。
f_table_name	varCHAR(128)	GEOMETRY 列が存在するテーブルの名前。
f_geometry_column	varCHAR(128)	GEOMETRY 列の名前。
coord_dimension	integer	GEOMETRY データのディメンションの数。
srid	integer	GEOMETRY 列の空間リファレンスシステム識別子 (SRID)。

列名	データ型	説明
type	varCHAR(128)	空間ジオメトリタイプ名。

サンプルクエリ

次の例では、SVV_GEOMETRY_COLUMNS の結果を示します。

```
SELECT * FROM svv_geometry_columns;

f_table_catalog  | f_table_schema  | f_table_name  | f_geometry_column   |
 coord_dimension | srid |  type
-----------------+-----------------+---------------+--------------------
+-----------------+------+--------------
dev              | public          | accomodations | shape               | 2
    | 0    | GEOMETRY
dev              | public          | zipcode       | wkb_geometry        | 2
    | 0    | GEOMETRY
```

SVV_IAM_PRIVILEGES

SVV_IAM_PRIVILEGES を使用して、ユーザー、ロール、およびグループに明示的に付与されている IAM 権限を確認します。

SVV_IAM_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可されたエントリのみを見ることができます。

テーブルの列

列名	データ型	説明
iam_arn	text	名前空間の名前。

列名	データ型	説明
command_type	text	権限タイプ。指定できる値は、COPY、UNLOAD、CREATE MODEL、または EXTERNAL FUNCTION です。
identity_id	integer	アイデンティティ ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティ名。
identity_type	text	アイデンティティのタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。

サンプルクエリ

次の例は、SVV_IAM_PRIVILEGES の結果を示しています。

```
SELECT * from SVV_IAM_PRIVILEGES ORDER BY IDENTITY_ID;
       iam_arn        | command_type | identity_id | identity_name | identity_type
----------------------+--------------+-------------+---------------+---------------
 default-aws-iam-role | COPY         |           0 | public        | public
 default-aws-iam-role | UNLOAD       |           0 | public        | public
 default-aws-iam-role | CREATE MODEL |           0 | public        | public
 default-aws-iam-role | EXFUNC       |           0 | public        | public
 default-aws-iam-role | COPY         |         106 | u1            | user
 default-aws-iam-role | UNLOAD       |         106 | u1            | user
 default-aws-iam-role | CREATE MODEL |         106 | u1            | user
 default-aws-iam-role | EXFUNC       |         106 | u1            | user
 default-aws-iam-role | COPY         |      118413 | r1            | role
 default-aws-iam-role | UNLOAD       |      118413 | r1            | role
 default-aws-iam-role | CREATE MODEL |      118413 | r1            | role
 default-aws-iam-role | EXFUNC       |      118413 | r1            | role
(12 rows)
```

SVV_IDENTITY_PROVIDERS

SVV_IDENTITY_PROVIDERS ビューは、ID プロバイダーの名前と追加のプロパティを返します。ID プロバイダーを作成する方法の詳細については、「ID プロバイダーを作成する」を参照してください。

SVV_IDENTITY_PROVIDERS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
uid	integer	登録された ID プロバイダーの一意の ID。
name	text	ID プロバイダー名。
type	text	ID プロバイダーのタイプ。
instanceid	text	同じタイプのインスタンス間での一意の差別化要因。
namespc	text	ID プロバイダーの名前空間プレフィックス。
params	text	ID プロバイダーのパラメータを持つ JSON オブジェクト。
有効	ブール	ID プロバイダーが有効かどうかを示す。

サンプルクエリ

ID プロバイダーのプロパティを表示するには、ID プロバイダーを作成した後に、次のようなクエリを実行します。

```
SELECT name, type, instanceid, namespc, params, enabled
FROM svv_identity_providers
```

```
ORDER BY 1;
```

サンプル出力には、パラメータの説明が含まれています。

```
       name        | type  |                 instanceid                 | namespc |
                                                   params

                        | enabled
------------------+-------+--------------------------------------+---------
+-------------------------------------------------------------------------------------------------------
+---------
 rs5517_azure_idp | azure | e40d4bb2-7670-44ae-bfb8-5db013221d73 | abc     |
 {"issuer":"https://login.microsoftonline.com/e40d4bb2-7670-44ae-bfb8-5db013221d73/
v2.0", "client_id":"871c010f-5e61-4fb1-83ac-98610a7e9110", "client_secret":,
 "audience":["https://analysis.windows.net/powerbi/connector/AmazonRedshift", "https://
analysis.windows.net/powerbi/connector/AWSRDS"]} | t
(1 row)
```

SVV_INTEGRATION

SVV_INTEGRATION は、統合の設定に関する詳細を表示します。

SVV_INTEGRATION はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

ゼロ ETL 統合の詳細については、「Working with zero-ETL integrations」を参照してください。

テーブルの列

列名	データ型	説明
integration_id	character (128)	統合に関連付けられている識別子です。
target_database	character (128)	統合データを受け取る Amazon Redshift のデータベース。
ソース	character (128)	統合のソースデータ。可能なデータ型には、 MySQL と PostgreSQL が含まれます。

列名	データ型	説明
state	character (128)	統合の状態。指定できる値には、PendingDbConnectState 、SchemaDiscoveryState 、CdcRefreshState 、および ErrorState などがあります。
current_lag	bigint	統合のソースとターゲットの間の現在のラグタイム (ミリ秒)。
last_replicated_checkpoint	character (128)	最後にレプリケートされたチェックポイント。
total_tables_replicated	integer	現在レプリケートされた状態にあるテーブルの総数。
total_tables_failed	integer	現在、失敗状態にあるテーブルの総数。
creation_time	timestamp	統合が作成された時刻 (UTC)。これは、ターゲットデータベースが統合から作成された時刻として定義されます。
refresh_interval	integer	ゼロ ETL ソースからターゲットデータベースにデータを更新する、おおよその時間間隔 (秒単位)。
source_database	character (128)	ソースデータベースの名前。

サンプルクエリ

次の SQL コマンドは、現在定義されている統合を表示します。

```
select * from svv_integration;

           integration_id               | target_database | source |      state
   | current_lag |      last_replicated_checkpoint      | total_tables_replicated |
 total_tables_failed |       creation_time        |  refresh_interval  | source_database
--------------------------------------+-----------------+--------+------------------
+-------------+-------------------------------------+------------------------
```

```
+---------------------+---------------------------+--------------------
+----------------
  99108e72-1cfd-414f-8cc0-0216acefac77 |     perfdb      |  MySQL | CdcRefreshState |
  56606106   | {"txn_seq":9834,"txn_id":126597515} |         152          |
 0         | 2023-09-19 21:05:27.520299|      720           + mysourceetl
```

SVV_INTEGRATION_TABLE_STATE

SVV_INTEGRATION_TABLE_STATE は、テーブルレベルの統合情報に関する詳細を表示します。

SVV_INTEGRATION_TABLE_STATE は、スーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

詳細については、「Working with zero-ETL integrations」を参照してください。

テーブルの列

列名	データ型	説明
integration_id	character(128)	統合に関連付けられている識別子です。
target_database	character(128)	Amazon Redshift データベースの名前。
schema_name	character(128)	Amazon Redshift スキーマの名前。
table_name	character(128)	テーブルの名前。
table_state	character(128)	テーブルの状態。使用できる値は Synced、Failed、Deleted、ResyncRequired および ResyncInitiated です。
table_last_replicated_checkpoint	character(128)	現在同期されているログ座標。
理由	character(256)	最後の状態遷移の理由。一般的な理由としては、テーブル内のデータ型がサポートされていないことや、テーブルにプライマリキーがないことが考えられます。一般的な問題のトラブルシューティング方法の詳細について

列名	データ型	説明
		は、「Troubleshooting zero-ETL integrations in Amazon Redshift」を参照してください。
last_updated_timestamp	タイムゾーンなしのタイムスタンプ	テーブルが最後に更新された時刻 (UTC)。

サンプルクエリ

次の SQL コマンドは、統合のログを表示します。

```
select * from svv_integration_table_state;

          integration_id               | target_database | schema_name |      table_name
   | Table_state  |table_last_replicated_checkpoint | reason | last_updated_timestamp
--------------------------------------+-----------------+-------------
+-------------------+--------------+----------------------------------
+--------------------------------------
 4798e675-8f9f-4686-b05f-92c538e19629 |  sample_test2    |    sample    |
 SampleTestChannel |    Synced     |   {"txn_seq":3,"txn_id":3122}    |         |
 2023-05-12 12:40:30.656625
```

SVV_INTERLEAVED_COLUMNS

インターリーブソートキーを使用するテーブルで、VACUUM REINDEX を使用してインデックスを再作成するかどうか決定するために、SVV_INTERLEAVED_COLUMNS ビューを使用します。VACUUM の実行頻度および VACUUM REINDEX の実行時期についての詳細は、「バキューム処理時間の最小化」を参照してください。

SVV_INTERLEAVED_COLUMNS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
tbl	integer	テーブル ID。

列名	データ型	説明
col	integer	列のゼロベースインデックス。
interleaved_skew	numeric 9,2)	テーブルのインターリーブソートキー列にある分散の量を示す比率。値 1.00 は分散がないことを示し、それ以上の値は、値が大きくなるほど大きな分散があることを示します。大きな分散のあるテーブルでは、VACUUM REINDEX コマンドでインデックスを再作成する必要があります。
last_reindex	timestar	指定されたテーブルに対して VACUUM REINDEX が実行された最終日時。テーブルのインデックスが一度も再作成されていない場合、または基礎となるシステムログテーブル STL_VACUUM が最後のインデックス再作成以降にローテーションされている場合、この値は NULL です。

サンプルクエリ

インデックスの再作成が必要となる可能性があるテーブルを識別するには、以下のクエリを実行します。

```
select tbl as tbl_id, stv_tbl_perm.name as table_name,
col, interleaved_skew, last_reindex
from svv_interleaved_columns, stv_tbl_perm
where svv_interleaved_columns.tbl = stv_tbl_perm.id
and interleaved_skew is not null;

 tbl_id | table_name | col | interleaved_skew | last_reindex
--------+------------+-----+------------------+--------------------
 100068 | lineorder  |   0 |             3.65 | 2015-04-22 22:05:45
 100068 | lineorder  |   1 |             2.65 | 2015-04-22 22:05:45
 100072 | customer   |   0 |             1.65 | 2015-04-22 22:05:45
 100072 | lineorder  |   1 |             1.00 | 2015-04-22 22:05:45
(4 rows)
```

SVV_LANGUAGE_PRIVILEGES

SVV_LANGUAGE_PRIVILEGES を使用して、現在のデータベース内のユーザー、ロール、およびグループに明示的に付与されている言語のアクセス許可を表示します。

SVV_LANGUAGE_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
language_name	text	言語の名前。
privilege_type	text	アクセス許可の種類。指定できる値は USAGE です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_LANGUAGE_PRIVILEGES の結果を示します。

```
SELECT language_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_language_privileges
WHERE identity_name IN ('role1', 'reguser');

 language_name | privilege_type | identity_name | identity_type | admin_option
---------------+----------------+---------------+---------------+---------------
    exfunc     |      USAGE     |    reguser    |      user     |     False
    exfunc     |      USAGE     |     role1     |      role     |     False
```

```
    plpythonu    |      USAGE      |    reguser    |      user       |     False
```

SVV_MASKING_POLICY

SVV_MASKING_POLICY を使用して、クラスターで作成されたすべてのマスキングポリシーを表示します。

SVV_MASKING_POLICY を閲覧できるのは、スーパーユーザーおよび `sys:secadmin` ロールを持つユーザーだけです。通常のユーザーには 0 行が表示されます。

テーブルの列

列名	データ型	説明
policy_database	text	マスキングポリシーが作成されたデータベースの名前。
policy_name	text	マスキングポリシーの名前。
input_columns	text	CREATE POLICY ステートメントの WITH 句に指定されている属性。
policy_expression	text	ポリシーで使用されるマスキング表現。
policy_modified_by	text	ポリシーを最後に変更したユーザーの名前。
policy_modified_time	timestamp	ポリシーが作成または最後に変更されたときのタイムスタンプ。

SVV_ML_MODEL_INFO

機械学習モデルの現在の状態に関する情報です。

SVV_ML_MODEL_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	char(128)	モデルのデータベース。
schema_name	char(128)	モデルのスキーマ。
user_name	char(128)	モデルの所有者。
model_name	char(128)	モデルの名前です。
life_cycle	char(20)	モデルのライフサイクルステータス。
is_refreshable	integer	トレーニングクエリの元のテーブルと列がまだ存在し、ユーザーがそれらに対するアクセス許可を持っている場合、それが更新可能かどうかについてのモデルの状態。指定できる値は 1 (更新可能) と 0 (更新不可) です。
model_state	char(128)	モデルの現在の状態。

サンプルクエリ

次のクエリでは、機械学習モデルの現在の状態を表示します。

```
SELECT schema_name, model_name, model_state
FROM svv_ml_model_info;

 schema_name |         model_name           |             model_state
-------------+------------------------------+--------------------------------------
 public      | customer_churn_auto_model    | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

SVV_ML_MODEL_PRIVILEGES

SVV_ML_MODEL_PRIVILEGES を使用して、クラスター内のユーザー、ロール、およびグループに明示的に付与された機械学習モデルのアクセス許可を表示します。

SVV_ML_MODEL_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
namespace_name	text	指定された機械学習モデルが存在する名前空間の名前。
model_name	text	機械学習モデルの名前。
model_version	integer	モデルのバージョン番号。
privilege_type	text	アクセス許可の種類。指定できる値は EXECUTE です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかど

列名	データ型	説明
		うかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_ML_MODEL_PRIVILEGES の結果を示します。

```
SELECT
 namespace_name,model_name,model_version,privilege_type,identity_name,identity_type,admin_optio
 FROM svv_ml_model_privileges
WHERE model_name = 'test_model';

 namespace_name | model_name | model_version | privilege_type |  identity_name |
 identity_type | admin_option
----------------+------------+---------------+----------------+----------------
+---------------+--------------
      public    | test_model |       1       |    EXECUTE     |     reguser    |
 user       |     False
      public    | test_model |       1       |    EXECUTE     |     role1      |
 role       |     False
```

SVV_MV_DEPENDENCY

SVV_MV_DEPENDENCY テーブルには、Amazon Redshift 内の、異なるマテリアライズドビュー間に存在する依存関係が表示されます。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

SVV_MV_DEPENDENCY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	char(128)	特定のマテリアライズドビューを含むデータベース。
schema_name	char(128)	マテリアライズドビューのスキーマ。
name	char(128)	マテリアライズドビューの名前。
dependent_database_name	char(128)	このマテリアライズドビューが依存するマテリアライズドビューのデータベース。
dependent_schema_name	char(128)	このマテリアライズドビューが依存するマテリアライズドビューのスキーマ。
dependent_name	char(128)	このマテリアライズドビューが依存するマテリアライズドビューの名前。

サンプルクエリ

次のクエリは、マテリアライズドビュー mv_over_foo がその定義内の依存関係として、マテリアライズドビュー mv_foo を使用していることを示す行を出力に返します。

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM svv_mv_dependency;

 database_name | schema_name          | name        | dependent_database_name |
 dependent_schema_name      | dependent_name
---------------+----------------------+-------------+-------------------------
+--------------------------+----------
 dev           | test_ivm_setup       | mv_over_foo |                     dev |
 test_ivm_setup             | mv_foo
```

SVV_MV_INFO

SVV_MV_INFO テーブルには、すべてのマテリアライズドビューの行、データが古くなっているかどうか、およびステータス情報が含まれます。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

SVV_MV_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	char(128)	マテリアライズドビューを含むデータベース。
schema_name	char(128)	データベースのスキーマ。
user_name	char(128)	マテリアライズドビューを所有するユーザー。
name	char(128)	マテリアライズドビューの名前。
is_stale	char(1)	t は、マテリアライズドビューが古くなっていることを示します。古いマテリアライズドビューとは、ベーステーブルは更新されているが、マテリアライズドビューは更新されていないビューのことを指します。前回の再起動以降に更新が実行されていない場合、この情報は正確ではない可能性があります。
state	integer	マテリアライズドビューの状態。 • 0 – マテリアライズドビューは、更新時に完全に再計算されます。 • 1 – マテリアライズドビューは増分です。 • 101 – 削除された列があるため、マテリアライズドビューを更新できません。この制約は、列

列名	データ型	説明
		がマテリアライズドビューで使用されていない場合でも適用されます。 • 102 – 列のタイプが変更されたため、マテリアライズドビューを更新できません。この制約は、列がマテリアライズドビューで使用されていない場合でも適用されます。 • 103 – テーブル名が変更されたため、マテリアライズドビューを更新できません。 • 104 – 列の名前が変更されたため、マテリアライズドビューを更新できません。この制約は、列がマテリアライズドビューで使用されていない場合でも適用されます。 • 105 – スキーマ名が変更されたため、マテリアライズドビューを更新できません。
autorewrite	char(1)	t は、マテリアライズドビューがクエリの自動書き換えに適格であることを示します。
autorefresh	char(1)	t は、マテリアライズドビューを自動的に更新できることを示します。

サンプルクエリ

すべてのマテリアライズドビューのステータスを表示するには、次のクエリを実行します。

```
select * from svv_mv_info;
```

このクエリは次のサンプル出力を返します。

```

database_name |       schema_name        | user_name |   name   |  is_stale | state |
 autorefresh | autorewrite
--------------+--------------------------+-----------+---------+-----------+-------
+-------------+----------------
 dev          | test_ivm_setup           | catch-22  | mv       |   f       |     1 |
       1 |           0
```

```
dev            | test_ivm_setup             | lotr         | old_mv  |    t         |       1 |
       0 |              1
```

SVV_QUERY_INFLIGHT

SVV_QUERY_INFLIGHT ビューを使用してデータベースで現在実行されているクエリを確認します。このビューは STV_INFLIGHT を STL_QUERYTEXT に結合します。SVV_QUERY_INFLIGHT はリーダーノードのみのクエリは表示しません。詳細については、「リーダーノード専用関数」を参照してください。

SVV_QUERY_INFLIGHT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
slice	integer	クエリが実行されているスライス。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
pid	integer	プロセス ID。セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、この値は一定です。この列を使用して、STL_ERROR テーブルに結合できます。
starttime	timestamp	クエリが開始された時刻。

列名	データ型	説明
suspended	integer	クエリが中断されているかどうかを示します。`0` =false で、**1** = true です。
text	character(200)	200 文字刻みのクエリのテキスト。
sequence	integer	クエリステートメントのセグメントのシーケンス番号。

サンプルクエリ

以下のサンプル出力では、現在実行中の 2 つのクエリ (SVV_QUERY_INFLIGHT クエリそれ自体、およびテーブルの 3 つの行に分割するクエリ 428) を示します。(starttime 列および statement 列は、この例の出力では切り捨てられています。)

```
select slice, query, pid, starttime, suspended, trim(text) as statement, sequence
from svv_query_inflight
order by query, sequence;

slice|query| pid  |      starttime       |suspended| statement | sequence
-----+-----+------+----------------------+---------+-----------+---------
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | select ...|    0
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | enueid ...|    1
1012 | 428 | 1658 | 2012-04-10 13:53:... |       0 | atname,...|    2
1012 | 429 | 1608 | 2012-04-10 13:53:... |       0 | select ...|    0
(4 rows)
```

SVV_QUERY_STATE

SVV_QUERY_STATE を使用して、現在実行されているクエリのランタイムについての情報を表示します。

SVV_QUERY_STATE ビューには STV_EXEC_STATE テーブルのデータのサブセットが含まれます。

SVV_QUERY_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

> **Note**
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
seg	integer	実行されているクエリセグメントの数。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。
step	integer	実行されているクエリステップの数。ステップとは、クエリのランタイムの最小単位です。各ステップはテーブルのスキャン、結果の返却、データのソートなど、作業の個別の単位を表します。
maxtime	interval	このステップを実行する最大時間 (マイクロ秒)。
avgtime	interval	このステップを実行する平均時間 (マイクロ秒)。
rows	bigint	実行中のステップが生成した行の数。
bytes	bigint	実行中のステップが生成したバイト数。
cpu	bigint	内部使用を目的とします。
memory	bigint	内部使用を目的とします。

列名	データ型	説明
rate_row	double precision	クエリが開始されてからの行毎秒率。行を合計し、クエリが開始されてから現在時刻までの秒数で除算することで算出されます。
rate_byte	double precision	クエリが開始されてからのバイト毎秒率。バイトを合計し、クエリが開始されてから現在時刻までの秒数で除算することで算出されます。
label	character(25)	クエリラベル: ステップの名前 (`scan`または`sort`など)。
is_diskbased	character(1)	クエリのこのステップがディスクベースの処理として実行されているかどうか: true (**t**) または false (**f**)。ハッシュ、ソート、集計といった特定のステップのみがディスクベースで実行できます。ステップの多くのタイプは、常にメモリ内で実行されます。
workmem	bigint	クエリステップに割り当てられた作業メモリの量 (バイト単位)。
num_parts	integer	ハッシュテーブルがハッシュステップ中に分割されるパーティションの数。この列にある正の数は、ハッシュステップがディスクベースのオペレーションとして実行されることを示すものではありません。ハッシュステップがディスクベースであったかどうかを調べるには、IS_DISKBASED 列の値を見てください。
is_rrscan	character(1)	true (**t**) の場合は、ステップで範囲限定スキャンが使用されたことを示します。デフォルトは false (**f**) です。
is_delayed_scan	character(1)	true (**t**) の場合は、ステップで遅延スキャンが使用されたことを示します。デフォルトは false (**f**) です。

サンプルクエリ

クエリの処理時間をステップごとに算出する

以下のクエリは、クエリ ID が 279 のクエリの各ステップを実行するためにかかった時間と、Amazon Redshift が処理したデータ行の数を表示します。

```
select query, seg, step, maxtime, avgtime, rows, label
from svv_query_state
```

```
where query = 279
order by query, seg, step;
```

このクエリは次のサンプル出力で示されているように、クエリ 279 についての処理情報を取得します。

```
query |   seg   | step | maxtime | avgtime |  rows   | label
------+---------+------+---------+---------+---------+-------------------
  279 |       3 |    0 | 1658054 | 1645711 | 1405360 | scan
  279 |       3 |    1 | 1658072 | 1645809 |       0 | project
  279 |       3 |    2 | 1658074 | 1645812 | 1405434 | insert
  279 |       3 |    3 | 1658080 | 1645816 | 1405437 | distribute
  279 |       4 |    0 | 1677443 | 1666189 | 1268431 | scan
  279 |       4 |    1 | 1677446 | 1666192 | 1268434 | insert
  279 |       4 |    2 | 1677451 | 1666195 |       0 | aggr
(7 rows)
```

ディスクで現在実行中のアクティブなクエリを確認する

次のクエリは、ディスクで現在実行中のアクティブなクエリを示します。

```
select query, label, is_diskbased from svv_query_state
where is_diskbased = 't';
```

このサンプル出力は、ディスクで現在実行中のアクティブなクエリを示します。

```
 query | label        | is_diskbased
-------+--------------+--------------
1025   | hash tbl=142 |      t
(1 row)
```

SVV_REDSHIFT_COLUMNS

SVV_REDSHIFT_COLUMNS を使用して、ユーザーがアクセスできるすべての列リストを表示します。この一連の列には、クラスター上の列と、リモートクラスターによって提供されるデータ共有の列が含まれます。

SVV_REDSHIFT_COLUMNS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	列を含むテーブルが存在するデータベースの名前。
schema_name	varCHAR(128)	テーブルのスキーマの名前。
table_name	varCHAR(128)	テーブルの名前。
column_name	varCHAR(128)	列の名前。
ordinal_position	integer	テーブルの列の位置。
data_type	varCHAR(32)	列のデータ型。
column_default	varCHAR(4000)	列のデフォルト値。
is_nullable	varCHAR(3)	列が null であるかどうかを定義する値。可能な値は yes、no、および " " (情報がないことを表す空の文字列) です。
encoding	varCHAR(128)	列のエンコード型。
distkey	boolean	この列がテーブルのディストリビューションキーである場合は true、それ以外の場合は false の値です。
sortkey	integer	ソートキー内で列の順序を指定する値。 テーブルが複合ソートキーを使用する場合、ソートキーに含まれるすべての列は、ソートキー内の列の位置を示す正の値を持ちます。

列名	データ型	説明
		テーブルがインターリーブソートキーを使用する場合、ソートキーに含まれる各列は正または負の値を交互に持ちます。ここで、絶対値は、ソートキー内にある列の位置を示します。 sortkey が 0 の場合、列はソートキーに含まれません。
column_acl	varCHAR(128)	列に指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。
解説	varCHAR(256)	解説。

サンプルクエリ

次の例では、SVV_REDSHIFT_COLUMNS の出力を返します。

```
SELECT *
FROM svv_redshift_columns
WHERE database_name = 'tickit_db'
    AND TABLE_NAME = 'tickit_sales_redshift'
ORDER BY COLUMN_NAME,
    TABLE_NAME,
    database_name
LIMIT 5;

database_name | schema_name |       table_name       | column_name | ordinal_position |
 data_type | column_default | is_nullable | encoding | distkey | sortkey | column_acl
 | remarks
--------------+-------------+-----------------------+-------------+------------------
+-----------+----------------+-------------+----------+---------+---------
+-------------+--------
   tickit_db   |    public    | tickit_sales_redshift |    buyerid    |         4          |
 integer  |                 |      NO      |    az64    |  False  |    0    |              |
```

```
   tickit_db   |   public     | tickit_sales_redshift |  commission |        9         |
 numeric   |      (8,2)      |      YES      |     az64    |  False  |    0    |           |
   tickit_db   |   public     | tickit_sales_redshift |    dateid    |        6         |
 smallint  |                 |      NO       |     none    |  False  |    1    |           |
   tickit_db   |   public     | tickit_sales_redshift |   eventid   |        5         |
 integer   |                 |      NO       |     az64    |  False  |    0    |       |
   tickit_db   |   public     | tickit_sales_redshift |   listid    |        2         |
 integer   |                 |      NO       |     az64    |  True   |    0    |           |
```

SVV_REDSHIFT_DATABASES

SVV_ REDSHIFT_DATABASES を使用して、ユーザーがアクセスできるすべてのデータベースリストを表示します。これには、クラスター上のデータベースと、リモートクラスターによって提供されるデータ共有から作成されたデータベースが含まれます。

SVV_REDSHIFT_DATABASES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	データベースの名前。
database_owner	integer	データベース所有者のユーザー ID。
database_type	varCHAR(32)	データベースのタイプ。指定できるタイプは、ローカルデータベースまたは共有データベースです。
database_acl	varCHAR(128)	この情報は、内部使用に限定されています。
database_options	varCHAR(128)	データベースのプロパティ。
database_isolation_level	varCHAR(128)	データベースの分離レベル。可能値には Snapshot

列名	データ型	説明
		Isolation や Serializable などがあります。

サンプルクエリ

次の例では、SVV_REDSHIFT_DATABASES の出力を返します。

```
select database_name, database_owner, database_type, database_options,
 database_isolation_level
from  svv_redshift_databases;


database_name | database_owner | database_type | database_options |
 database_isolation_level
--------------+----------------+---------------+------------------+------------------
   dev        |  1             | local         | NULL             | Serializable
```

SVV_REDSHIFT_FUNCTIONS

SVV_REDSHIFT_FUNCTIONS を使用して、ユーザーがアクセスできるすべての関数リストを表示します。この一連の関数には、クラスター上の関数と、リモートクラスターによって提供されるデータ共有の関数が含まれます。

SVV_REDSHIFT_FUNCTIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	これらの関数を持つクラスターが存在するデータベースの名前。
schema_name	varCHAR(128)	特定の関数を指定するスキーマの名前。

列名	データ型	説明
function_name	varCHAR(128)	指定された関数の名前。
function_type	varCHAR(128)	関数のタイプ。使用できる値は、通常の関数、集計関数、およびストアドプロシージャです。
argument_type	varCHAR(512)	関数の入力引数の型を表す文字列。
result_type	varCHAR(128)	関数の戻り値のデータ型。

サンプルクエリ

次の例では、SVV_REDSHIFT_FUNCTIONS の出力を返します。

```
SELECT *
FROM svv_redshift_functions
WHERE database_name = 'tickit_db'
    AND SCHEMA_NAME = 'public'
ORDER BY function_name
LIMIT 5;

database_name | schema_name |      function_name     |  function_type    |
 argument_type   | result_type
--------------+-------------+-----------------------+------------------
+------------------+-------------
   tickit_db   |    public   |     shared_function    | REGULAR FUNCTION | integer,
 integer |   integer
```

SVV_REDSHIFT_SCHEMA_QUOTA

データベースの各スキーマのクォータと現在のディスク使用量を表示します。

SVV_REDSHIFT_SCHEMA_QUOTA はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このビューは、プロビジョニングされたクラスターまたは Redshift Serverless ワークグループをクエリするときに使用できます。

テーブルの列

列名	データ型	説明
database_name	character(128)	スキーマを含むデータベース。
schema_name	character(128)	スキーマの名前。
schema_owner	integer	スキーマの所有者の内部ユーザー ID。
クォータ	integer	スキーマが使用できるディスク容量 (MB)。
disk_usage	integer	スキーマによって現在使用されているディスク容量 (MB)。

サンプルクエリ

次の例では、`sales_schema` という名前のスキーマのクォータと現在のディスク使用量を表示します。

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage FROM
 svv_redshift_schema_quota
WHERE SCHEMA_NAME = 'sales_schema';


schema_name    | quota | disk_usage
--------------+-------+------------
sales_schema   | 2048  | 30
```

SVV_REDSHIFT_SCHEMAS

SVV_REDSHIFT_SCHEMAS を使用して、ユーザーがアクセスできるすべてのスキーマリストを表示します。この一連のスキーマには、クラスター上のスキーマと、リモートクラスターによって提供されるデータ共有のスキーマが含まれます。

SVV_REDSHIFT_SCHEMAS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	指定されたスキーマが存在するデータベースの名前。
schema_name	varCHAR(128)	名前空間またはスキーマの名前。
schema_owner	integer	スキーマの所有者の内部ユーザー ID。
schema_type	varCHAR(16)	スキーマのタイプ。指定できる値は、共有スキーマとローカルスキーマです。
schema_acl	varCHAR(128)	スキーマに指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。
schema_option	varCHAR(128)	スキーマのオプション。

サンプルクエリ

次の例では、SVV_REDSHIFT_SCHEMAS の出力を返します。

```
SELECT *
```

```
FROM svv_redshift_schemas
WHERE database_name = 'tickit_db'
ORDER BY database_name,
    SCHEMA_NAME;

database_name |     schema_name      | schema_owner | schema_type | schema_acl |
 schema_option
--------------+--------------------+-------------+------------+------------
+---------------
   tickit_db  |       public       |      1      |    shared   |            |
```

SVV_REDSHIFT_TABLES

SVV_REDSHIFT_TABLES を使用して、ユーザーがアクセスできるすべてのテーブルリストを表示します。この一連のテーブルには、クラスター上のテーブルと、リモートクラスターによって提供されるデータ共有のテーブルが含まれます。

SVV_REDSHIFT_TABLES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
database_name	varCHAR(128)	指定されたテーブルが存在するデータベースの名前。
schema_name	varCHAR(128)	テーブルのスキーマの名前。
table_name	varCHAR(128)	テーブルの名前。
table_type	varCHAR(128)	テーブルの種類。指定できる値は、ビューとテーブルです。
table_acl	varCHAR(128)	テーブルに指定されたユーザーまたはユーザーグループのアクセス許可を定義する文字列。

列名	データ型	説明
解説	varCHAR(128)	解説。
table_owner	varCHAR(128)	テーブルの所有者。

サンプルクエリ

次の例では、SVV_REDSHIFT_TABLES の出力を返します。

```
SELECT *
FROM svv_redshift_tables
WHERE database_name = 'tickit_db' AND TABLE_NAME LIKE 'tickit_%'
ORDER BY database_name,
TABLE_NAME;

database_name | schema_name |          table_name        | table_type | table_acl |
 remarks | table_owner
--------------+-------------+---------------------------+------------+-----------
+---------+-----------
   tickit_db  |    public   | tickit_category_redshift |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_date_redshift   |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_event_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |  tickit_listing_redshift |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_sales_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_users_redshift  |    TABLE   |           |
   +
   tickit_db  |    public   |   tickit_venue_redshift  |    TABLE   |           |
```

table_acl の値が NULL の場合、対応するテーブルにはアクセス権限が明示的に付与されません。

SVV_RELATION_PRIVILEGES

SVV_RELATION_PRIVILEGES を使用して、現在のデータベースのユーザー、ロール、グループに明示的に付与されているリレーション (テーブルおよびビュー) のアクセス許可を表示します。

SVV_RELATION_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- SYSLOG ACCESS UNRESTRICTED アクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。データの可視性の詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
namespace_name	text	指定されたリレーションが存在する名前空間の名前。
relation_name	text	リレーションの名前。
privilege_type	text	アクセス許可の種類。指定できる値は、INSERT、SELECT、UPDATE、DELETE、REFERENCES、または DROP です。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

サンプルクエリ

次の例では、SVV_RELATION_PRIVILEGES の結果を示します。

```
SELECT
 namespace_name,relation_name,privilege_type,identity_name,identity_type,admin_option
 FROM svv_relation_privileges
WHERE relation_name = 'orders' AND privilege_type = 'SELECT';

 namespace_name | relation_name | privilege_type |  identity_name | identity_type |
 admin_option
----------------+---------------+----------------+----------------+---------------
+--------------
     public     |    orders     |     SELECT     |    reguser     |     user      |
 False
     public     |    orders     |     SELECT     |     role1      |     role      |
 False
```

SVV_RLS_APPLIED_POLICY

SVV_RLS_APPLIED_POLICY を使用して、RLS で保護された関係を参照するクエリに対する RLS ポリシーの適用をトレースします。

SVV_RLS_APPLIED_POLICY は以下のユーザーに表示されます。

- スーパーユーザー
- `sys:operator` ロールを持つユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

sys:secadmin にはこのシステム許可が付与されていないことに注意してください。

テーブルの列

列名	データ型	説明
username	text	クエリを実行したユーザーの名前。
query	integer	クエリの ID。
xid	long	トランザクションのコンテキスト。
pid	integer	クエリを実行しているリーダープロセス。
recordtime	時系	クエリが記録された時間。

列名	データ型	説明
コマンド	char(1)	RLS ポリシーが適用されたコマンド。使用可能な値は次のとおりです。k は不明、s は選択、u は更新、i は挿入、y はユーティリティ、d は削除。
datname	text	行レベルのセキュリティポリシーがアタッチされている関係のデータベースの名前。
relschema	text	行レベルのセキュリティポリシーがアタッチされている関係のスキーマの名前。
relname	text	行レベルのセキュリティポリシーがアタッチされている関係の名前。
polname	text	関係にアタッチされている行レベルのセキュリティポリシー名。
poldefault	char(1)	関係にアタッチされている行レベルのセキュリティポリシーのデフォルト設定。使用可能な値は、デフォルトの false ポリシーが適用されている場合は false の f、デフォルトの true ポリシーが適用されている場合は true の t です。

サンプルクエリ

次の例では、SVV_RLS_APPLIED_POLICY の結果を示します。SVV_RLS_APPLIED_POLICY を照会するには、ACCESS SYSTEM TABLE 許可が必要です。

```
-- Check what RLS policies were applied to the run query.
SELECT username, command, datname, relschema, relname, polname, poldefault
FROM svv_rls_applied_policy
WHERE datname = CURRENT_DATABASE() AND query = PG_LAST_QUERY_ID();

 username | command |  datname  | relschema |          relname          |      polname
   | poldefault
----------+---------+-----------+-----------+---------------------------
+-----------------+------------
   molly  |    s    | tickit_db |   public  | tickit_category_redshift |
 policy_concerts |
```

SVV_RLS_ATTACHED_POLICY

SVV_RLS_ATTACHED_POLICY を使用して、現在接続されているデータベースに 1 つ以上の行レベルのセキュリティポリシーがアタッチされているすべての関係とユーザーのリストを確認します。

sys:secadmin ロールを持つユーザーのみがこのビューを照会できます。

テーブルの列

列名	データ型	説明
relschema	text	行レベルのセキュリティポリシーがアタッチされている関係のスキーマの名前。
relname	text	行レベルのセキュリティポリシーがアタッチされている関係の名前。
relkind	text	テーブルなど、オブジェクトのタイプ。
polname	text	関係にアタッチされている行レベルのセキュリティポリシー名。
grantor	text	このポリシーをアタッチしたユーザーの名前。
grantee	text	このポリシーをアタッチしたユーザーまたはロールの名前。
granteekind	text	被付与者のタイプ。指定できる値は ユーザーまたはロールです。
is_pol_on	ブール値	テーブルで行レベルのセキュリティポリシーがテーブルで有効になっているかどうか示すパラメータ。可能な値は true と false です。
is_rls_on	ブール値	テーブルで行レベルのセキュリティがテーブルで有効になっているかどうか示すパラメータ。可能な値は true と false です。
rls_conjunction_type	character(3)	リレーションが RLS ポリシーを and をで組み合わせるか、or で組み合わせるかを示すパラメータ。

サンプルクエリ

次の例では、SVV_RLS_ATTACHED_POLICY の結果を示します。

```
--Inspect the policy in SVV_RLS_ATTACHED_POLICY
SELECT * FROM svv_rls_attached_policy;

 relschema |         relname          | relkind |     polname      | grantor | grantee
 | granteekind | is_pol_on | is_rls_on | rls_conjuntion_type
-----------+--------------------------+---------+------------------+---------+----------
+-------------+-----------+-----------+--------------------
 public    | tickit_category_redshift |  table  | policy_concerts |   bob    |  analyst
 |    role     |    True   |    True   |        and
 public    | tickit_category_redshift |  table  | policy_concerts |   bob    |  dbadmin
 |    role     |    True   |    True   |        and
```

SVV_RLS_POLICY

SVV_RLS_POLICY を使用して、Amazon Redshift クラスターで作成されたすべての行レベルのセキュリティポリシーのリストを確認します。

SVV_RLS_POLICY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
poldb	text	行レベルのセキュリティポリシーが作成されるデータベースの名前。
polname	text	行レベルのセキュリティポリシーの名前。
polalias	text	ポリシー定義で使用されるテーブルエイリアス。
polatts	text	ポリシー定義で規定される属性。
polqual	text	CREATE POLICY ステートメントの USING 句で規定されるポリシー条件。
polenabled	ブール値	ポリシーがグローバルに有効化されているかの有無。
polmodifiedby	text	ポリシーを最後に作成または変更したユーザーの名前。

列名	データ型	説明
polmodifiedtime	timestamp	ポリシーが作成または最後に変更されたときのタイムスタンプ。

サンプルクエリ

次の例では、SVV_RLS_POLICY の結果を示します。

```
-- Create some policies.
CREATE RLS POLICY pol1 WITH (a int) AS t USING ( t.a IS NOT NULL );
CREATE RLS POLICY pol2 WITH (c varchar(10)) AS t USING ( c LIKE '%public%');

-- Inspect the policy in SVV_RLS_POLICY
SELECT * FROM svv_rls_policy;

 poldb | polname | polalias |                     polatts                      |
        polqual                | polenabled | polmodifiedby |   polmodifiedtime
-------+---------+----------+--------------------------------------------------
+---------------------------------------+------------+---------------
+---------------------
 my_db | pol1    | t        | [{"colname":"a","type":"integer"}]               |
 "t"."a" IS NOT NULL                   | t          | policy_admin  | 2022-02-11
 14:40:49
 my_db | pol2    | t        | [{"colname":"c","type":"character varying(10)"}] |
 "t"."c" LIKE CAST('%public%' AS TEXT) | t          | policy_admin  | 2022-02-11
 14:41:28
```

SVV_RLS_RELATION

SVV_RLS_RELATION を使用して、RLS で保護されたすべての関係のリストを確認します。

SVV_RLS_RELATION はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
datname	text	関係を含むデータベースの名前。
relschema	text	関係を含むスキーマの名前。
relname	text	リレーションの名前。
relkind	text	テーブルやビューなど、関係のタイプ。
is_rls_on	ブール値	関係が RLS で保護されているかどうか示すパラメータ。
is_rls_datashare_on	ブール値	データ共有において、関係が RLS 保護されているかどうか示すパラメータ。
rls_conjunction_type	character(3)	リレーションが RLS ポリシーを and をで組み合わせるか、`or` で組み合わせるかを示すパラメータ。
rls_datashare_conjunction_type	character(3)	リレーションが、データ共有上で RLS ポリシーを and で組み合わせるか、`or` で組み合わせるかを示すパラメータ。

サンプルクエリ

次の例では、SVV_RLS_RELATION の結果を示します。

```
ALTER TABLE tickit_category_redshift ROW LEVEL SECURITY ON FOR DATASHARES;


--Inspect RLS state on the relations using SVV_RLS_RELATION.
SELECT datname, relschema, relname, relkind, is_rls_on, is_rls_datashare_on FROM
 svv_rls_relation ORDER BY relname;

  datname   | relschema |         relname          | relkind | is_rls_on |
 is_rls_datashare_on | rls_conjunction_type | rls_datashare_conjunction_type
-----------+-----------+--------------------------+---------+-----------
+---------------------+----------------------+-------------------------------
```

```
 tickit_db |   public   | tickit_category_redshift |  table   |      t     |           t
       |          and          |                    and
(1 row)
```

SVV_ROLE_GRANTS

SVV_ROLE_GRANTS を使用して、クラスター内のロールが明示的に付与されたロールのリストを表示します。

SVV_ROLE_GRANTS は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
role_id	integer	ロールの ID。
role_name	text	ロールの名前。
granted_role_id	integer	付与されたロールの ID。
granted_role_name	text	付与されたロールの名前。

サンプルクエリ

次の例では、SVV_ROLE_GRANTS の出力を返します。

```
GRANT ROLE role1 TO ROLE role2;
GRANT ROLE role2 TO ROLE role3;

SELECT role_name, granted_role_name FROM svv_role_grants;

 role_name |  granted_role_name
-----------+--------------------
```

```
   role2   |      role1
   role3   |      role2
(2 rows)
```

SVV_ROLES

SVV_ROLES を使用してロール情報を表示します。

このテーブルはすべてのユーザーに表示されます。

テーブルの列

列名	データ型	説明
role_id	integer	ロール ID。
role_name	text	ロールの名前。
role_owner	text	ロール所有者の名前。
external_id	text	サードパーティーにおけるアイデンティティプロバイダー内のロールの一意の識別子。

サンプルクエリ

次の例では、SVV_ROLES の出力を返します。

```
SELECT role_name,role_owner FROM svv_roles WHERE role_name IN ('role1', 'role2');

 role_name | role_owner
-----------+------------
   role1   | superuser
   role2   | superuser
```

SVV_SCHEMA_PRIVILEGES

SVV_SCHEMA_PRIVILEGES を使用して、現在のデータベース内のユーザー、ロール、およびグループに明示的に付与されているスキーマのアクセス許可を表示します。

SVV_SCHEMA_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
namespace_name	text	指定されたスキーマが存在する名前空間の名前。
privilege_type	text	アクセス許可の種類。スキーマの privilege_scope を使用するアクセス許可の場合、指定できる値は CREATE、USAGE、ALTER です。スキーマ以外の privilege_scope 値の場合、指定できる値には、アクセス許可のスコープで使用できる任意のアクセス許可タイプが含まれます。
identity_id	integer	アイデンティティの ID。指定できる値は、ユーザー ID、ロール ID、またはグループ ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は、ユーザー、ロール、グループ、またはパブリックです。
admin_option	ブール値	ユーザーが他のユーザーおよびロールにアクセス許可を付与できるかどうかを示す値。ロールおよびグループのアイデンティティタイプでは、常に false です。

列名	データ型	説明
privilege_scope	text	privilege_type で指定されたアクセス許可の範囲。可能な値は以下のとおりです。 • SCHEMA • TABLES • FUNCTIONS 範囲指定されたアクセス許可の詳細については、「スコープ設定アクセス許可」を参照してください。

サンプルクエリ

次の例では、SVV_SCHEMA_PRIVILEGES の結果を示します。

```
SELECT namespace_name,privilege_type,identity_name,identity_type,admin_option FROM
 svv_schema_privileges
WHERE namespace_name = 'test_schema1';

 namespace_name | privilege_type |  identity_name | identity_type | admin_option
----------------+----------------+----------------+---------------+--------------
 test_schema1   |    USAGE       |     reguser    |     user      |   False
 test_schema1   |    USAGE       |     role1      |     role      |   False
```

SVV_SCHEMA_QUOTA_STATE

各スキーマのクォータと現在のディスク使用量を表示します。

通常のユーザーは、USAGE 権限を持つスキーマの情報を表示できます。スーパーユーザーは、現在のデータベース内のすべてのスキーマの情報を表示できます。

SVV_SCHEMA_QUOTA_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
schema_id	integer	名前空間またはスキーマの ID。
schema_name	character (128)	名前空間またはスキーマの名前。
schema_owner	integer	スキーマの所有者の内部ユーザー ID。
クォータ	integer	スキーマが使用できるディスク容量 (MB)。
disk_usage	integer	スキーマによって現在使用されているディスク容量 (MB)。
disk_usage_pct	double precision	設定されたクォータのうち、スキーマによって現在使用されているディスク容量の割合。

サンプルクエリ

次の例では、スキーマのクォータと現在のディスク使用量が表示されます。

```
SELECT TRIM(SCHEMA_NAME) "schema_name", QUOTA, disk_usage, disk_usage_pct FROM
 svv_schema_quota_state
WHERE SCHEMA_NAME = 'sales_schema';
schema_name    | quota | disk_usage | disk_usage_pct
---------------+-------+------------+----------------
sales_schema   | 2048  | 30         | 1.46
(1 row)
```

SVV_SYSTEM_PRIVILEGES

SVV_SYSTEM_PRIVILEGES は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、アクセスを許可された ID 、あるいは自らが所有する ID のみ見ることができます。

テーブルの列

列名	データ型	説明
system_privilege	text	システムのアクセス許可の名前。
identity_id	integer	アイデンティティの ID。指定できる値は ユーザー ID またはロール ID です。
identity_name	text	アイデンティティの名前。
identity_type	text	ID のタイプ。指定できる値は ユーザーまたはロールです。

サンプルクエリ

次の例では、指定されたパラメータの結果を示します。

```
SELECT system_privilege,identity_name,identity_type FROM svv_system_privileges
WHERE system_privilege = 'ALTER TABLE' AND identity_name = 'sys:superuser';

 system_privilege | identity_name | identity_type
------------------+---------------+---------------
   ALTER TABLE    | sys:superuser |     role
```

SVV_TABLE_INFO

データベースのテーブルに関する概要情報を表示します。ビューではシステムテーブルが絞り込まれ、ユーザー定義テーブルのみが表示されます。

SVV_TABLE_INFO ビューを使用すると、クエリのパフォーマンスに影響する可能性のあるテーブル設計の問題を診断し、それに対処できます。これには、圧縮エンコード、分散キー、ソートスタイル、データ分散スキュー、テーブルサイズ、統計情報が含まれます。SVV_TABLE_INFO ビューは、空のテーブルの情報を返しません。

SVV_TABLE_INFO ビューは、STV_BLOCKLIST、STV_NODE_STORAGE_CAPACITY、STV_TBL_PERM、および STV_SLICES システムテーブルと、PG_DATABASE、PG_ATTRIBUTE、PG_CLASS、PG_NAMESPACE、および PG_TYPE カタログテーブルからの情報を要約します。

SVV_TABLE_INFO はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。ユーザーにビューのクエリを許可するには、SVV_TABLE_INFO で SELECT のアクセス許可をユーザーに付与します。

テーブルの列

列名	データ型	説明
database	text	データベース名。
schema	text	スキーマ名。
table_id	oid	テーブル ID。
table	text	テーブル名。
encoded	text	いずれかの列で圧縮エンコードが定義されているかどうかを示す値。
diststyle	text	キー分散が定義されている場合の、分散スタイルまたは分散キー列。指定できる値には、`EVEN`、`KEY(column)`、`ALL`、`AUT`

列名	データ型	説明
		、AUTO(EVEN) 、および AUTO(KEY(*column*)) などがあります。
sortkey1	text	ソートキーが定義されている場合の、ソートキーの最初の列。指定できる値には、*column*、AUTO(SORT KEY) 、AUTO(SORT KEY(*column*)) などがあります。
max_varchar	integer	VARCHAR データ型を使用する最大の列のサイズ。
sortkey1_enc	character(32)	ソートキーが定義されている場合の、ソートキーの最初の列の圧縮エンコード。
sortkey_num	integer	ソートキーとして定義された列の数。
size	bigint	テーブルのサイズ (1 MB のデータブロック単位)。
pct_used	numeric(10,4)	テーブルで使用されている使用可能スペースの割合。
empty	bigint	内部使用を目的とします。この列は使用されておらず、将来のリリースでは削除されます。
unsorted	numeric(5,2)	テーブル内のソートされていない行の割合。

列名	データ型	説明
stats_off	numeric(5,2)	テーブルの統計情報の古さを示す数。0 は最新で、100 は最新でないことを示します。
tbl_rows	numeric(38,0)	テーブル内の合計行数。この値には、削除対象としてマークされ、まだバキューム処理されていない列が含まれます。
skew_sortkey1	numeric(19,2)	ソートキーが定義されている場合の、ソートしないキーの最大の列のサイズから、ソートキーの最初の列のサイズの割合。この値を使用して、ソートキーの効果を評価します。
skew_rows	numeric(19,2)	最も多くの行を含むスライスの行数と、最も少ない行を含むスライスの行数の比率。
estimated_visible_rows	numeric(38,0)	テーブル内の予測された行。この値には、削除対象としてマークされた行は含まれません。

<table>
<tr><th>列名</th><th>データ型</th><th>説明</th></tr>
<tr><td>risk_event</td><td>text</td><td>テーブルに関するリスク情報。フィールドは、以下の部分に分割されます。
<pre><code>risk_type |xid|timestamp</code></pre>

<code>risk_type</code> 、ここで 1 は <code>COPY command with the EXPLICIT_IDS option</code> が実行されたことを示します。Amazon Redshift は、テーブル内の IDENTITY 列の一意性をチェックしなくなりました。詳細については、「EXPLICIT_IDS」を参照してください。
トランザクション ID <code>xid</code>。リスクが発生していました。
<code>timestamp</code> (COPY コマンド実行時)。

次の例は、フィールドの値を示しています。
<pre><code>1|1107|2019-06-22
 07:16:11.292952</code></pre></td></tr>
<tr><td>vacuum_sort_benefit</td><td>numeric(12,2)</td><td>VACUUM ソートを実行した場合に推定されるスキャンクエリパフォーマンスの最大改善率 (%)。</td></tr>
</table>

列名	データ型	説明
create_time	タイムゾーンなしのタイムスタンプ	テーブルが作成された時刻のタイムスタンプ。

サンプルクエリ

次の例は、データベースのすべてのユーザー定義テーブル用のエンコード、分散スタイル、ソート、およびデータスキューを示しています。ここで "table" は予約語であるため、二重引用符で囲む必要があります。

```
select "table", encoded, diststyle, sortkey1, skew_sortkey1, skew_rows
from svv_table_info
order by 1;

table           | encoded | diststyle       | sortkey1     | skew_sortkey1 | skew_rows
----------------+---------+-----------------+--------------+---------------+----------
category        | N       | EVEN            |              |               |
date            | N       | ALL             | dateid       |          1.00 |
event           | Y       | KEY(eventid)    | dateid       |          1.00 |      1.02
listing         | Y       | KEY(listid)     | dateid       |          1.00 |      1.01
sales           | Y       | KEY(listid)     | dateid       |          1.00 |      1.02
users           | Y       | KEY(userid)     | userid       |          1.00 |      1.01
venue           | N       | ALL             | venueid      |          1.00 |
(7 rows)
```

SVV_TABLES

SVV_TABLES を使用して、ローカルと外部カタログのテーブルを表示します。

SVV_TABLES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
table_catalog	text	既存のテーブルがあるカタログの名前。
table_schema	text	テーブルのスキーマの名前。
table_name	text	テーブルの名前。
table_type	text	テーブルの種類。指定できる値は、ビュー、外部テーブル、およびベーステーブルです。
解説	text	解説。

SVV_TRANSACTIONS

現在データベーステーブルのロックを保持するトランザクションについて情報を記録します。SVV_TRANSACTIONS のビューを使用して、開いているトランザクション、ロックの衝突の問題を識別します。ロックについての詳細については、「同時書き込み操作を管理する」および「LOCK」を参照してください。

SVV_TRANSACTIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
txn_owner	text	トランザクションの所有者の名前。
txn_db	text	トランザクションに関連付けられたデータベースの名前。

列名	データ型	説明
xid	bigint	トランザクション ID。
pid	integer	ロックに関連付けられたプロセス ID。
txn_start	timestamp	トランザクションの開始時間。
lock_mode	text	このプロセスで保持、またはリクエストされたロックモードの名前。`lock_mode` が `ExclusiveLock` で、`granted` が true の場合 (t)、このトランザクション ID は、開いているトランザクションです。
lockable_object_type	text	ロックを保持またはリクエストしているオブジェクトのタイプは、テーブルである場合は、`relation` で、トランザクションである場合は、`transactionid` です。
リレーション	integer	ロックを取得するテーブル (リレーション) のテーブル ID。この値は、`lockable_object_type` が `transactionid` である場合、NULL です。
許可済	boolean	ロックが許可済か `t` 保留中か (f) を示す値。

サンプルクエリ

次のコマンドは、すべてのアクティブなトランザクションおよび各トランザクションにリクエストされたロックを示しています。

```
select * from svv_transactions;

 txn_
 lockable_
 owner | txn_db |  xid   |  pid  |         txn_start          |      lock_mode      |
 object_type    | relation | granted
-------+--------+--------+-------+----------------------------+---------------------
+----------------+----------+---------
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | AccessShareLock     |
 relation       |   100068 | t
 root  | dev    | 438484 | 22223 | 2016-03-02 18:42:18.862254 | ExclusiveLock       |
 transactionid  |          | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    50860 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | AccessShareLock     |
 relation       |    52310 | t
 root  | tickit | 438490 | 22277 | 2016-03-02 18:42:48.084037 | ExclusiveLock       |
 transactionid  |          | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100068 | f
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | RowExclusiveLock    |
 relation       |    16688 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessShareLock     |
 relation       |   100064 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100166 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100171 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | AccessExclusiveLock |
 relation       |   100190 | t
 root  | dev    | 438505 | 22378 | 2016-03-02 18:43:27.611292 | ExclusiveLock       |
 transactionid  |          | t
(12 rows)

(12 rows)
```

SVV_USER_GRANTS

SVV_USER_GRANTS を使用して、クラスター内のロールが明示的に付与されているユーザーのリストを表示します。

SVV_USER_GRANTS は以下のユーザーに表示されます。

- スーパーユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

他のユーザーは、自分に明示的に付与されたロールのみを表示できます。

テーブルの列

列名	データ型	説明
user_id	integer	このユーザーのユーザー ID。
user_name	text	ユーザーの名前。
role_id	integer	付与されたロールのロール ID。
role_name	text	付与されたロールのロール名。
admin_option	ブール値	ユーザーがロールを他のユーザーおよびロールに付与できるかどうかを示す値。

サンプルクエリ

次のクエリは、ユーザーにロールを付与します。また、ロールが明示的に付与されたユーザーのリストを示します。

```
GRANT ROLE role1 TO reguser;
GRANT ROLE role2 TO reguser;
GRANT ROLE role1 TO superuser;
GRANT ROLE role2 TO superuser;
```

```
SELECT user_name,role_name,admin_option FROM svv_user_grants;

 user_name | role_name | admin_option
-----------+-----------+--------------
 superuser |  role1    | False
 reguser   |  role1    | False
 superuser |  role2    | False
  reguser  |  role2    | False
```

SVV_USER_INFO

SVV_USER_INFO ビューを使用して、Amazon Redshift データベースユーザーに関するデータを取得できます。

SVV_USER_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_name	text	このロールのユーザー名。
user_id	integer	このユーザーのユーザー ID。
createdb	ブール値	ユーザーがデータベースを作成する権限を持っているかどうかを示す値。
スーパーユーザー	ブール値	ユーザーがスーパーユーザーであるかどうかを示す値。
catalog_update	ブール値	ユーザーがシステムカタログを更新できるかどうかを示す値。
connection_limit	text	ユーザーが開くことができる接続数。
syslog_access	text	ユーザーがシステムログにアクセスできるかどうかを示す値。指定できる値は RESTRICTED と UNRESTRICTED の 2 つです。RESTRICTED

列名	データ型	説明
		は、スーパーユーザーではないユーザーが自分のレコードを表示できることを意味します。UNRESTRICTED は、スーパーユーザーではないユーザーが、SELECT 権限を持っているシステムビューおよびテーブル内のすべてのレコードを表示できることを意味します。
last_ddl_timestamp	timestamp	ユーザーが実行した最後のデータ定義言語 (DDL) create ステートメントのタイムスタンプ。
session_timeout	integer	タイムアウトするまで、セッションが非アクティブまたはアイドル状態を維持する最大秒数。0 の場合は、タイムアウトが設定されていないことを示します。クラスターのアイドルまたは非アクティブ状態のタイムアウト設定については、「Amazon Redshift 管理ガイド」の「Amazon Redshift でのクォータと制限」を参照してください。
external_user_id	text	サードパーティーにおける ID プロバイダー内のユーザーの一意の識別子。

サンプルクエリ

次のコマンドは SVV_USER_INFO からユーザー情報を取得します。

```
SELECT * FROM SVV_USER_INFO;
```

SVV_VACUUM_PROGRESS

このビューは、現在進行中のバキューム処理が終了するまでにかかる時間の予測を返します。

SVV_VACUUM_PROGRESS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_VACUUM_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

SVV_VACUUM_SUMMARY の詳細については、「SVV_VACUUM_SUMMARY」を参照してください。

SVL_VACUUM_PERCENTAGE の詳細については、「SVL_VACUUM_PERCENTAGE」を参照してください。

> **Note**
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
table_name	text	現在バキューム処理中のテーブル名、または現在進行中の処理がない場合は最後にバキューム処理されたテーブル。
status	text	バキュームオペレーションの一環で実行された現在のアクティビティの説明。 • Initialize • Sort • Merge • 削除 • 選択 • 失敗 • 完了 • Skipped • INTERLEAVED SORTKEY の順序の構築
time_remaining_estimate	text	現在のバキューム処理が完了するまでの残り時間の予測 (分および秒単位)。例えば、**`5m 10s`** などです。予測時間は、バキューム処理が最初のソート処理を完了するまでは返却されません。進行中のバキューム処理がない場合は、最後に実行されたバキュームが、STATUS 列に **`Completed`** があり、TIME_REMAINING_ESTIMATE 列が空の状態で表示され

列名	データ型	説明
		ます。予測は通常、バキューム処理が進むにしたがってより精度が増します。

サンプルクエリ

次のクエリでは (数分の間隔を開けて実行)、SALESNEW という大型のテーブルがバキューム処理される様子を示します。

```
select * from svv_vacuum_progress;

table_name    |            status             | time_remaining_estimate
--------------+-------------------------------+-------------------------
salesnew      |  Vacuum: initialize salesnew  |
(1 row)
...
select * from svv_vacuum_progress;

table_name   |         status         | time_remaining_estimate
-------------+------------------------+-------------------------
salesnew     |  Vacuum salesnew sort  | 33m 21s
(1 row)
```

次のクエリでは、現在進行中のバキューム処理がないことを示します。最後にバキューム処理されたテーブルは SALES テーブルです。

```
select * from svv_vacuum_progress;

table_name   |  status  | time_remaining_estimate
-------------+----------+-------------------------
  sales      | Complete |
(1 row)
```

SVV_VACUUM_SUMMARY

SVV_VACUUM_SUMMARY ビューは STL_VACUUM テーブル、STL_QUERY テーブル、STV_TBL_PERM テーブルを結合し、システムがログ記録したバキューム処理についての情報を要約します。ビューはバキュームトランザクションごとに、テーブルにつき 1 行を返します。

ビューは処理の経過時間、作成されたソートパーティションの数、必須のマージインクリメントの数、処理の実行前後の行およびブロックカウントの差異を示します。

SVV_VACUUM_SUMMARY はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_VACUUM_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

SVV_VACUUM_PROGRESS の詳細については、「SVV_VACUUM_PROGRESS」を参照してください。

SVL_VACUUM_PERCENTAGE の詳細については、「SVL_VACUUM_PERCENTAGE」を参照してください。

> **Note**
>
> このビューはプロビジョニングされたクラスターをクエリする場合のみ使用できます。

テーブルの列

列名	データ型	説明
table_name	text	バキューム処理されたテーブルの名前。
xid	bigint	バキューム処理のトランザクション ID。
sort_partitions	bigint	バキューム処理のソートフェーズ間に作成されたソートパーティションの数。
merge_increments	bigint	バキューム処理のマージフェーズを実行するために必要なマージインクリメントの数。
elapsed_time	bigint	バキューム処理の経過した実行時間 (マイクロ秒)。
row_delta	bigint	バキューム処理前後の合計テーブル行数の差異。

列名	データ型	説明
sortedrow_delta	bigint	バキューム処理前後のソート済みテーブル行の数の差異。
block_delta	integer	バキューム処理前後のテーブルのブロックカウントの差異。
max_merge_partitions	integer	この列は、パフォーマンスの分析に使用され、バキュームが 1 回のマージフェーズで処理できるテーブルのパーティションの最大数を表します (バキュームでは、ソートされていないリージョンを 1 つ以上のソートされたパーティションにソートます。テーブルの列数と現在の Amazon Redshift の設定に応じて、マージフェーズでは繰り返しマージ処理のうち 1 回で最大数のパーティションを処理できます。マージフェーズは、ソートされたパーティションの数がマージパーティションの最大数を超えても引き続き有効ですが、繰り返しマージ処理をさらに続ける必要があります。)

サンプルクエリ

次のクエリは、3 つの異なるテーブルでのバキューム処理の統計を返します。SALES テーブルが 2 回バキューム処理されています。

```
select table_name, xid, sort_partitions as parts, merge_increments as merges,
elapsed_time, row_delta, sortedrow_delta as sorted_delta, block_delta
from svv_vacuum_summary
order by xid;

table_  | xid  |parts|merges| elapsed_ | row_    | sorted_ | block_
name    |      |     |      | time     | delta   | delta   | delta
--------+------+-----+------+----------+---------+---------+--------
users   | 2985 |   1 |    1 | 61919653 |       0 |   49990 |      20
category| 3982 |   1 |    1 | 24136484 |       0 |      11 |       0
sales   | 3992 |   2 |    1 | 71736163 |       0 | 1207192 |      32
sales   | 4000 |   1 |    1 | 15363010 | -851648 | -851648 |    -140
(4 rows)
```

SYS モニタリングビュー

モニタリングビューは、プロビジョニングされたクラスターとサーバーレスワークグループのクエリとワークロードリソースの使用状況を監視するための、Amazon Redshift のシステムビューです。これらのビューは、`pg_catalog` スキーマ内にあります。これらのビューで提供される情報を表示するには、SQL SELECT ステートメントを実行します。

特に明記されていない限り、これらのビューは Amazon Redshift クラスターと Amazon Redshift Serverless ワークグループで使用できます。

SYS_SERVERLESS_USAGE は、Amazon Redshift Serverless のみの使用状況データを収集します。

トピック

- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_QUERY_DETAIL
- SYS_QUERY_HISTORY
- SYS_QUERY_TEXT
- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SERVERLESS_USAGE
- SYS_SESSION_HISTORY
- SYS_SPATIAL_SIMPLIFY
- SYS_STREAM_SCAN_ERRORS
- SYS_STREAM_SCAN_STATES
- SYS_TRANSACTION_HISTORY
- SYS_UDF_LOG
- SYS_UNLOAD_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_USERLOG
- SYS_VACUUM_HISTORY

SYS_ANALYZE_COMPRESSION_HISTORY

COPY または ANALYZE COMPRESSION コマンドの実行中に圧縮分析オペレーションの詳細を記録します。

SYS_ANALYZE_COMPRESSION_HISTORY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	エントリを生成したユーザーの ID。
start_time	timestamp	圧縮分析オペレーションを開始した時刻。
transaction_id	bigint	圧縮分析オペレーションのトランザクション ID。
table_id	integer	分析されたテーブルのテーブル ID。
table_name	character(128)	分析されたテーブルの名前。
column_position	integer	圧縮エンコードを決定するために分析されたテーブルの列のインデックス。
old_encoding	character(15)	圧縮分析前のエンコードタイプ。
new_encoding	character(15)	圧縮分析後のエンコードタイプ。
mode	character(14)	指定できる値は以下のとおりです。 PRESET `new_encoding` が、列のデータタイプに基づいて Amazon Redshift COPY コマンドによって決定されることを指定します。サンプリングされているデータはありません。 ON `new_encoding` が、サンプルデータの分析に基づいて Amazon Redshift COPY コマンドによって決定されることを指定します。

列名	データ型	説明
		ANALYZE ONLY new_encoding が、サンプルデータの分析に基づいて Amazon Redshift ANALYZE COMPRESSION コマンドによって決定されることを指定します。ただし、分析された列のエンコードタイプは変更されません。

サンプルクエリ

次の例では、同じセッションで実行された最後の COPY コマンドで、`lineitem` テーブルの圧縮分析の詳細を検査します。

```
select transaction_id, table_id, btrim(table_name) as table_name, column_position,
 old_encoding, new_encoding, mode
from sys_analyze_compression_history
where transaction_id = (select transaction_id from sys_query_history where query_id =
 pg_last_copy_id()) order by column_position;

 transaction_id  |  table_id   | table_name | column_position |  old_encoding   |
 new_encoding    |      mode
-----------------+-------------+------------+-----------------+-----------------
+-----------------+-------------
      8196       |   248126    | lineitem   |        0        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        1        | mostly32        | lzo
           | ON
      8196       |   248126    | lineitem   |        2        | lzo             |
 delta32k        | ON
      8196       |   248126    | lineitem   |        3        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        4        | bytedict        |
 bytedict        | ON
      8196       |   248126    | lineitem   |        5        | mostly32        |
 mostly32        | ON
      8196       |   248126    | lineitem   |        6        | delta           | delta
           | ON
      8196       |   248126    | lineitem   |        7        | delta           | delta
           | ON
```

```
       8196        |   248126     | lineitem   |        8        | lzo              | zstd
            | ON
       8196        |   248126     | lineitem   |        9        | runlength        | zstd
            | ON
       8196        |   248126     | lineitem   |       10        | delta            | lzo
            | ON
       8196        |   248126     | lineitem   |       11        | delta            | delta
            | ON
       8196        |   248126     | lineitem   |       12        | delta            | delta
            | ON
       8196        |   248126     | lineitem   |       13        | bytedict         | zstd
            | ON
       8196        |   248126     | lineitem   |       14        | bytedict         | zstd
            | ON
       8196        |   248126     | lineitem   |       15        | text255          | zstd
            | ON
(16 rows)
```

SYS_ANALYZE_HISTORY

ANALYZE オペレーションの詳細を記録します。

SYS_ANALYZE_HISTORY はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	エントリを生成したユーザーの ID。
transaction_id	long	トランザクション ID。
query_id	long	SYS_QUERY_HISTORY 内のクエリ識別子。
database_name	char(30)	データベースの名前。
table_name	char(30)	テーブルの名前。
table_id	integer	テーブルの ID。

列名	データ型	説明
is_automatic	char(1)	オペレーションにデフォルトで Amazon Redshift の ANALYZE オペレーションが含まれている場合、値は true (t) です。ANALYZE コマンドが明示的に実行された場合、値は false (f) です。
status	char(15)	ANALYZE コマンドの結果。指定できる値は、Full、Skipped、および PredicateColumn です。
start_time	timestamp	ANALYZE オペレーションの実行を開始した時刻 (UTC)。
end_time	timestamp	ANALYZE オペレーションの実行を終了した時刻 (UTC)。
rows	double	テーブル内の行の合計数。
modified_rows	double	最後の ANALYZE オペレーション以降に変更された合計行数。
analyze_threshold_percent	integer	analyze_threshold_percent パラメータの値。
last_analyze_time	timestamp	テーブルが前回分析された時刻 (UTC)。

サンプルクエリ

```
 user_id | transaction_id | database_name | schema_name |      table_name      |
 table_id | is_automatic | Status |          start_time          |           end_time
     | rows | modified_rows | analyze_threshold_percent |  last_analyze_time
---------+----------------+---------------+-------------+---------------------
+----------+--------------+--------+----------------------------+----------
+------------------+------+---------------+--------------------------
+---------------------
     101 |           8006 |           dev |      public | test_table_562bf8dc
 |   110427 |            f |   Full | 2023-09-21 18:33:08.504646 | 2023-09-21
 18:33:24.296498 |    5 |             5 |                         0 | 2000-01-01
 00:00:00
```

SYS_APPLIED_MASKING_POLICY_LOG

SYS_APPLIED_MASKING_POLICY_LOG を使用して、DDM で保護されたリレーションを参照するクエリに対する動的データマスキングポリシーの適用をトレースします。

SYS_APPLIED_MASKING_POLICY_LOG は、以下のユーザーに表示されます。

- スーパーユーザー
- `sys:operator` ロールを持つユーザー
- ACCESS SYSTEM TABLE のアクセス許可を持つユーザー

通常のユーザーには 0 行が表示されます。

SYS_APPLIED_MASKING_POLICY_LOG は、`sys:secadmin` ロールを持つユーザーには表示されないことに注意してください。

動的データマスキングの詳細については、「動的データマスキング」を参照してください。

テーブルの列

列名	データ型	説明
policy_name	text	マスキングポリシーの名前。
user_id	text	クエリを実行したユーザーの ID。
record_time	timestamp	システムビューエントリを記録した時刻。
session_id	整数	プロセス ID。
transaction_id	long	トランザクション ID。
query_id	整数	クエリ ID。
database_name	text	クエリを実行したデータベースの名前。

列名	データ型	説明
relation_name	text	マスキングポリシーをアタッチする先のテーブルの名前。
schema_name	text	テーブルが含まれているスキーマの名前
attachment_id	long	アタッチされたマスキングポリシーの ID。
relation_kind	text	マスキングポリシーを適用する先のリレーションのタイプ。指定できる値は TABLE、VIEW、LATE BINDING VIEW、MATERIALIZED VIEW です。

サンプルクエリ

次の例は、mask_credit_card_full マスキングポリシーが credit_db.public.credit_cards テーブルにアタッチされていることを示しています。

```
select policy_name, database_name, relation_name, schema_name, relation_kind
from sys_applied_masking_policy_log;

policy_name           | database_name | relation_name | schema_name | relation_kind
----------------------+---------------+---------------+-------------+---------------
mask_credit_card_full | credit_db     | credit_cards  | public      | table

(1 row)
```

SYS_AUTO_TABLE_OPTIMIZATION

自動最適化用に定義されたテーブルに Amazon Redshift によって実行された自動アクションを記録します。

SYS_AUTO_TABLE_OPTIMIZATION は、スーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
transaction_id	long	トランザクション識別子。
session_id	整数	alter コマンドを実行したプロセスのセッション識別子。
table_id	整数	テーブル識別子。
alter_table_type	character (32)	レコメンデーションのタイプ。指定できる値は distkey、sortkey、および encode です。
status	character (128)	レコメンデーションの完了ステータス。使用できる値は、Start、Complete、Skipped、Abort、Checkpoint 、および Failed です。
event_time	timestamp	ステータス列のタイムスタンプ。
alter_from	character (200)	レコメンデーションを適用する前に、テーブルの以前のディストリビューションスタイルとソートキー。この値は 200 文字刻みの増分に切り捨てられます。

サンプルクエリ

次の例では、結果の行に Amazon Redshift が実行したアクションが表示されます。

```
SELECT table_id, alter_table_type, status, event_time, alter_from
FROM SYS_AUTO_TABLE_OPTIMIZATION;

 table_id |  alter_table_type   |                        status
 |          event_time          |  alter_from
----------+---------------------+-----------------------------------------------------------------
+-----------------------------+-----------------
   118082 | sortkey             | Start
 | 2020-08-22 19:42:20.727049   |
   118078 | sortkey             | Start
 | 2020-08-22 19:43:54.728819   |
```

```
    118082 | sortkey              | Start
  | 2020-08-22 19:42:52.690264  |
    118072 | sortkey              | Start
  | 2020-08-22 19:44:14.793572  |
    118082 | sortkey              | Failed
  | 2020-08-22 19:42:20.728917  |
    118078 | sortkey              | Complete
  | 2020-08-22 19:43:54.792705  | SORTKEY: None;
    118086 | sortkey              | Complete
  | 2020-08-22 19:42:00.72635   | SORTKEY: None;
    118082 | sortkey              | Complete
  | 2020-08-22 19:43:34.728144  | SORTKEY: None;
    118072 | sortkey              | Skipped:Retry exceeds the maximum limit for a table.
  | 2020-08-22 19:44:46.706155  |
    118086 | sortkey              | Start
  | 2020-08-22 19:42:00.685255  |
    118082 | sortkey              | Start
  | 2020-08-22 19:43:34.69531   |
    118072 | sortkey              | Start
  | 2020-08-22 19:44:46.703331  |
    118082 | sortkey              | Checkpoint: progress 14.755079%
  | 2020-08-22 19:42:52.692828  |
    118072 | sortkey              | Failed
  | 2020-08-22 19:44:14.796071  |
    116723 | sortkey              | Abort:This table is not AUTO.
  | 2020-10-28 05:12:58.479233  |
    110203 | distkey              | Abort:This table is not AUTO.
  | 2020-10-28 05:45:54.67259   |
```

SYS_CONNECTION_LOG

認証の試みと、接続および切断をログに記録します。

SYS_CONNECTION_LOG はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
event	character(50)	接続または認証イベント。

列名	データ型	説明
record_time	timestamp	イベントが発生した時刻。
remote_host	character(45)	リモートホストの名前または IP アドレス。
remote_port	character(32)	リモートホストのポート番号。
session_id	integer	ステートメントに関連付けられるプロセス ID。
database_name	character(50)	データベース名。
user_name	character(50)	ユーザーネーム。
auth_method	character(32)	認証方法。
duration	integer	接続時間 (マイクロ秒)。
ssl_version	character(50)	Secure Sockets Layer (SSL) バージョン。
ssl_cipher	character(128)	SSL 暗号。
mtu	integer	最大送信単位 (MTU)。
ssl_compression	character(64)	SSL 圧縮タイプ。
ssl_expansion	character(64)	SSL 拡張タイプ。
iam_auth_guid	character(36)	CloudTrail リクエストの IAM 認証 ID。
application_name	character(250)	セッションのアプリケーションの初期名または更新名。

列名	データ型	説明
driver_version	character(64)	サードパーティーの SQL クライアントツールから Amazon Redshift クラスターに接続する ODBC または JDBC ドライバーのバージョン。
os_version	character(64)	Amazon Redshift クラスターに接続するクライアントマシン上にあるオペレーティングシステムのバージョン。
plugin_name	character(32)	Amazon Redshift クラスターへの接続に使用されるプラグインの名前。
protocol_version	integer	Amazon Redshift ドライバーが、サーバーとの接続を確立する際に使用する内部プロトコルのバージョン。プロトコルのバージョンは、ドライバとサーバ間でネゴシエートされます。バージョンは、利用可能な機能を説明します。有効な値を次に示します。 • 0 (BASE_SERVER_PROTOCOL_VERSION) • 1 (EXTENDED_RESULT_METADATA_SERVERSION) – クエリごとのラウンドトリップを省くため、サーバーが追加の結果セットのメタデータ情報を送信します。 • 2 (BINARY_PROTOCLO_VERSION) – 結果セットのデータ型に応じて、サーバーがバイナリ形式でデータを送信します。 • 3 (EXTENDED2_RESULT_METADATA_SERVERSION) – サーバーが列の大文字と小文字の区別 (照合順序) 情報を送信します。
global_session_id	character(36)	現在のセッションのグローバル一意識別子。セッション ID は、ノード障害による再起動後も存続します。

サンプルクエリ

オープン接続の詳細を表示するには、以下のクエリを実行します。

```
select record_time, user_name, database_name, remote_host, remote_port
```

```
from sys_connection_log
where event = 'initiating session'
and session_id not in
(select session_id from sys_connection_log
where event = 'disconnecting session')
order by 1 desc;

record_time         | user_name    | database_name    | remote_host    | remote_port

--------------------+--------------+------------------+----------------
+---------------------------------
2014-11-06 20:30:06 | rdsdb        | dev              | [local]        |

2014-11-06 20:29:37 | test001      | test             | 10.49.42.138   | 11111

2014-11-05 20:30:29 | rdsdb        | dev              | 10.49.42.138   | 33333

2014-11-05 20:28:35 | rdsdb        | dev              | [local]        |
(4 rows)
```

以下の例は、失敗した認証の試みと、成功した接続および切断を反映しています。

```
select event, record_time, remote_host, user_name
from sys_connection_log order by record_time;

            event      |         record_time        |  remote_host   | user_name

-----------------------+----------------------------+----------------+---------
authentication failure | 2012-10-25 14:41:56.96391  | 10.49.42.138   | john

authenticated          | 2012-10-25 14:42:10.87613  | 10.49.42.138   | john

initiating session     | 2012-10-25 14:42:10.87638  | 10.49.42.138   | john

disconnecting session  | 2012-10-25 14:42:19.95992  | 10.49.42.138   | john

(4 rows)
```

以下の例は、ODBC ドライバーのバージョン、クライアントマシンのオペレーティングシステム、および Amazon Redshift クラスターへの接続に使用されるプラグインを示しています。この例では、使用されるプラグインは、ログイン名とパスワードを使用した標準の ODBC ドライバー認証に使用されます。

```
select driver_version, os_version, plugin_name from sys_connection_log;

driver_version                          |  os_version                       |
 plugin_name
----------------------------------------+-----------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64              | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64   | none
```

次の例では、クライアントマシン上のオペレーティングシステムのバージョン、ドライバのバージョン、およびプロトコルのバージョンを表示します。

```
select os_version, driver_version, protocol_version from sys_connection_log;

os_version                      |  driver_version              | protocol_version
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

SYS_COPY_JOB (プレビュー)

これは、プレビューで使用できる自動コピー (SQL COPY JOB) に関する、プレリリースドキュメントです。 ドキュメントと機能はどちらも変更されることがあります。この機能については、テスト環境のみで使用し、本番環境では使用しないことをお勧めします。パブリックプレビューは 2024 年 10 月 31 日に終了します。プレビュークラスターは、プレビュー終了 2 週間後に自動的に削除されます。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

SYS_COPY_JOB を使用して、COPY JOB コマンドの詳細を表示します。

このビューには、作成された COPY JOB コマンドが含まれています。

SYS_COPY_JOB はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
job_id	bigint	コピージョブ識別子。
job_name	character(128)	コピージョブの名前。
iam_role	character(128)	COPY ステートメントで指定された IAM ロール。
job_text	character(256)	COPY ステートメントのパラメータ。
is_auto	integer	COPY JOB が Amazon Redshift によって自動的に実行されるかどうかを示します。1 は true を示し、`0` は false を示します。
on_error_suspend	integer	この情報は、内部使用に限定されています。

SYS_COPY_REPLACEMENTS

無効な UTF-8 文字が COPY コマンドの ACCEPTINVCHARS オプションによって置き換えられたときのログを表示します。SYS_COPY_REPLACEMENTS へのログエントリは、少なくとも 1 つの置き換えが必要であった各ノードスライスの最初の 100 行ごとに 1 つが追加されます。

このビューを使用して、サーバーレスワークグループとプロビジョニングされたクラスターに関する情報を確認できます。

SYS_COPY_REPLACEMENTS はすべてのユーザーに表示されます。　スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを生成したユーザーの ID。
query_id	bigint	クエリ ID。他のシステムテーブルやビューを結合するために使用する列。
table_id	integer	テーブル ID。
file_name	character (256)	COPY コマンドの入力ファイルへの完全なパス。
column_name	character (127)	無効な UTF-8 文字を含む最初のフィールド。
line_number	bigint	入力データファイル内の無効な UTF-8 文字を含む行番号。-1 は、列データファイルからコピーする場合などに行番号が使用できないことを示します。
raw_line	character (1024)	無効な UTF-8 文字を含む生のロードデータ。

サンプルクエリ

次の例は、最後に実行された COPY 操作で置き換えられた文字を返します。

```
select query_idp, table_id, file_name, line_number, colname
from sys_copy_replacements
where query = pg_last_copy_id();


 query_id | table_id |   file_name                                                 |
 line_number | column_name
 ---------+----------+-----------------------------------------------------------
+-------------+--------
    96    |    26    | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt                   |
 123 | city
```

```
    96     |    26     | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt                |
 456 | city
    96     |    26     | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt                |
 789 | city
    96     |    26     | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt                |
 012 | city
    96     |    26     | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt                |
 119 | city
...
```

SYS_DATASHARE_CHANGE_LOG

プロデューサークラスターとコンシューマークラスターの両方でデータ共有への変更を追跡するための統合ビューを記録します。

SYS_DATASHARE_CHANGE_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	アクションを実行するユーザーの ID。
user_name	varCHAR(28)	アクションを実行するユーザーの名前。
session_id	integer	セッションの ID
transaction_id	bigint	トランザクションの ID。
share_id	integer	影響を受けるデータ共有の ID。
share_name	varCHAR(28)	データ共有の名前。
source_database_id	integer	データ共有が属するデータベースの ID。

列名	データ型	説明
source_database_name	varCHAR(128)	データ共有が属するデータベースの名前。
consumer_database_id	integer	データ共有からインポートされたデータベースの ID。
consumer_database_name	varCHAR(128)	データ共有からインポートされたデータベースの名前。
arn	varchar(192)	インポートされたデータベースをバッキングするリソースの ARN。
record_time	timestamp	アクションのタイムスタンプ。
action	varCHAR(128)	実行されているアクション。可能な値は、CREATE DATASHARE、DROP DATASHARE、GRANT ALTER、REVOKE ALTER、GRANT SHARE、REVOKE SHARE、ALTER ADD、ALTER REMOVE、ALTER SET、GRANT USAGE、REVOKE USAGE、CREATE DATABASE、共有データベースでの GRANT または REVOAKE USAGE、DROP SHARED DATABASE、ALTER SHARED DATABASE です。
status	integer	アクションのステータス。可能な値は、SUCCESS コードと ERROR-ERROR CODE です。
share_object_type	varCHAR(64)	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトのタイプ。使用可能な値は、スキーマ、テーブル、列、関数、およびビューです。こちらは、プロデューサクラスターのフィールドです。
share_object_id	integer	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトの ID。こちらは、プロデューサクラスターのフィールドです。

列名	データ型	説明
share_object_name	varCHAR(28)	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトの名前。こちらは、プロデューサクラスターのフィールドです。
target_user_type	varCHAR(6)	権限が付与されたユーザーまたはグループのタイプ。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。
target_user_id	integer	権限が付与されたユーザーまたはグループの ID。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。
target_user_name	varCHAR(28)	権限が付与されたユーザーまたはグループの名前。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。
consumer_account	varCHAR(6)	データコンシューマーのアカウント ID。こちらは、プロデューサクラスターのフィールドです。
consumer_namespace	varCHAR(4)	データコンシューマーアカウントの名前空間。こちらは、プロデューサクラスターのフィールドです。
producer_account	varCHAR(6)	データ共有が属するプロデューサアカウントのアカウント ID。こちらは、コンシューマークラスタークラスターのフィールドです。
producer_namespace	varCHAR(4)	データ共有が属する製品アカウントの名前空間。こちらは、コンシューマークラスタークラスターのフィールドです。
attribute_name	varCHAR(4)	データ共有データベースまたは共有データベースの属性名。
attribute_value	varCHAR(28)	データ共有データベースまたは共有データベースの属性値。
メッセージ	varCHAR(12)	アクションが失敗したときのエラーメッセージ。

サンプルクエリ

次の例は、SYS_DATASHARE_CHANGE_LOG ビューを示しています。

```
SELECT DISTINCT action
FROM sys_datashare_change_log
WHERE share_object_name LIKE 'tickit%';

         action
 -----------------------
  "ALTER DATASHARE ADD"
```

SYS_DATASHARE_CROSS_REGION_USAGE

SYS_DATASHARE_CROSS_REGION_USAGE ビューを使用して、クロスリージョンデータ共有クエリによって発生したクロスリージョンデータ転送使用量の概要を取得します。SYS_DATASHARE_CROSS_REGION_USAGE はセグメントレベルで詳細を集計します。

SYS_DATASHARE_CROSS_REGION_USAGE はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
query_id	integer	クエリの ID。この値を使用して他のシステムテーブルおよびビューを結合します。
child_query_sequence	integer	書き換えられたユーザークエリのシーケンス。
segment_id	bigint	セグメントの番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
start_time	時系	データ転送が開始される UTC の時刻。
end_time	時系	データ転送が終了した UTC の時刻。

列名	データ型	説明
transferred_data	bigint	プロデューサーリージョンからコンシューマーリージョンに転送されたデータのバイト数。
source_region	char(25)	クエリのデータ転送元であるプロデューサーリージョン。

サンプルクエリ

次の例は、SYS_DATASHARE_CROSS_REGION_USAGE ビューを示しています。

```
SELECT query_id, segment_id, transferred_data, source_region
from sys_datashare_cross_region_usage
where query_id = pg_last_query_id()
order by query_id, segment_id;

  query_id | segment_id | transferred_data | source_region
-----------+------------+------------------+---------------
    200048 |          2 |          4194304 |    us-west-1
    200048 |          2 |          4194304 |    us-east-2
```

SYS_DATASHARE_USAGE_CONSUMER

データ共有のアクティビティと使用状況を記録します。このビューは、コンシューマークラスターにのみ関連しています。

SYS_DATASHARE_USAGE_CONSUMER はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	リクエストを発行しているユーザーの ID。

列名	データ型	説明
session_id	integer	クエリを実行しているリーダープロセスの ID。
transaction_id	bigint	現在のトランザクションのコンテキスト。
request_id	varCHAR(50)	リクエストされた API コールの一意の ID。
request_type	varCHAR(25)	プロデューサクラスターに対して行われたリクエストのタイプ。
transaction_uid	varCHAR(50)	トランザクションの一意の ID。
record_time	timestamp	アクションが記録される時刻。
status	integer	リクエストされた API コールのステータス。
error_message	varCHAR(512)	エラーのメッセージ。

サンプルクエリ

次の例は、SYS_DATASHARE_USAGE_CONSUMER ビューを示しています。

```
SELECT request_type, status, trim(error) AS error
FROM sys_datashare_usage_consumer

  request_type   | status | error_message
----------------+--------+---------------
 "GET RELATION" |   0    |
```

SYS_DATASHARE_USAGE_PRODUCER

データ共有のアクティビティと使用状況を記録します。このビューは、コンシューマークラスターにのみ関連しています。

SYS_DATASHARE_USAGE_PRODUCER はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
share_id	integer	データ共有のオブジェクト ID (OID)。
share_name	varCHAR(28)	データ共有の名前。
request_id	varCHAR(0)	リクエストされた API コールの一意の ID。
request_type	varCHAR(5)	プロデューサクラスターに対して行われたリクエストのタイプ。
object_type	varCHAR(4)	データ共有から共有されているオブジェクトのタイプ。使用可能な値は、スキーマ、テーブル、列、関数、およびビューです。
object_oid	integer	データ共有から共有されているオブジェクトの ID。
object_name	varCHAR(28)	データ共有から共有されているオブジェクトの名前。
consumer_account	varCHAR(6)	データ共有が共有されるコンシューマーアカウントのアカウント。
consumer_namespace	varCHAR(4)	データ共有が共有されるコンシューマーアカウントの名前空間。
consumer_transaction_uid	varCHAR(0)	コンシューマークラスター上のステートメントの一意のトランザクション ID。
record_time	timestamp	アクションが記録される時刻。

列名	データ型	説明
status	integer	データ共有のステータス。
error_message	varCHAR(512)	エラーのメッセージ。
consumer_region	char(64)	コンシューマークラスターが属するリージョン。

サンプルクエリ

次の例は、SYS_DATASHARE_USAGE_PRODUCER ビューを示しています。

```
SELECT DISTINCT
FROM sys_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
 ------------------
   "GET RELATION"
```

SYS_EXTERNAL_QUERY_DETAIL

SYS_EXTERNAL_QUERY_DETAIL は、クエリの詳細をセグメントレベルで表示します。それぞれの行が、特定の WLM クエリの各セグメントと対応します。この情報には、処理された行数、処理されたバイト数、Amazon S3 の外部テーブルのパーティション情報などの詳細が含まれます。同時にこのビューの各行には、SYS_QUERY_DETAIL ビューと対応するエントリも表示されます。ただし、このビューの場合に、は外部クエリ処理に関連する詳細情報が含まれます。

SYS_EXTERNAL_QUERY_DETAIL はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを送信したユーザーの ID。
query_id	bigint	外部クエリのクエリ識別子。
transaction_id	bigint	トランザクション識別子。
child_query_sequence	integer	書き換えられたユーザークエリーのシーケンス。segment_id と同様に 0 で開始されます。
segment_id	integer	クエリセグメントのセグメント ID。
source_type	character(32)	クエリのデータソースタイプ (Redshift Spectrum の場合は S3、横串検索の場合は PG)。
start_time	timestamp	クエリが開始された時刻。
end_time	timestamp	クエリが完了した時刻。
duration	bigint	クエリが消費した時間 (マイクロ秒)。
total_partitions	integer	Amazon S3 でのクエリに必要なパーティションの数。
qualified_partitions	integer	Amazon S3 のクエリがスキャンしたパーティションの数。
scanned_files	bigint	スキャンされた Amazon S3 ファイルの数。

列名	データ型	説明
returned_rows	bigint	Amazon S3 クエリでスキャンされた行数、または横串検索で返された行数。
returned_bytes	bigint	Amazon S3 クエリでスキャンされたバイト数、または横串検索で返されたバイト数。
file_format	text	Amazon S3 ファイルのファイル形式。
file_location	text	外部テーブル用の Amazon S3 の場所。
external_query_text	text	横串検索でのセグメントレベルのクエリテキスト。
warning_message	character(4000)	クエリの実行時に表示される警告メッセージ。
table_name	character(136)	実行中のステップでのテーブル名。
is_recursive	character(1)	サブフォルダの再帰的スキャンがあるかどうかを示します。
is_nested	character(1)	ネストされた列データ型にアクセスするかどうかを示します。
s3list_time	bigint	ファイルをリストする期間 (ミリ秒単位)。

列名	データ型	説明
get_partition_time	long	AWS Glue Data Catalog と Apache Hive の特定の外部オブジェクトのパーティションを一覧表示して絞り込むのにかかった時間。

サンプルクエリ

次のクエリは、外部クエリの詳細を表示します。

```
SELECT query_id,
       segment_id,
       start_time,
       end_time,
       total_partitions,
       qualified_partitions,
       scanned_files,
       returned_rows,
       returned_bytes,
       trim(external_query_text) query_text,
       trim(file_location) file_location
FROM sys_external_query_detail
ORDER BY query_id, start_time DESC
LIMIT 2;
```

サンプル出力。

```
 query_id | segment_id |         start_time         |          end_time
   | total_partitions | qualified_partitions | scanned_files | returned_rows |
 returned_bytes | query_text | file_location
----------+------------+----------------------------+----------------------------
+------------------+----------------------+---------------+---------------
+----------------+-----------+---------------
   763251 |          0 | 2022-02-15 22:32:23.312448 | 2022-02-15 22:32:24.036023 |
          3 |                    3 |             3 |         38203 |        2683414 |
          |
```

```
    763254 |          0 | 2022-02-15 22:32:40.17103  | 2022-02-15 22:32:40.839313 |
            3 |                      3 |               3 |           38203 |          2683414 |
            |
```

SYS_EXTERNAL_QUERY_ERROR

システムビュー SYS_EXTERNAL_QUERY_ERROR をクエリして Redshift Spectrum のスキャンエラーに関する情報を取得できます。SYS_EXTERNAL_QUERY_ERROR は、ログに記録されたエラーのサンプルを表示します。デフォルトは、クエリあたり 10 エントリです。

SYS_EXTERNAL_QUERY_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	この行を生成したユーザーの ID。
query_id	bigint	この行を生成したクエリのクエリ ID。
file_location	char(256)	クエリされているデータの場所。
rowid	char(2100)	ファイル内のエラー場所。`rowid` パーツは : (コロン) で区切られ、追加のパーツを今後追加することができます。 `row_offset :row_group :row_id` row_offset は、ファイル内の行のオフセット (バイト単位) で、サポートされていないファイルフォーマットの場合は -1 に設定されます。テーブルは row_groups に分割され、各グループには異なる row_id の行があります。
column_name	char(127)	クエリによって返された列の名前。

列名	データ型	説明
original_value	char(1024)	クエリされた元の値。
modified_value	char(1024)	クエリで指定されているデータ処理設定オプションに基づいて返された変更済みの値。
トリガー	char(128)	クエリで指定されているデータ処理オプション。
アクション	char(128)	クエリで指定されているデータ処理オプションに関連付けられたアクション。
action_value	char(128)	クエリで指定されているデータ処理オプションに関連付けられたアクションパラメータの値。
error_code	integer	クエリで指定されているデータ処理オプションの結果コード。

サンプルクエリ

次のクエリは、データ処理操作が実行された行のリストを返します。

```
SELECT * FROM sys_external_query_error;
```

クエリによって以下のような結果が返されます。

```
  user_id   query_id  file_location                                  rowid
 column_name          original_value           modified_value      trigger
 action                    action_value                  error_code
     100      1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
 league_name          Barclays Premier League    Barclays Premier Lea UNSPECIFIED
 TRUNCATE                                          156
     100      1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0
 league_nspi          34595                      32767               UNSPECIFIED
 OVERFLOW_VALUE                                    199
     100      1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1
 league_nspi          34151                      32767               UNSPECIFIED
 OVERFLOW_VALUE                                    199
```

```
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:2
 league_name             Barclays Premier League     Barclays Premier Lea UNSPECIFIED
 TRUNCATE                                             156
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:2
 league_nspi             33223                       32767                UNSPECIFIED
 OVERFLOW_VALUE                                       199
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:3
 league_name             Barclays Premier League     Barclays Premier Lea UNSPECIFIED
 TRUNCATE                                             156
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:3
 league_nspi             32808                       32767                UNSPECIFIED
 OVERFLOW_VALUE                                       199
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:4
 league_nspi             32790                       32767                UNSPECIFIED
 OVERFLOW_VALUE                                       199
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:5
 league_name             Spanish Primera Division    Spanish Primera Divi UNSPECIFIED
 TRUNCATE                                             156
     100      1574007    s3://spectrum-uddh/league/spi_global_rankings.0:6
 league_name             Spanish Primera Division    Spanish Primera Divi UNSPECIFIED
 TRUNCATE                                             156
```

SYS_INTEGRATION_ACTIVITY

SYS_INTEGRATION_ACTIVITY は、完了した統合実行に関する詳細を表示します。

SYS_INTEGRATION_ACTIVITY は、スーパーユーザーにのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

ゼロ ETL 統合の詳細については、「Amazon Redshift 管理ガイド」の「ゼロ ETL 統合の開始方法」を参照してください。

テーブルの列

列名	データ型	説明
integration_id	character (128)	統合に関連付けられている識別子です。
target_database	character (128)	統合データを受け取る Amazon Redshift のデータベース。

列名	データ型	説明
ソース	character (128)	統合のソースデータ。可能なデータタイプには、MySQL と PostgreSQL が含まれます。
checkpoint_name	character (128)	binlog 座標をレプリケートするチェックポイントの名前。
checkpoint_type	character (16)	チェックポイントのタイプ。可能な値は、snapshot、cdc です。
checkpoint_bytes	bigint	このチェックポイントのバイト数。
last_commit_timestamp	timestamp	このチェックポイントで最後にコミットされたときのタイムスタンプ。
modified_tables	integer	チェックポイントで変更されたテーブルの数。
integration_start_time	時系	このチェックポイントの統合が開始された時刻 (UTC)。
integration_end_time	時系	このチェックポイントの統合が終了した時刻 (UTC)。

サンプルクエリ

次の SQL コマンドは、統合のログを表示します。

```
select * from sys_integration_activity;

          integration_id              | target_database | source |
   checkpoint_name                  | checkpoint_type  | checkpoint_bytes |
 last_commit_timestamp    | modified_tables |    integration_start_time    |
 integration_end_time
--------------------------------------+-----------------+--------
+--------------------------------------------+------------------+------------------
+--------------------------+-----------------+----------------------------
+----------------------------
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1          | MySQL   | checkpoints/
checkpoint_3_241_3_510.json     |        cdc        |         762        | 2023-05-10
```

```
 23:00:14.201 |         1        | 2023-05-10 23:00:45.054265 | 2023-05-10
 23:00:46.339826
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1          | MySQL   | checkpoints/
checkpoint_3_16329_3_17839.json |         cdc         |        13488       | 2023-05-11
 01:33:57.411 |         2        | 2023-05-11 02:19:09.440121 | 2023-05-11
 02:19:16.090492
 76b15917-afae-4447-b7fd-08e2a5acce7b |   demo1          | MySQL   | checkpoints/
checkpoint_3_5103_3_5532.json   |         cdc         |         1657       | 2023-05-10
 23:13:14.205 |         2        | 2023-05-10 23:13:23.545487 | 2023-05-10
 23:13:25.652144
```

SYS_INTEGRATION_TABLE_STATE_CHANGE

SYS_INTEGRATION_TABLE_STATE_CHANGE は統合のテーブルステート変更ログの詳細を表示します。

スーパーユーザーは、このテーブルのすべての行を表示できます。

詳細については、「ゼロ ETL 統合での作業」を参照してください。

テーブルの列

列名	データ型	説明
integration_id	character (128)	統合に関連付けられている識別子です。
database_name	character (128)	Amazon Redshift データベースの名前。
schema_name	character (128)	Amazon Redshift スキーマの名前。
table_name	character (128)	テーブルの名前。
new_state	character (128)	テーブルの状態。使用できる値は、Synced、ResyncRequired 、ResyncInitiated 、Deleted、Failed、および ResyncDeleted です。

列名	データ型	説明
table_last_replicated_checkpoint	character (128)	現在同期されているログ座標。
state_change_reason	character (256)	最後の状態遷移の理由。
record_time	timestamp	このレコードが更新された時刻 (UTC)。

サンプルクエリ

次の SQL コマンドは、統合のログを表示します。

```
select * from sys_integration_table_state_change;

            integration_id             | database_name | schema_name | table_name
 | new_state |  table_last_replicated_checkpoint   | state_change_reason |
 record_time
--------------------------------------+---------------+-------------+------------
+-----------+-------------------------------------+---------------------
+----------------------------
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest79    |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:39:50.087868
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest56    |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:39:45.54005
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest50    |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:40:20.362504
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest80t3s | sbtest18    |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 19:40:32.544084
 99108e72-1cfd-414f-8cc0-0216acefac77 | perfdb        | sbtest40t3s | sbtest23    |
 Synced    | {"txn_seq":9834,"txn_id":126597515} |                     | 2023-09-20
 15:49:05.186209
```

SYS_LOAD_DETAIL

データのロードを追跡またはトラブルシューティングするための情報を返します。

このビューには、各データファイルがデータベーステーブルにロードされるのに合わせて進捗が記録されます。

このビューはすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	エントリを生成したユーザーの ID。
query_id	integer	クエリ ID。
file_name	character(256)	ロードするファイル名。
bytes_scanned	integer	Amazon S3 のファイルからスキャンされたバイト数。
lines_scanned	integer	ロードされたファイルからスキャンされた行数。この数は、実際にロードされた行数とは一致しない可能性があります。例えば、ロードによってスキャンが実行されたとき、COPY コマンドの MAXERROR オプションに基づいて、いくつかの不良レコードが許容される可能性があります。
record_time	timestamp	このエントリが最後に更新された時刻。
splits_scanned	このファイルの分割数。	このファイルの分割数。
start_time	timestamp	このファイル処理が開始した時刻。
end_time	timestamp	このファイル処理が終了した時刻。

サンプルクエリ

次の例は、前回の COPY 操作の詳細を返します。

```
select query_id, trim(file_name) as file, record_time
```

```
from sys_load_detail
where query_id = pg_last_copy_id();

 query_id |               file                |          record_time
----------+-----------------------------------+----------------------------
 28554    | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

次のクエリでは、TICKIT データベース内のテーブルについて、最新のロード状況を示すエントリが得られます。

```
select query_id, trim(file_name), record_time
from sys_load_detail
where file_name like '%tickit%' order by query_id;

 query_id |           btrim          |          record_time
----------+--------------------------+----------------------------
 22475    | tickit/allusers_pipe.txt | 2013-02-08 20:58:23.274186
 22478    | tickit/venue_pipe.txt    | 2013-02-08 20:58:25.070604
 22480    | tickit/category_pipe.txt | 2013-02-08 20:58:27.333472
 22482    | tickit/date2008_pipe.txt | 2013-02-08 20:58:28.608305
 22485    | tickit/allevents_pipe.txt| 2013-02-08 20:58:29.99489
 22487    | tickit/listings_pipe.txt | 2013-02-08 20:58:37.632939
 22593    | tickit/allusers_pipe.txt | 2013-02-08 21:04:08.400491
 22596    | tickit/venue_pipe.txt    | 2013-02-08 21:04:10.056055
 22598    | tickit/category_pipe.txt | 2013-02-08 21:04:11.465049
 22600    | tickit/date2008_pipe.txt | 2013-02-08 21:04:12.461502
 22603    | tickit/allevents_pipe.txt| 2013-02-08 21:04:14.785124
 22605    | tickit/listings_pipe.txt | 2013-02-08 21:04:20.170594

(12 rows)
```

レコードがこのシステムビューのログファイルに書き込まれていても、ロードがそれを含むトランザクションの一部として正しくコミットされているとは限りません。ロードのコミットを確認するには、STL_UTILITYTEXT ビューをクエリして、COPY トランザクションに対応する COMMIT レコードを探します。例えば、このクエリは、STL_UTILITYTEXT に対するサブクエリに基づいて SYS_LOAD_DETAIL と STL_QUERY を結合します。

```
select l.query_id,rtrim(l.file_name),q.xid
from sys_load_detail l, stl_query q
where l.query_id=q.query
and exists
```

```
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query_id |          rtrim            |  xid
----------+---------------------------+-------
 22600    | tickit/date2008_pipe.txt  | 68311
 22480    | tickit/category_pipe.txt  | 68066
  7508    | allusers_pipe.txt         | 23365
  7552    | category_pipe.txt         | 23415
  7576    | allevents_pipe.txt        | 23429
  7516    | venue_pipe.txt            | 23390
  7604    | listings_pipe.txt         | 23445
 22596    | tickit/venue_pipe.txt     | 68309
 22605    | tickit/listings_pipe.txt  | 68316
 22593    | tickit/allusers_pipe.txt  | 68305
 22485    | tickit/allevents_pipe.txt | 68071
  7561    | allevents_pipe.txt        | 23429
  7541    | category_pipe.txt         | 23415
  7558    | date2008_pipe.txt         | 23428
 22478    | tickit/venue_pipe.txt     | 68065
   526    | date2008_pipe.txt         |  2572
  7466    | allusers_pipe.txt         | 23365
 22482    | tickit/date2008_pipe.txt  | 68067
 22598    | tickit/category_pipe.txt  | 68310
 22603    | tickit/allevents_pipe.txt | 68315
 22475    | tickit/allusers_pipe.txt  | 68061
   547    | date2008_pipe.txt         |  2572
 22487    | tickit/listings_pipe.txt  | 68072
  7531    | venue_pipe.txt            | 23390
  7583    | listings_pipe.txt         | 23445
(25 rows)
```

SYS_LOAD_ERROR_DETAIL

SYS_LOAD_ERROR_DETAIL により、COPY コマンドで発生したエラーの詳細が表示されます。それぞれの行に 1 つの COPY コマンドが対応します。これには、実行中と終了した COPY コマンドの両方が含まれています。

SYS_LOAD_ERROR_DETAIL はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	コピーを送信したユーザーの ID。
query_id	bigint	コピーのクエリ ID。
transaction_id	bigint	トランザクション識別子。
session_id	integer	コピーを実行しているプロセスのプロセス ID。
database_name	character(64)	コピーが発行された時点でユーザーが接続されていたデータベースの名前。
table_id	integer	テーブル識別子。
start_time	timestamp	コピーが開始した時刻 (UTC)。
file_name	character(256)	ロードする入力ファイルへの完全なパス。
line_number	bigint	ロードファイル内のエラーが発生した行の番号。JSON ファイルをロードした場合は、エラーの発生した JSON オブジェクトの最終行の行番号。
column_name	character(127)	エラーが発生したフィールド。
column_type	character(10)	エラーの発生したフィールドのデータ型。

列名	データ型	説明
column_length	character(10)	定義されている場合、列の長さ。このフィールドは、データ型に長さの制限がある場合、値を持ちます。例えば、列でのデータ型が「character(3)」の場合、この列には値「3」が格納されます。
position	integer	フィールド内でのエラーの位置。
error_code	integer	エラーコードです。
error_message	character(512)	発生したエラーに関する説明。

サンプルクエリ

次のクエリは、特定のクエリに対するコピーコマンドのロードエラーの詳細を表示しています。

```
SELECT query_id,
       table_id,
       start_time,
       trim(file_name) AS file_name,
       trim(column_name) AS column_name,
       trim(column_type) AS column_type,
       trim(error_message) AS error_message
FROM sys_load_error_detail
WHERE query_id = 762949
ORDER BY start_time
LIMIT 10;
```

サンプル出力。

```
 query_id | table_id |          start_time          |                file_name
      | column_name | column_type |                  error_message
```

```
----------+----------+---------------------------
+-------------------------------------------+-------------+-------------
+-------------------------------------------------
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_000 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
   762949 |   137885 | 2022-02-15 22:14:46.759151 | s3://load-test/copyfail/
wrong_format_001 | id          | int4        | Invalid digit, Value 'a', Pos 0, Type:
 Integer
```

SYS_LOAD_HISTORY

SYS_LOAD_HISTORY を使用して、COPY コマンドの詳細を表示します。いくつかのフィールドの累積統計を含む COPY コマンドが、それぞれの行に対応して表示されます。この情報には、実行中と終了した COPY コマンドの両方が含まれます。

SYS_LOAD_HISTORY はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	コピーを送信したユーザーの ID。
query_id	bigint	コピーのクエリ ID。
transaction_id	bigint	トランザクション識別子。
session_id	integer	コピーを実行しているプロセスのプロセス ID。
database_name	text	対象のオペレーションが発行された時点で、ユーザーが接続されていたデータベースの名前。

列名	データ型	説明
status	text	コピーのステータス。有効な値は running、completed、abortedです。
table_name	text	コピー先のテーブル名。
start_time	timestamp	コピーが開始した時刻。
end_time	timestamp	コピーが完了した時刻。
duration	bigint	COPY コマンドが消費した時間 (マイクロ秒)。
data_source	text	コピーする入力ファイルのための Amazon S3 の場所。
file_format	text	ソースファイル形式。形式には、csv、txt、json、avro、orc、parquet が含まれます。
loaded_rows	bigint	テーブルにコピーされた行数。
loaded_bytes	bigint	テーブルにコピーされたバイト数。
source_file_count	integer	ソースファイルとして数えられるファイルの数。
source_file_bytes	bigint	ソースファイル内のバイト数。
file_count_scanned	integer	Amazon S3 のスキャンされたファイル数。
file_bytes_scanned	bigint	Amazon S3 のファイルからスキャンされたバイト数。

列名	データ型	説明
error_count	bigint	エラーの数。
copy_job_id	bigint	コピージョブ識別子。0 はジョブ ID がないことを示します。

サンプルクエリ

次のクエリは、特定のコピーコマンドのロードされた行、バイト、テーブル、データソースを表示します。

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
WHERE query_id IN (6389,490791,441663,74374,72297)
ORDER BY query_id,
         data_source DESC;
```

サンプル出力。

```
 query_id |    table_name     |                               data_source                              
              | loaded_rows | loaded_bytes
----------+------------------
+---------------------------------------------------------------------------+-------------
+---------------
     6389 | store_returns     | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
store_returns/    |   287999764 | 1196240296158
    72297 | web_site          | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
web_site/         |          54 |         43808
    74374 | ship_mode         | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
ship_mode/        |          20 |          1320
   441663 | income_band       | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
income_band/      |          20 |          2152
   490791 | customer_address  | s3://load-test/data-sources/tpcds/2.8.0/textfile/1T/
customer_address/ |     6000000 |     722924305
```

次のクエリは、コピーコマンドのロードされた行、バイト、テーブル、データソースを表示します。

```
SELECT query_id,
       table_name,
       data_source,
       loaded_rows,
       loaded_bytes
FROM sys_load_history
ORDER BY query_id DESC
LIMIT 10;
```

サンプル出力。

```
 query_id |       table_name       |                                 data_source
                          | loaded_rows |  loaded_bytes
----------+-----------------------
+-----------------------------------------------------------------------------
+-------------+-----------------
   491058 | web_site               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_site/              |          54 |           43808
   490947 | web_sales              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_sales/             |   720000376 |  22971988122819
   490923 | web_returns            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_returns/           |    71997522 |     96597496325
   490918 | web_page               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/web_page/              |        3000 |            1320
   490907 | warehouse              | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/warehouse/             |          20 |            1320
   490902 | time_dim               | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/time_dim/              |       86400 |            1320
   490876 | store_sales            | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_sales/           |  2879987999 | 151666241887933
   490870 | store_returns          | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store_returns/         |   287999764 |   1196405607941
   490865 | store                  | s3://load-test/data-sources/tpcds/2.8.0/
textfile/1T/store/                 |        1002 |          365507
```

次のクエリは、コピーコマンドの毎日ロードされる行とバイトを表示します。

```
SELECT date_trunc('day',start_time) AS exec_day,
       SUM(loaded_rows) AS loaded_rows,
       SUM(loaded_bytes) AS loaded_bytes
```

```
FROM sys_load_history
GROUP BY exec_day
ORDER BY exec_day DESC;
```

サンプル出力。

```
      exec_day       | loaded_rows |   loaded_bytes
---------------------+-------------+------------------
 2022-01-20 00:00:00 |  6347386005 |  258329473070606
 2022-01-19 00:00:00 | 19042158015 |  775198502204572
 2022-01-18 00:00:00 | 38084316030 | 1550294469446883
 2022-01-17 00:00:00 | 25389544020 | 1033271084791724
 2022-01-16 00:00:00 | 19042158015 |  775222736252792
 2022-01-15 00:00:00 | 19834245387 |  798122849155598
 2022-01-14 00:00:00 | 75376544688 | 3077040926571384
```

SYS_MV_REFRESH_HISTORY

結果には、すべてのマテリアライズドビューの更新履歴に関する情報が含まれます。結果には、手動や自動などの更新タイプ、および最新の更新のステータスが含まれます。

SYS_MV_REFRESH_HISTORY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	更新を送信したユーザーの識別子。
session_id	integer	マテリアライズドビューの更新を実行するプロセスのプロセス識別子。
transaction_id	bigint	トランザクション識別子。
database_name	char(128)	マテリアライズドビューを含むデータベース。

列名	データ型	説明
schema_name	char(128)	マテリアライズドビューのスキーマ。
mv_id	bigint	マテリアライズドビューのオブジェクト ID。
mv_name	char(128)	マテリアライズドビューの名前。
refresh_type	char(32)	更新のタイプ (手動または自動など)。
status	text	更新のステータス。ステータスの詳細については、SVL_MV_REFRESH_STATUS のステータス列を参照してください。
start_time	timestamp	更新の開始時間。
end_time	timestamp	更新の終了時間。
duration	bigint	マテリアライズドビューの更新にかかった時間 (マイクロ秒単位)。

サンプルクエリ

次のクエリは、マテリアライズドビューの更新履歴を表示します。

```
SELECT user_id,
     session_id,
     transaction_id,
     database_name,
     schema_name,
     mv_id,
     mv_name,
     refresh_type,
```

```
    status,
    start_time,
    end_time,
    duration
    from sys_mv_refresh_history;
```

このクエリは、次のサンプル出力を返します。

```
 user_id | session_id | transaction_id | database_name | schema_name                |
 mv_id  |  mv_name            |  refresh_type  |  status
        |  start_time                 |  end_time                   |  duration
---------+------------+----------------+---------------
+----------------------------+--------+--------------------+----------------
+---------------------------------------------------+---------------------------
+----------------------------+-----------
       1 | 1073815659 |          15066 | dev           | test_stl_mv_refresh_schema |
 203762 | mv_incremental     | Manual         | MV was already updated
        | 2023-10-26 15:59:20.952179 | 2023-10-26 15:59:20.952866 |      687
       1 | 1073815659 |          15068 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | MV was already updated
        | 2023-10-26 15:59:21.008049 | 2023-10-26 15:59:21.008658 |      609
       1 | 1073815659 |          15070 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | MV was already updated
        | 2023-10-26 15:59:21.064252 | 2023-10-26 15:59:21.064885 |      633
       1 | 1073815659 |          15074 | dev           | test_stl_mv_refresh_schema
 | 203762 | mv_incremental     | Manual         | Refresh successfully updated MV
 incrementally   | 2023-10-26 15:59:29.693329 | 2023-10-26 15:59:43.482842 | 13789513
       1 | 1073815659 |          15076 | dev           | test_stl_mv_refresh_schema |
 203771 | mv_nonincremental  | Manual         | Refresh successfully recomputed MV from
 scratch | 2023-10-26 15:59:43.550184 | 2023-10-26 15:59:47.880833 |  4330649
       1 | 1073815659 |          15078 | dev           | test_stl_mv_refresh_schema |
 203779 | mv_refresh_error   | Manual         | Refresh failed due to an internal error
        | 2023-10-26 15:59:47.949052 | 2023-10-26 15:59:52.494681 |  4545629
(6 rows)
```

SYS_MV_STATE

結果には、すべてのマテリアライズドビューの状態に関する情報が含まれます。ベーステーブル情報、スキーマのプロパティ、最近のイベント (列のドロップなど) に関する情報が含まれます。

SYS_MV_STATE は、すべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	bigint	イベントを作成したユーザーの ID。
transaction_id	bigint	イベントのトランザクション ID。
database_name	char(128)	マテリアライズドビューを含むデータベース。
event_desc	char(500)	ステータスの変更を促したイベント。値の例は次のとおりです。 • 列のタイプが変更されました • 列が削除されました • 列の名前が変更されました • スキーマ名が変更されました • 小さいテーブルの変換 • TRUNCATE • Vacuum この列には他にも指定できる値があることに注意してください。
start_time	timestamp	イベントの開始時間。

列名	データ型	説明
base_table_database_name	char(128)	ベーステーブルのデータベース名。
base_table_schema	char(128)	ベーステーブルのスキーマ。
base_table_name	char(128)	ベーステーブルの名前。
mv_schema	char(128)	マテリアライズドビューのスキーマ。
mv_name	char(128)	マテリアライズドビューの名前。
state	character(32)	マテリアライズドビューの変更された状態は以下のとおりです。 • 再計算 • 更新不可

サンプルクエリ

次のクエリは、マテリアライズドビューの状態を示します。

```
select * from sys_mv_state;
```

このクエリは、次のサンプル出力を返します。

```
 user_id | transaction_id | database_name | event_desc                    | start_time
              | base_table_database_name | base_table_schema | base_table_name     |
 mv_schema   | mv_name                    | state
---------+----------------+---------------+-----------------------------
+----------------------------+---------------------------+-------------------
+---------------------+-------------+-----------------------------+--------------
 106     | 12720          | tickit_db     | TRUNCATE                      | 2023-07-26
 14:59:12.788268 | tickit_db                | mv_schema         | test_table_95d6d861 |
 mv_schema    | materialized_view_a1f3f862 | Recompute
```

```
106     | 12724          | tickit_db     | ALTER TABLE ALTER DISTSTYLE | 2023-07-26
14:59:51.409014 | tickit_db                   | mv_schema          | test_table_58102435 |
mv_schema   | materialized_view_ca746631 | Recompute
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.822928 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Unrefreshable
106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
15:00:08.051244 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Unrefreshable
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.857755 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Unrefreshable
106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
15:00:08.051358 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5ef0d754 | Unrefreshable
106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
14:59:12.788159 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5750a8d4 | Recompute
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.857799 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
106     | 12720          | tickit_db     | TRUNCATE                    | 2023-07-26
14:59:12.788327 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_5ef0d754 | Recompute
106     | 12727          | tickit_db     | ALTER TABLE ALTER SORTKEY   | 2023-07-26
15:00:08.006235 | tickit_db                   | mv_schema          | test_table_58102435 |
mv_schema   | materialized_view_ca746631 | Recompute
106     | 12720          | tickit_db     | Column was renamed          | 2023-07-26
14:59:12.82297  | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
106     | 12727          | tickit_db     | Table was renamed           | 2023-07-26
15:00:08.051321 | tickit_db                   | mv_schema          | test_table_95d6d861 |
mv_schema   | materialized_view_a1f3f862 | Unrefreshable
```

SYS_PROCEDURE_CALL

SYS_PROCEDURE_CALL ビューを使用して、ストアドプロシージャ呼び出しに関する情報として、開始時刻、終了時刻、ストアドプロシージャ呼び出しのステータス、ネストされたストアドプロシージャ呼び出しの呼び出し階層などを取得できます。ストアドプロシージャ呼び出しごとにクエリ ID を受け取ります。

SYS_PROCEDURE_CALL はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
session_user_id	integer	セッションの作成元であり、最上位ストアドプロシージャ呼び出しの呼び出し元であるユーザーの ID。
security_user_id	integer	ストアドプロシージャ内でステートメントを実行するために使用された権限を持つユーザーの ID。ストアドプロシージャが DEFINER の場合、これがストアドプロシージャの所有者 user_id になります。
query_id	integer	ストアドプロシージャ呼び出しのクエリ ID。
query_text	char(4000)	ストアドプロシージャ呼び出しのクエリのテキスト。
start_time	timestamp	クエリの実行が開始した時刻 (UTC)。例えば、タイムスタンプは、秒の小数部に 6 桁の精度を使用します。2009-06-12 11:29:19.131358.
end_time	timestamp	クエリの実行が終了した時刻 (UTC)。タイムスタンプは、秒の小数部に 6 桁の精度を使用します。例: 2009-06-12 11:29:19.131358。

列名	データ型	説明
status	char(10)	ストアドプロシージャ呼び出しのステータス。ストアドプロシージャがシステムによって停止されたか、ユーザーによってキャンセルされた場合、値は canceled です。ストアドプロシージャ呼び出しが最後まで実行された場合、値は success です。
caller_procedure _query_id	integer	ストアドプロシージャ呼び出しが別のストアドプロシージャ呼び出しによって呼び出された場合、この列は外部呼び出しのクエリ ID になります。それ以外の場合、フィールドは NULL です。

サンプルクエリ

次のクエリは、ネストされたストアドプロシージャ呼び出し階層を返します。

```
select query_id, datediff(seconds, start_time, end_time) as elapsed_time, status,
 trim(query_text) as call, caller_procedure_query_id from sys_procedure_call;
```

サンプル出力。

```
 query_id | elapsed_time | status  |                        call                        |
 caller_procedure_query_id
----------+--------------+---------+----------------------------------------------------
+---------------------------
     3087 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d(1)    |
          3085
     3085 |           18 | success | CALL proc_bd906c98c45443ffa165e9552056902d_2(1); |

(2 rows)
```

SYS_PROCEDURE_MESSAGES

SYS_PROCEDURE_MESSAGES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
transaction_id	bigint	トランザクション識別子。
query_id	integer	ストアドプロシージャ呼び出しのクエリ ID。
record_time	timestamp	メッセージが生成された時刻 (UTC)。
log_level	char(10)	生成されたメッセージのログレベル。指定できる値は LOG、INFO、NOTICE、WARNING、EXCEPTION です。
message	char(1024)	生成されたメッセージのテキスト。
line_number	integer	生成されたメッセージの行番号。

サンプルクエリ

次のクエリは、SYS_PROCEDURE_MESSAGES のサンプル出力を示しています。

```
select transaction_id, query_id, record_time, log_level, trim(message), line_number
 from sys_procedure_messages;
```

```
transaction_id | query_id |          record_time          | log_level |            btrim
        | line_number
```

```
---------------+----------+---------------------------+-----------
+--------------------------+-------------
     25267     |   80562  | 2023-07-17 14:38:31.910136 |   NOTICE  |
 test_notice_msg_b9f1e749  |      8
     25267     |   80562  | 2023-07-17 14:38:31.910002 |    LOG    |
 test_log_msg_833c7420     |     6
     25267     |   80562  | 2023-07-17 14:38:31.910111 |    INFO   |
 test_info_msg_651373d9    |     7
     25267     |   80562  | 2023-07-17 14:38:31.910154 |   WARNING |
 test_warning_msg_831c5747 |      9
(4 rows)
```

SYS_QUERY_DETAIL

SYS_QUERY_DETAIL により、ステップレベルでクエリの詳細を表示できます。各行が特定の WLM クエリのステップを表し、詳細情報が含まれます、このビューには、DDL、DML、ユーティリティコマンド (コピーおよびアンロード) など、多くの種類のクエリが含まれています。クエリの種類によっては、一部の列との関連がない場合があります。例えば、external_scanned_bytes は内部テーブルには関係しません。

SYS_QUERY_DETAIL はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを送信したユーザーの ID。
query_id	bigint	クエリ識別子。
child_query_sequence	integer	書き換えられたユーザークエリのシーケンス。
stream_id	integer	クエリストリームのストリーム識別子。

列名	データ型	説明
segment_id	integer	クエリ実行セグメントのセグメント識別子。
step_id	integer	セグメント内のステップ識別子。
step_name	text	セグメント内のステップ名。指定できる値は、`aggregate` 、`broadcast` 、`delete`、`distribute` 、`hash`、`hashjoin`、`insert`、`lin` 、`unique`、`window` です。
table_id	integer	永続テーブルスキャン時のテーブル識別子。
table_name	character(136)	実行中のステップでのテーブル名。
is_rrscan	文字	ステップがスキャンステップであるかどうかを示す値。true (t) の場合は、そのステップで範囲が限定されたスキャンが実行されたことを示します。
start_time	timestamp	クエリステップが開始された時刻。
end_time	timestamp	クエリステップが完了した時刻。
duration	bigint	そのステップに消費された時間 (マイクロ秒)。
アラート	text	アラートイベントの説明。

列名	データ型	説明
input_bytes	bigint	現在のステップの入力バイト数。
input_rows	bigint	現在のステップの入力行数。
output_bytes	bigint	現在のステップの出力バイト数。
output_rows	bigint	現在のステップの出力行数。
blocks_read	bigint	ステップが読み込んだブロックの数。
blocks_write	bigint	ステップが書き込んだブロックの数。
local_read_IO	bigint	ローカルディスクキャッシュから読み取られるブロック数。
remote_read_IO	bigint	リモートから読み込んだブロックの数。
ソース	text	スキャンされたデータベースオブジェクトの型。この列が値を持つのは、行の step_name 値が scan である場合のみです。
data_skewness	integer	すべてのステップ間における出力行の分布の歪み。0% から 100% までの範囲の数値です。数値が大きいほど、分布のバランスに歪みがあります。

列名	データ型	説明
time_skewness	integer	すべてのステップ間における実行時間分布の歪み。0% から 100% までの範囲の数値です。数値が大きいほど、分布のバランスに歪みがあります。
is_active	文字	ステップレベルでのクエリの状態。可能な値は、ステップがアクティブに実行中であることを示す「t」、ステップの実行が完了したことを示す「f」です。
spilled_block_local_disk	bigint	ローカルディスクに流出したブロックの数。
spilled_block_remote_disk	bigint	Amazon Simple Storage Service に流出したブロックの数。
step_attribute	character(64)	関連のステップに関する情報が含まれます。スキャンステップに指定できる値: `multi-dimensional` 。

サンプルクエリ

次の例では、SYS_QUERY_DETAIL の出力を返します。

次のクエリは、ステップ名、input_bytes、output_bytes、input_rows、output_rows を含め、ステップレベルでクエリメタデータの詳細を表示します。

```
SELECT query_id,
       child_query_sequence,
       stream_id,
```

```
       segment_id,
       step_id,
       trim(step_name) AS step_name,
       duration,
       input_bytes,
       output_bytes,
       input_rows,
       output_rows
FROM sys_query_detail
WHERE query_id IN (193929)
ORDER BY query_id,
         stream_id,
         segment_id,
         step_id DESC;
```

サンプル出力。

```
 query_id | child_query_sequence | stream_id | segment_id | step_id | step_name  |
 duration     | input_bytes | output_bytes | input_rows | output_rows
----------+----------------------+-----------+------------+---------+------------
+-----------------+-------------+--------------+------------+-------------
   193929 |                    2 |         0 |          0 |       3 | hash       |
    37144 |           0 |      9350272 |          0 |      292196
   193929 |                    5 |         0 |          0 |       3 | hash       |
     9492 |           0 |        23360 |          0 |        1460
   193929 |                    1 |         0 |          0 |       3 | hash       |
    46809 |           0 |      9350272 |          0 |      292196
   193929 |                    4 |         0 |          0 |       2 | return     |
     7685 |           0 |          896 |          0 |         112
   193929 |                    1 |         0 |          0 |       2 | project    |
    46809 |           0 |            0 |          0 |      292196
   193929 |                    2 |         0 |          0 |       2 | project    |
    37144 |           0 |            0 |          0 |      292196
   193929 |                    5 |         0 |          0 |       2 | project    |
     9492 |           0 |            0 |          0 |        1460
   193929 |                    3 |         0 |          0 |       2 | return     |
    11033 |           0 |        14336 |          0 |         112
   193929 |                    2 |         0 |          0 |       1 | project    |
    37144 |           0 |            0 |          0 |      292196
   193929 |                    1 |         0 |          0 |       1 | project    |
    46809 |           0 |            0 |          0 |      292196
   193929 |                    5 |         0 |          0 |       1 | project    |
     9492 |           0 |            0 |          0 |        1460
```

```
   193929 |                      3 |          0 |           0 |         1 | aggregate  |
      11033 |           0 |     201488 |           0 |        14
   193929 |                      4 |          0 |           0 |         1 | aggregate  |
       7685 |           0 |      28784 |           0 |        14
   193929 |                      5 |          0 |           0 |         0 | scan       |
       9492 |           0 |      23360 |      292196 |      1460
   193929 |                      4 |          0 |           0 |         0 | scan       |
       7685 |           0 |       1344 |         112 |       112
   193929 |                      2 |          0 |           0 |         0 | scan       |
      37144 |           0 |    7304900 |      292196 |    292196
   193929 |                      3 |          0 |           0 |         0 | scan       |
      11033 |           0 |      13440 |         112 |       112
   193929 |                      1 |          0 |           0 |         0 | scan       |
      46809 |           0 |    7304900 |      292196 |    292196
   193929 |                      5 |          0 |           0 |        -1 |            |
       9492 |       12288 |          0 |           0 |         0
   193929 |                      1 |          0 |           0 |        -1 |            |
      46809 |       16384 |          0 |           0 |         0
   193929 |                      2 |          0 |           0 |        -1 |            |
      37144 |       16384 |          0 |           0 |         0
   193929 |                      4 |          0 |           0 |        -1 |            |
       7685 |       28672 |          0 |           0 |         0
   193929 |                      3 |          0 |           0 |        -1 |            |
      11033 |      114688 |          0 |           0 |         0
```

データベース内のテーブルを、使用頻度の高い順に表示するには、次の例を使用します。*sample_data_dev* を独自のデータベースに置き換えます。このクエリはクラスターの作成時に開始されるクエリをカウントしますが、データウェアハウスのスペースが不足している場合はシステムビューデータが保存されないことに注意してください。

```
SELECT table_name, COUNT (DISTINCT query_id)
FROM SYS_QUERY_DETAIL
WHERE table_name LIKE 'sample_data_dev%'
GROUP BY table_name
ORDER BY COUNT(*) DESC;

+---------------------------------+-------+
|           table_name            | count |
+---------------------------------+-------+
| sample_data_dev.tickit.venue    |     4 |
| sample_data_dev.myunload1.venue |     3 |
| sample_data_dev.tickit.listing  |     1 |
| sample_data_dev.tickit.category |     1 |
```

```
| sample_data_dev.tickit.users     |      1 |
| sample_data_dev.tickit.date      |      1 |
| sample_data_dev.tickit.sales     |      1 |
| sample_data_dev.tickit.event     |      1 |
+----------------------------------+-------+
```

SYS_QUERY_HISTORY

ユーザークエリーの詳細表示には、SYS_QUERY_HISTORY を使用します。いくつかのフィールドに関する累積統計を含むユーザークエリが、それぞれの行により表されます。このビューには、データ定義言語 (DDL)、データ操作言語 (DML)、コピー、アンロード、Amazon Redshift Spectrum など、さまざまな種類のクエリが含まれています。ここには、実行中のクエリと終了したクエリの両方が表示されます。

SYS_QUERY_HISTORY は、すべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを送信したユーザーの ID。
query_id	bigint	クエリ識別子。
query_label	character(320)	クエリ名の短縮形。
transaction_id	bigint	トランザクション識別子。
session_id	integer	クエリを実行しているプロセスのプロセス識別子。
database_name	character(128)	クエリが発行されたときにユーザーが接続されたデータベースの名前。
query_type	character(32)	クエリのタイプ (SELECT、INSERT、UPDATE、UNLOA

列名	データ型	説明
		D、COPY、COMMAND、DDL、UTILITY、CTAS、OTHER など)。
status	character(10)	クエリのステータス。有効な値: 計画 (planning)、キュー済み (queued)、実行中 (running)、終了中 (returning)、失敗 (failed)、キャンセル済み (canceled)、成功 (success)。
result_cache_hit	ブール値	クエリが結果キャッシュと一致するかどうかを示します。
start_time	timestamp	クエリが開始された時刻。
end_time	timestamp	クエリが完了した時刻。
elapsed_time	bigint	クエリの実行が消費した合計時間 (マイクロ秒)。
queue_time	bigint	サービスクラスクエリのキューが消費した合計時間 (マイクロ秒)。
execution_time	bigint	サービスクラスで実行されている合計時間 (マイクロ秒)。
error_message	character(512)	クエリが失敗した理由。
returned_rows	bigint	クライアントに返された行の数。
returned_bytes	bigint	クライアントに返されたバイト数。

列名	データ型	説明
query_text	character(4000)	クエリ文字列。この文字列は切り詰められることがあります。
redshift_version	character(256)	クエリ実行時の Amazon Redshift のバージョン。
usage_limit	character(150)	クエリによって到達した使用制限 ID のリスト。
compute_type	varCHAR(32)	クエリがメインクラスターまたは同時実行スケーリングクラスターのどちらで実行されるかを示します。指定できる値は、`primary` (クエリはメインクラスターで実行)、`secondary` (クエリはセカンダリクラスターで実行)、または `primary-scale` (クエリは同時実行クラスターで実行) です。これはプロビジョニングされたクラスターにのみ適用されます。
compile_time	bigint	クエリのコンパイルに費やされた合計時間 (マイクロ秒)。
planning_time	bigint	クエリのプランニングに費やされた合計時間 (マイクロ秒)。
lock_wait_time	bigint	リレーションロックを待機して費やされた合計時間 (マイクロ秒)。

列名	データ型	説明
service_class_id	integer	サービスクラスの ID サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。 この列は、プロビジョニングされたクラスターで実行されるクエリにのみ使用されます。Redshift Serverless で実行されるクエリの場合、この列には -1 が含まれます。
service_class_name	character(64)	サービスクラス名。 この列は、プロビジョニングされたクラスターで実行されるクエリにのみ使用されます。Amazon Redshift Serverless で実行されるクエリの場合、この列は空です。

列名	データ型	説明
query_priority	character(20)	クエリが実行されたキューの優先度。可能な値は以下のとおりです。 • NULL • lowest • low • 正常 • high • highest NULL は、そのクエリではクエリの優先度がサポートされていないことを意味します。 この列は、プロビジョニングされたクラスターで実行されるクエリにのみ使用されます。Redshift Serverless で実行されるクエリの場合、この列は空です。

列名	データ型	説明
short_query_accelerated	character(10)	ショートクエリアクセラレーション (SQA) を使用してクエリが高速化されたかどうか。可能な値は以下のとおりです。 • true • false • NULL この列は、プロビジョニングされたクラスターで実行されるクエリにのみ使用されます。Redshift Serverless で実行されるクエリの場合、この列は空です。
user_query_hash	character(40)	クエリリテラルを含む、クエリから生成されたクエリハッシュ。同じクエリテキストを持つ繰り返しクエリは、同じ user_query_hash 値を持ちます。
generic_query_hash	character(40)	クエリリテラルを除く、クエリから生成されたクエリハッシュ。同じクエリテキストを持つが、クエリリテラルが異なる繰り返しクエリは、同じ generic_query_hash 値を持ちます。
query_hash_version	integer	クエリから生成されたクエリハッシュのバージョン番号。

サンプルクエリ

次のクエリは、実行中とキューに登録されたクエリを返します。

```
SELECT user_id,
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time
FROM sys_query_history
WHERE status IN ('running','queued')
ORDER BY start_time;
```

サンプル出力。

```
 user_id | query_id | transaction_id | session_id | status  | database_name |
 start_time          |           end_time           | result_cache_hit | elapsed_time |
 queue_time | execution_time
---------+----------+----------------+------------+---------+---------------
+----------------------------+----------------------------+------------------
+--------------+------------+---------------
     101 |   760705 |         852337 | 1073832321 | running | tpcds_1t      |
 2022-02-15 19:03:19.67849 | 2022-02-15 19:03:19.739811 | f                |
 61321 |          0 |              0
```

次のクエリは、特定のクエリのクエリ開始時間、終了時間、キュー時間、経過時間、プランニング時間、およびその他のメタデータを返します。

```
SELECT user_id,
       query_id,
       transaction_id,
       session_id,
       status,
       trim(database_name) AS database_name,
       start_time,
```

```
       end_time,
       result_cache_hit,
       elapsed_time,
       queue_time,
       execution_time,
       planning_time,
       trim(query_text) as query_text
FROM sys_query_history
WHERE query_id = 3093;
```

サンプル出力。

```
user_id | query_id | transaction_id | session_id |   status   | database_name |
   start_time          |          end_time          | result_cache_hit | elapsed_time |
 queue_time | execution_time | planning_time | query_text
--------+----------+----------------+------------+------------
+---------------+----------------------------+----------------------------
+------------------+--------------+------------+---------------+---------------
+-------------------------------------
    106 |     3093 |          11759 | 1073750146 | success    | dev           |
 2023-03-16 16:53:17.840214 | 2023-03-16 16:53:18.106588 | f                |
 266374 |          0 |         105725 |        136589 | select count(*) from item;
```

次のクエリは、最近実行された 10 件の SELECT クエリを一覧表示します。

```
SELECT query_id,
       transaction_id,
       session_id,
       start_time,
       elapsed_time,
       queue_time,
       execution_time,
       returned_rows,
       returned_bytes
FROM sys_query_history
WHERE query_type = 'SELECT'
ORDER BY start_time DESC limit 10;
```

サンプル出力。

```
 query_id | transaction_id | session_id |         start_time         | elapsed_time |
 queue_time | execution_time | returned_rows | returned_bytes
----------+----------------+------------+----------------------------+--------------
+------------+----------------+---------------+----------------
   526532 |          61093 | 1073840313 | 2022-02-09 04:43:24.149603 |       520571 |
        0 |         481293 |              1 |           3794
   526520 |          60850 | 1073840313 | 2022-02-09 04:38:27.24875  |       635957 |
        0 |         596601 |              1 |           3679
   526508 |          60803 | 1073840313 | 2022-02-09 04:37:51.118835 |       563882 |
        0 |         503135 |              5 |          17216
   526505 |          60763 | 1073840313 | 2022-02-09 04:36:48.636224 |       649337 |
        0 |         589823 |              1 |            652
   526478 |          60730 | 1073840313 | 2022-02-09 04:36:11.741471 |     14611321 |
        0 |       14544058 |              0 |              0
   526467 |          60636 | 1073840313 | 2022-02-09 04:34:11.91463  |     16711367 |
        0 |       16633767 |              1 |            575
   511617 |         617946 | 1074009948 | 2022-01-20 06:21:54.44481  |      9937090 |
        0 |        9899271 |            100 |          12500
   511603 |         617941 | 1074259415 | 2022-01-20 06:21:45.71744  |      8065081 |
        0 |        7582500 |            100 |           8889
   511595 |         617935 | 1074128320 | 2022-01-20 06:21:44.030876 |      1051270 |
        0 |        1014879 |              1 |             72
   511584 |         617931 | 1074030019 | 2022-01-20 06:21:42.764088 |       609033 |
        0 |         485887 |            100 |           8438
```

次のクエリは、毎日実行される選択クエリ数と平均クエリ経過時間を表示しています。

```
SELECT date_trunc('day',start_time) AS exec_day,
       status,
       COUNT(*) AS query_cnt,
       AVG(datediff (microsecond,start_time,end_time)) AS elapsed_avg
FROM sys_query_history
WHERE query_type = 'SELECT'
AND start_time >= '2022-01-14'
AND start_time <= '2022-01-18'
GROUP BY exec_day,
         status
ORDER BY exec_day,
         status;
```

サンプル出力。

```
      exec_day        | status   | query_cnt | elapsed_avg
```

```
---------------------+---------+-----------+------------
 2022-01-14 00:00:00 | success |      5253 |  56608048
 2022-01-15 00:00:00 | success |      7004 |  56995017
 2022-01-16 00:00:00 | success |      5253 |  57016363
 2022-01-17 00:00:00 | success |      5309 |  55236784
 2022-01-18 00:00:00 | success |      8092 |  54355124
```

次のクエリは、毎日実行されるクエリ経過時間のパフォーマンスを表示しています。

```
SELECT distinct date_trunc('day',start_time) AS exec_day,
       query_count.cnt AS query_count,
       Percentile_cont(0.5) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P50_runtime,
       Percentile_cont(0.8) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P80_runtime,
       Percentile_cont(0.9) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P90_runtime,
       Percentile_cont(0.99) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS P99_runtime,
       Percentile_cont(1.0) within group(ORDER BY elapsed_time) OVER (PARTITION BY
 exec_day) AS max_runtime
FROM sys_query_history
LEFT JOIN (SELECT  date_trunc('day',start_time) AS day, count(*) cnt
           FROM sys_query_history
           WHERE query_type = 'SELECT'
           GROUP by 1) query_count
ON date_trunc('day',start_time) = query_count.day
WHERE query_type = 'SELECT'
ORDER BY exec_day;
```

サンプル出力。

```

      exec_day       | query_count | p50_runtime | p80_runtime | p90_runtime |
 p99_runtime  | max_runtime
---------------------+-------------+-------------+-------------+-------------
+--------------+-------------
 2022-01-14 00:00:00 |        5253 |  16816922.0 |  69525096.0 | 158524917.8 |
 486322477.52 | 1582078873.0
 2022-01-15 00:00:00 |        7004 |  15896130.5 |  71058707.0 | 164314568.9 |
 500331542.07 | 1696344792.0
 2022-01-16 00:00:00 |        5253 |  15750451.0 |  72037082.2 | 159513733.4 |
 480372059.24 | 1594793766.0
```

```
 2022-01-17 00:00:00 |        5309 |  15394513.0 |  68881393.2 | 160254700.0 |
 493372245.84 | 1521758640.0
 2022-01-18 00:00:00 |        8092 |  15575286.5 |  68485955.4 | 154559572.5 |
 463552685.39 | 1542783444.0
 2022-01-19 00:00:00 |        5860 |  16648747.0 |  72470482.6 | 166485138.2 |
 492038228.67 | 1693483241.0
 2022-01-20 00:00:00 |        1751 |  15422072.0 |  69686381.0 | 162315385.0 |
 497066615.00 | 1439319739.0
 2022-02-09 00:00:00 |          13 |   6382812.0 |  17616161.6 |  21197988.4 |
 23021343.84 |   23168439.0
```

次のクエリは、クエリタイプの分散を表示しています。

```
SELECT query_type,
       COUNT(*) AS query_count
FROM sys_query_history
GROUP BY query_type
ORDER BY query_count DESC;
```

サンプル出力。

```
 query_type | query_count
------------+-------------
 UTILITY    |      134486
 SELECT     |       38537
 DDL        |        4832
 OTHER      |         768
 LOAD       |         768
 CTAS       |         748
 COMMAND    |          92
```

次の例は、複数のクエリ間のクエリハッシュ結果の違いを示しています。以下のクエリを確認します。

```
CREATE TABLE test_table (col1 INT);

INSERT INTO test_table VALUES (1),(2);

SELECT * FROM test_table;

SELECT * FROM test_table;
```

```
SELECT col1 FROM test_table;

SELECT * FROM test_table WHERE col1=1;

SELECT * FROM test_table WHERE col1=2;

SELECT query_id, TRIM(user_query_hash) AS user_query_hash, TRIM(generic_query_hash)
 AS generic_query_hash, TRIM(query_text) AS text FROM sys_query_history ORDER BY
 start_time
DESC LIMIT 10;
```

以下は出力例です。

```
query_id | user_query_hash | generic_query_hash | text
---------+-----------------+--------------------+----------
24723049 | oPuFtjEPLTs=    | oPuFtjEPLTs=       | select query_id,
 trim(user_query_hash) as user_query_hash, trim(generic_query_hash) as
 generic_query_hash, query_hash_version, trim(query_text) as text from
 sys_query_history order by start_time\r\ndesc limit 20
24723045 | Gw2Kwdd8m2I=    | IwfRu8/XAKI=       | select * from test_table where col1=2
 limit 100
24723041 | LNw2vx0GDXo=    | IwfRu8/XAKI=       | select * from test_table where col1=1
 limit 100
24723036 | H+qep/c82Y8=    | H+qep/c82Y8=       | select col1 from test_table limit 100
24723033 | H+qep/c82Y8=    | H+qep/c82Y8=       | select * from test_table limit 100
24723029 | H+qep/c82Y8=    | H+qep/c82Y8=       | select * from test_table limit 100
24723023 | 50sirx9E1hU=    | uO36Z1a/QYs=       | insert into test_table values (1),(2)
24723021 | YSVnlivZHeo=    | YSVnlivZHeo=       | create table test_table (col1 int)
```

test_table には 1 つの列しかないため、`SELECT * FROM test_table;` と `SELECT col1 FROM test_table;` は同じ user_query_hash 値を持ちます。`SELECT * FROM test_table WHERE col1=1;` と `SELECT * FROM test_table WHERE col1=2;` はクエリリテラル 1 と 2 の外で同じであるため、user_query_hash 値は異なりますが、同じ generic_query_hash 値を持ちます。

SYS_QUERY_TEXT

SYS_QUERY_TEXT を使用すると、すべてのクエリのクエリテキストを表示できます。各行は、シーケンス番号 0 から始まる最大 4000 文字のクエリのクエリテキストを表します。クエリステートメントに 4000 文字を超える文字が含まれている場合は、各行のシーケンス番号を増やすことで、そのステートメントの追加の行がログに記録されます。このビューでは、DDL、ユーティリ

ティ、Amazon Redshift クエリ、リーダーノードのみのクエリなど、すべてのユーザークエリテキストがログに記録されます。

SYS_QUERY_TEXT はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを送信したユーザーの ID。
query_id	bigint	クエリ識別子。
transaction_id	bigint	ステートメントに関連付けられているトランザクション ID。
session_id	integer	クエリを実行しているセッションのプロセス識別子。
start_time	timestamp	クエリが開始される時刻。
sequence	integer	1 つのステートメントに含まれる文字数が 4000 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
text	character (4000)	4000 文字単位の SQL クエリのテキスト。このフィールドには、バックスラッシュ (\\) や改行 (\\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

次のクエリは、実行中とキューに登録されたクエリを返します。

```
SELECT user_id,
 query_id,
 transaction_id,
 session_id, start_time,
 sequence, trim(text) as text from sys_query_text
 ORDER BY sequence;
```

サンプル出力。

```
 user_id | query_id | transaction_id | session_id |        start_time          |
 sequence |                                                             text
--------+----------+----------------+------------+---------------------------
+----------
+-------------------------------------------------------------------------------------
   100  |     4    |       1396      | 1073750220 | 2023-04-28 16:44:55.887184 |
   0    | SELECT trim(text) as text, sequence FROM sys_query_text WHERE query_id =
 pg_last_query_id() AND user_id > 1 AND start
_time > '2023-04-28 16:44:55.922705+00:00'::timestamp order by sequence;
```

次のクエリは、データベース内のグループに付与されたアクセス許可、またはグループから取り消されたアクセス許可を返します。

```
SELECT
    SPLIT_PART(text, ' ', 1) as grantrevoke,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'GROUP'))), ' ', 2) as group,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), ' '))), 'ON', 1) as type,
    SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 2) || ' ' ||
 SPLIT_PART((SUBSTRING(text, STRPOS(UPPER(text), 'ON'))), ' ', 3) as entity
FROM SYS_QUERY_TEXT
WHERE (text LIKE 'GRANT%' OR text LIKE 'REVOKE%') AND text LIKE '%GROUP%';

+-------------+----------+--------+----------+
| grantrevoke |  group   |  type  |  entity  |
+-------------+----------+--------+----------+
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | SELECT | TABLE t1 |
| GRANT       | bi_group | USAGE  | TABLE t1 |
```

```
| GRANT        | bi_group | SELECT | TABLE t1 |
| GRANT        | bi_group | SELECT | TABLE t1 |
+-------------+----------+--------+----------+
```

SYS_RESTORE_LOG

SYS_RESTORE_LOG を使用して、RA3 ノードへの従来のサイズ変更中のクラスター内の各テーブルについて、移行の進行状況をモニタリングします。サイズ変更操作中のデータ移行の履歴スループットをキャプチャします。RA3 ノードへの従来のサイズ変更について詳しくは、「従来のサイズ変更」を参照してください。

SYS_RESTORE_LOG は、スーパーユーザーにのみ表示されます。

テーブルの列

列名	データ型	説明
event_time	timestamp	ログエントリがいつ記録されるかを示すタイムスタンプ。
database_name	char(128)	データベースの名前。
schema_name	char(128)	スキーマの名前。
table_name	char(128)	テーブルの名前。
table_id	integer	テーブルの ID。
アクション	char(128)	エントリ時に実行されたアクション。値には、「移行開始」、「チェックポイント」、「再開」、「完了」、「キャンセル」、「リセット」などがあります。
table_size	long	テーブルのサイズ。
total_data_processed	long	この時点でテーブル用に処理されたデータのサイズ (MB 単位)。

列名	データ型	説明
delta_data_processed	long	event_time の前回の更新以降に処理されたデータのサイズ (MB 単位) 。これは、前回記録された時間間隔以降に処理されたデータの量を判断するのに役立ちます。これを table_size と比較すると、データ処理がどれだけ速く進んでいるかがわかります。
message	char(512)	アクション列の値の詳細な説明。
redistribution_type	char(32)	テーブルの再分散タイプ。KEY 変換または EVEN リバランスタスクのいずれか。分散スタイルの詳細については、「分散スタイル」を参照してください。

サンプルクエリ

次のクエリは、SYS_RESTORE_LOG を使用してデータ処理のスループットを計算します。

```
SELECT
    ROUND(sum(delta_data_processed) / 1024.0, 2) as data_processed_gb,
    ROUND(datediff(sec, min(event_time), max(event_time)) / 3600.0, 2) as duration_hr,
    ROUND(data_processed_gb/duration_hr, 2) as throughput_gb_per_hr
from sys_restore_log;
```

サンプル出力。

```
 data_processed_gb | duration_hr | throughput_gb_per_hr
-------------------+-------------+----------------------
              0.91 |        8.37 |                 0.11
(1 row)
```

次のクエリは、すべての再分散タイプを表示します。

```
SELECT * from sys_restore_log ORDER BY event_time;
```

```
 database_name |     schema_name      |      table_name      | table_id |
 action             | total_data_processed | delta_data_processed |         event_time
       | table_size | message |   redistribution_type
---------------+----------------------+----------------------+----------
+-----------------------------+----------------------+----------------------
+----------------------------+------------+---------+-------------------------
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:00.744977 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:18:02.756675 |         90 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t2_autokey   |   106430 |
 Redistribution completed    |                   90 |                   90 | 2024-01-05
 02:23:30.643718 |         90 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | customer_key         |   106424 |
 Redistribution completed    |                  325 |                  325 | 2024-01-05
 02:23:45.998249 |        325 |         | Restore Distkey Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.083849 |         30 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution started      |                    0 |                      | 2024-01-05
 02:23:46.855728 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t5_auto_even |   106436 |
 Redistribution completed    |                   45 |                   45 | 2024-01-05
 02:24:16.343029 |         45 |         | Rebalance Disteven Table
 dev           | schemaaaa877096d844d | dp30907_t1_even      |   106428 |
 Redistribution completed    |                   30 |                   30 | 2024-01-05
 02:24:20.584703 |         30 |         | Rebalance Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution started      |                    0 |                      | 2024-01-05
 04:54:55.641741 |        190 |         | Restore Disteven Table
 dev           | schemaefd028a2a48a4c | customer_even        |   130512 |
 Redistribution checkpointed |     29.4342113157737 |     29.4342113157737 | 2024-01-05
 04:55:04.770696 |        190 |         | Restore Disteven Table
(8 rows)
```

SYS_RESTORE_STATE

SYS_RESTORE_STATE を使用して、従来のサイズ変更中の各テーブルの移行の進行状況をモニタリングします。これは、ターゲットノードタイプが RA3 の場合、特に当てはまります。RA3 ノードへの従来のサイズ変更について詳しくは、「従来のサイズ変更」を参照してください。

SYS_RESTORE_STATE は、スーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを送信したユーザーの ID。
database_name	char(64)	テーブルのデータベースの名前。
schema_id	integer	テーブルのスキーマ ID。
table_id	integer	テーブル ID。
table_name	char(128)	テーブルの名前。
redistribution_status	char(128)	テーブルの再分散の進捗状況。指定できる値は `Completed` 、`In progress`、および `Pending` です。
percentage_redistributed	フロート	テーブルの再分散の進行状況のパーセンテージ。有効な値は 0～100% です。例えば、値が 25 の場合、データの 25% が再分散されることを示しています。
redistribution_type	char(32)	テーブルの再分散タイプ。KEY 変換または EVEN リ

列名	データ型	説明
		バランスタスクのいずれか。分散スタイルの詳細については、「分散スタイル」を参照してください。

サンプルクエリ

次のクエリは、実行中とキューに登録されたクエリのレコードを返します。

```
SELECT * FROM sys_restore_state;
```

サンプル出力。

```
 userid | database_name | schema_id | table_id |   table_name   | redistribution_status
 | precentage_redistributed |   redistribution_type
--------+---------------+-----------+----------+----------------
+------------------------+--------------------------+-------------------------
    1   |     test1     |   124865   |  124878   | customer_key_4 |         Pending
 |      0                  |  Rebalance Disteven Table
    1   |      dev      |   124865   |  124874   | customer_key_3 |         Pending
 |      0                  |  Rebalance Disteven Table
    1   |      dev      |   124865   |  124870   | customer_key_2 |        Completed
 |     100                 |  Rebalance Disteven Table
    1   |      dev      |   124865   |  124866   | customer_key_1 |       In progress
 |     13.52               |  Restore Distkey Table
```

以下にデータ処理のステータスを示します。

```
SELECT
    redistribution_status, ROUND(SUM(block_count) / 1024.0, 2) AS total_size_gb
FROM sys_restore_state sys inner join stv_tbl_perm stv
    on sys.table_id = stv.id
GROUP BY sys.redistribution_status;
```

サンプル出力。

```
 redistribution_status | total_size_gb
-----------------------+---------------
```

```
 Completed            |          0.07
 Pending              |          0.71
 In progress          |          0.20
(3 rows)
```

SYS_SCHEMA_QUOTA_VIOLATIONS

スキーマのクォータを超過したときのオカレンス、トランザクション ID、その他の有用な情報を記録します。このシステムテーブルは STL_SCHEMA_QUOTA_VIOLATIONS から複写したものです。

r_SYS_SCHEMA_QUOTA_VIOLATIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
owner_id	integer	スキーマの所有者の ID。
user_id	integer	エントリを生成したユーザーの ID。
transaction_id	bigint	ステートメントに関連付けられるトランザクション ID。
session_id	integer	ステートメントに関連付けられるプロセス ID。
schema_id	integer	名前空間またはスキーマの ID。
schema_name	character (128)	名前空間またはスキーマの名前。
クォータ	integer	スキーマが使用できるディスク容量 (MB)。
disk_usage	integer	スキーマによって現在使用されているディスク容量 (MB)。
record_time	タイムゾーンなしのタイムスタンプ	違反が発生した時刻。

サンプルクエリ

次のクエリは、クォータ違反の結果を示します。

```
SELECT user_id, TRIM(schema_name) "schema_name", quota, disk_usage, record_time FROM
sys_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

このクエリは、指定したスキーマについて次のような出力を返します。

```

user_id| schema_name  | quota | disk_usage | record_time
-------+--------------+-------+------------+----------------------------
104    | sales_schema | 2048  | 2798       | 2020-04-20 20:09:25.494723
(1 row)
```

SYS_SERVERLESS_USAGE

リソースの Amazon Redshift サーバーレスでの使用状況を詳細表示するには、SYS_SERVERLESS_USAGE を使用します。このシステムビューは、プロビジョニングされた Amazon Redshift クラスターには適用されません。

このビューは、サーバーレス使用状況の概要を表示します。この情報には、クエリの処理に使用されるコンピューティング容量や、1 分ごとに使用される Amazon Redshift マネージド型ストレージ容量などが含まれています。コンピューティング容量の単位は、Redshift プロセッシング単位 (RPU) が使用され、実行するワークロードについて、RPU 秒として秒単位で測定されます。RPU は、データウェアハウスにロードされたデータ、Amazon S3 データレイクからクエリされたデータ、または横串検索を使用してオペレーショナルデータベースからアクセスされたデータに対するクエリを処理するために使用されます。Amazon Redshift Serverless では、SYS_SERVERLESS_USAGE にある情報を 7 日間保持します。

コンピューティングコスト請求の例については、「Amazon Redshift Serverless での請求」を参照してください。

SYS_SERVERLESS_USAGE はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
start_time	timestamp	インターバルが開始した時刻。
end_time	timestamp	インターバルが完了した時刻。
compute_seconds	double precision	この時間インターバルで消費された累積コンピューティング単位を表す RPU 秒。この値は、アカウントに割り当てられたベース RPU 容量を考慮に入れています。
compute_capacity	double precision	この期間内に割り当てられたコンピューティングユニット (Redshift プロセッシング単位 = RPU) の平均数。 compute_capacity 値は動的な変更が可能です。
data_storage	integer	この時間の間隔で使用されたデータストレージの平均容量 (MB 単位)。 データベースからデータが読み込まれたり削除されたりすると、使用されるデータストレージが動的に変化することがあります。
cross_region_transferred_data	integer	この時間帯にクロスリージョンから転送されたバイト単位の蓄積データ。

列名	データ型	説明
charged_seconds	integer	この時間間隔に請求された累積コンピュートユニット (RPU) 秒数。これはトランザクションの終了後に計算されるため、トランザクションの実行中は 0 になることがあります。charged_seconds を使用して Amazon Redshift Serverless ワークグループのコストを計算します。この値は、Amazon Redshift Serverless ワークグループに割り当てられた RPU 容量を考慮に入れています。

使用に関する注意事項

- compute_seconds が 0 であるが charged_seconds が 0 より大きい場合も、その逆の場合もあります。これはシステムビューでのデータの記録方法によって生じる通常の動作です。サーバーレスの使用状況の詳細をより正確に表すには、データを集計することをお勧めします。

例

charged_seconds をクエリして、ある時間間隔に使用された RPU 時間の合計請求額を取得するには、次のクエリを実行します。

```
select trunc(start_time) "Day",
(sum(charged_seconds)/3600::double precision) * <Price for 1 RPU> as cost_incurred
from sys_serverless_usage
group by 1
order by 1
```

この間隔の中には、アイドル時間が含まれる場合があることにご注意ください。アイドル時間は消費される RPU に加算されません。

SYS_SESSION_HISTORY

SYS_SESSION_HISTORY を使用して、現在のアクティブなセッションとセッション履歴に関する情報を表示します。

SYS_SESSION_HISTORY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	char(50)	エントリを生成したユーザーの ID。
session_id	integer	ステートメントに関連付けられたセッションの ID。
database_name	char(50)	データベースの名前。
status	char	セッションのステータス。指定できる値は `active`、`timed out`、および `closed` です。
session_timeout	integer	タイムアウトするまで、セッションが非アクティブまたはアイドル状態を維持する最大秒数。0 の場合は、タイムアウトが設定されていないことを示します。
start_time	timestamp	接続が確立されたときのタイムスタンプ。
end_time	timestamp	接続が停止したときのタイムスタンプ。

例

次は SYS_SESSION_HISTORY のサンプル出力です。

```
select * from sys_session_history;
 user_id | session_id |   database_name   | status | session_timeout |
 start_time          |            end_time
---------+------------+-------------------+--------+-----------------
+----------------------------+----------------------------
```

```
       1 | 1073971370 | dev              | closed |         0         | 2023-07-17
15:50:12.030104 | 2023-07-17 15:50:12.123218
       1 | 1073979694 | dev              | closed |         0         | 2023-07-17
15:50:24.117947 | 2023-07-17 15:50:24.131859
       1 | 1073873049 | dev              | closed |         0         | 2023-07-17
15:49:29.067398 | 2023-07-17 15:49:29.070294
       1 | 1073873086 | database18127a4a | closed |         0         | 2023-07-17
15:49:29.119018 | 2023-07-17 15:49:29.125925
       1 | 1073832112 | dev              | closed |         0         | 2023-07-17
15:49:29.164688 | 2023-07-17 15:49:29.179631
       1 | 1073987697 | dev              | closed |         0         | 2023-07-17
15:49:29.26749  | 2023-07-17 15:49:29.273034
       1 | 1073922429 | dev              | closed |         0         | 2023-07-17
15:49:33.35315  | 2023-07-17 15:49:33.367499
       1 | 1073766783 | dev              | closed |         0         | 2023-07-17
15:49:45.38237  | 2023-07-17 15:49:45.396902
       1 | 1073807506 | dev              | active |         0         | 2023-07-17
15:51:48        |
```

SYS_SPATIAL_SIMPLIFY

システムビュー SYS_SPATIAL_SIMPLIFY をクエリして、COPY コマンドを通じて簡略化した空間ジオメトリオブジェクトに関する情報を取得できます。シェープファイルで COPY を使用する場合、SIMPLIFY `tolerance`、SIMPLIFY AUTO、および SIMPLIFY AUTO `max_tolerance` の取り込みオプションを指定できます。簡略化の結果は SYS_SPATIAL_SIMPLIFY システムビューに要約されます。

SIMPLIFY AUTO `max_tolerance` が設定されている場合、このビューには、最大サイズを超えた各ジオメトリの行が含まれます。SIMPLIFY `tolerance` が設定されている場合、COPY オペレーション全体の 1 行が保存されます。この行は、COPY クエリ ID と指定された簡略化の許容値を参照します。

シェープファイルのロードの詳細については、「シェープファイルを Amazon Redshift にロードする」を参照してください。

SYS_SPATIAL_SIMPLIFY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
query_id	bigint	この行を生成したクエリの ID (COPY コマンド)。
line_number	bigint	COPY SIMPLIFY AUTO オプションが指定されている場合、この値はシェープファイル内の簡略化されたレコードのレコード番号です。
maximum_tolerance	double precision	COPY コマンドで指定された距離許容値。これは、SIMPLIFY AUTO オプションを使用した最大許容値、または SIMPLIFY オプションを使用した固定許容値のいずれかです。
initial_size	bigint	簡略化前の GEOMETRY データ値のサイズ (バイト単位)。
簡略化済み	char(1)	COPY SIMPLIFY AUTO オプションが指定されている場合、ジオメトリが正常に簡略化されている場合は t、それ以外の場合は f です。指定した最大許容値で簡略化した後でも、そのサイズが最大ジオメトリのサイズよりも大きい場合、ジオメトリが正常に簡略化されないことがあります。
final_size	bigint	COPY SIMPLIFY AUTO オプションが指定されている場合、これは簡略化後のジオメトリのサイズ (バイト単位) です。
final_tolerance	double precision	簡略化のために選択された最終的な許容値。

サンプルクエリ

次のクエリは、COPY で簡略化されたレコードのリストを返します。

```
SELECT * FROM sys_spatial_simplify;
```

```
 query_id | line_number | maximum_tolerance | initial_size | simplified | final_size |
  final_tolerance
----------+-------------+-------------------+--------------+------------+------------
+----------------------
    20    |      1184704 |                -1 |      1513736 | t          |    1008808 |
  1.276386653895e-05
    20    |      1664115 |                -1 |      1233456 | t          |    1023584 |
 6.11707814796635e-06
```

SYS_STREAM_SCAN_ERRORS

ストリーミング取り込みによってロードされたレコードのエラーを記録します。

SYS_STREAM_SCAN_ERRORS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
external_schema_name	character (128)	Kinesis ストリームまたは Amazon MSK トピックのスキーマの名前。大文字と小文字は区別されます。
stream_name	character (255)	ストリームまたはトピックの名前。大文字と小文字は区別されます。
mv_name	character (128)	関連するマテリアライズドビューの名前。何もない場合は空。大文字と小文字は区別されます。
transaction_id	bigint	トランザクション ID。
query_id	bigint	クエリ ID。

列名	データ型	説明
stream_timestamp_type	character (1)	ストリームタイムスタンプのタイプ。大文字と小文字は区別されます。
stream_timestamp	タイムゾーンなしのタイムスタンプ	レコードが到着した時刻。
record_time	タイムゾーンなしのタイムスタンプ	エラーメッセージが記録された時刻。
partition_id	character (128)	パーティション/シャード ID。大文字と小文字は区別されます。
position	character (128)	レコードの位置。これは Kinesis のシーケンス番号または Amazon MSK のオフセットに対応します。大文字と小文字は区別されます。
error_code	integer	エラーコードです。
error_reason	character (128)	エラーの理由。大文字と小文字は区別されます。

SYS_STREAM_SCAN_STATES

ストリーミング取り込みによってロードされたレコードのスキャン状態を記録します。

SYS_STREAM_SCAN_STATES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
external_schema_name	character (128)	外部スキーマ名。大文字と小文字は区別されます。
stream_name	character (255)	ストリーム名。大文字と小文字は区別されます。
mv_name	character (128)	関連するマテリアライズドビューの名前。何もない場合は空。大文字と小文字は区別されます。
transaction_id	bigint	トランザクション ID。
query_id	bigint	クエリ ID。
record_time	タイムゾーンなしのタイムスタンプ	データが記録された時刻。
partition_id	character (128)	パーティションまたはシャード ID。大文字と小文字は区別されます。
latest_position	character (128)	バッチで最後に読み取られたレコードの位置。これは Kinesis のシーケンス番号または Amazon MSK のオフセットに対応します。大文字と小文字は区別されます。

列名	データ型	説明
scanned_rows	bigint	バッチでスキャンされたレコードの数。
skipped_rows	bigint	バッチでスキップされたレコードの数。
scanned_bytes	bigint	バッチでスキャンされたバイト数。
stream_record_time_min	タイムゾーンなしのタイムスタンプ	バッチ内の最も早いレコードの Kinesis または Amazon MSK の到着時刻。
stream_record_time_max	タイムゾーンなしのタイムスタンプ	バッチ内の最新のレコードの Kinesis または Amazon MSK の到着時刻。

次のクエリは、特定のクエリのストリームとトピックデータを表示します。

```
select
 query_id,mv_name::varchar,external_schema_name::varchar,stream_name::varchar,sum(scanned_rows)
 total_records,
sum(scanned_bytes) total_bytes from sys_stream_scan_states where query in
 (5401180,8601939) group by 1,2,3,4;

  query_id  |    mv_name     | external_schema_name |    stream_name    | total_records |
  total_bytes
------------+----------------+----------------------+-----------------+---------------
+----------------
```

```
 5401180    | kinesistest    | kinesis              | kinesisstream    |    1493255696 |
 3209006490704
 8601939    | msktest        | msk                  | mskstream        |      14677023 |
   31056580668
(2 rows)
```

SYS_TRANSACTION_HISTORY

SYS_TRANSACTION_HISTORY を使用して、クエリの追跡時にトランザクションの詳細を確認します。

SYS_TRANSACTION_HISTORY はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	エントリを生成したユーザーの ID。
transaction_id	bigint	トランザクションの ID。
isolation_level	text	トランザクションの分離レベル。指定できる値は `Serializable` および `Snapshot Isolation` です。
status	text	トランザクションのステータス。可能なステータスは `committed` と `rolledback` です。
transaction_start_time	timestamp	トランザクションの開始時刻。
commit_start_time	timestamp	コミットの開始時刻。
commit_end_time	timestamp	コミットの終了時刻。

列名	データ型	説明
blocks_committed	bigint	このコミットの一部として書き込む必要があったブロックの数。
undo_transaction_id	bigint	トランザクションを取り消した場合の、取り消しトランザクションの ID。それ以外の場合、この値は -1 です。

サンプルクエリ

```
select * from sys_transaction_history order by transaction_start_time desc;

 user_id | transaction_id | isolation_level |   status   |   transaction_start_time
   |      commit_start_time       |       commit_end_time       | blocks_committed |
 undo_transaction_id
---------+----------------+-----------------+------------+----------------------------
+----------------------------+----------------------------+------------------
+---------------------
     100 |           1310 | Serializable    | committed  | 2023-08-27 21:03:11.822205 |
 2023-08-28 21:03:11.825287 | 2023-08-28 21:03:11.854883 |               17 |
        -1
     101 |           1345 | Serializable    | committed  | 2023-08-27 21:03:12.000278 |
 2023-08-28 21:03:12.003736 | 2023-08-28 21:03:12.030061 |               17 |
        -1
     102 |           1367 | Serializable    | committed  | 2023-08-27 21:03:12.1532   |
 2023-08-28 21:03:12.156124 | 2023-08-28 21:03:12.186468 |               17 |
        -1
     100 |           1370 | Serializable    | committed  | 2023-08-27 21:03:12.199316 |
 2023-08-28 21:03:12.204854 | 2023-08-28 21:03:12.238186 |               24 |
        -1
     100 |           1408 | Serializable    | committed  | 2023-08-27 21:03:53.891107 |
 2023-08-28 21:03:53.894825 | 2023-08-28 21:03:53.927465 |               17 |
        -1
     100 |           1409 | Serializable    | rolledback | 2023-08-27 21:03:53.936431 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:08.712532 |                0 |
      1409
```

```
      101 |          1415 | Serializable     | committed  | 2023-08-27 21:04:24.283188 |
 2023-08-28 21:04:24.289196 | 2023-08-28 21:04:24.374318 |                 25 |
        -1
      101 |          1416 | Serializable     | committed  | 2023-08-27 21:04:24.38818  |
 2023-08-28 21:04:24.391688 | 2023-08-28 21:04:24.415135 |                 17 |
        -1
      100 |          1417 | Serializable     | rolledback | 2023-08-27 21:04:24.424252 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:28.354826 |                  0 |
      1417
      101 |          1418 | Serializable     | rolledback | 2023-08-27 21:04:24.425195 |
 2000-01-01 00:00:00        | 2023-08-28 21:04:28.680355 |                  0 |
      1418
      100 |          1420 | Serializable     | committed  | 2023-08-27 21:04:28.697607 |
 2023-08-28 21:04:28.702374 | 2023-08-28 21:04:28.735541 |                 23 |
        -1
      101 |          1421 | Serializable     | committed  | 2023-08-27 21:04:28.744854 |
 2023-08-28 21:04:28.749344 | 2023-08-28 21:04:28.779029 |                 23 |
        -1
      101 |          1423 | Serializable     | committed  | 2023-08-27 21:04:28.78942  |
 2023-08-28 21:04:28.791556 | 2023-08-28 21:04:28.817485 |                 16 |
        -1
      100 |          1430 | Serializable     | committed  | 2023-08-27 21:04:28.917788 |
 2023-08-28 21:04:28.919993 | 2023-08-28 21:04:28.944812 |                 16 |
        -1
      102 |          1494 | Serializable     | committed  | 2023-08-27 21:04:37.029058 |
 2023-08-28 21:04:37.033137 | 2023-08-28 21:04:37.062001 |                 16 |
        -1
```

SYS_UDF_LOG

ユーザー定義関数 (UDF) の実行中に生成されたシステム定義エラーおよび警告メッセージを記録します。

SYS_UDF_LOG はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
query_id	bigint	クエリ識別子。

列名	データ型	説明
function_name	text	ユーザー定義関数の名前。
record_time	timestamp	レコードを作成した時刻。
sequence	integer	1 つのログメッセージのシーケンス。
message	text	ログメッセージテキスト。

サンプルクエリ

次の例では UDF がシステム定義されたエラーをどのように処理するかを示します。最初のブロックは、引数の逆を返す UDF 関数の定義を示します。関数を実行して引数として 0 に渡すと、関数はエラーを返します。最後のステートメントは SYS_UDF_LOG に記録されたエラーメッセージを返します。

```
-- Create a function to find the inverse of a number.
CREATE OR REPLACE FUNCTION f_udf_inv(a int)

RETURNS float

IMMUTABLE AS $$return 1/a

$$ LANGUAGE plpythonu;

-- Run the function with 0 to create an error.
Select f_udf_inv(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;


query_id    |    record_time              |                  message
----------+----------------------------
+-------------------------------------------------------
2211        | 2023-08-23 15:53:11.360538 |  ZeroDivisionError: integer division or
 modulo by zero line 2, in f_udf_inv\n return 1/a\n
```

次の例では、UDF にログされたメッセージと警告メッセージを追加することで、ゼロで分割された操作がエラーメッセージで停止する代わりに警告メッセージとなります。

```
-- Create a function to find the inverse of a number and log a warning if you input 0.
CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with 0 to trigger the warning.
Select f_udf_inv_log(0);

-- Query SYS_UDF_LOG to view the message.
Select query_id, record_time, message::varchar from sys_udf_log;

 query_id |        record_time         |                                    message
----------+----------------------------
+------------------------------------------------------------------------------
     0    | 2023-08-23 16:10:48.833503 | WARNING: You attempted to divide by zero.
\nReturning zero instead of error.\n
```

SYS_UNLOAD_DETAIL

SYS_UNLOAD_DETAIL を使用して、UNLOAD 操作の詳細を表示します。UNLOAD ステートメントによって作成された各ファイルについて 1 行ずつ記録します。例えば、UNLOAD で 12 個のファイルが作成される場合、SYS_UNLOAD_DETAIL には対応する 12 行が含まれます。

SYS_UNLOAD_DETAIL はすべてのユーザーが表示できます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	エントリを生成したユーザーの ID。
query_id	integer	UNLOAD コマンドのクエリ識別子。
session_id	integer	クエリステートメントに関連付けられたプロセス ID。
transaction_id	bigint	ステートメントに関連付けられるトランザクション ID。
file_name	character (1280)	Amazon S3 オブジェクトのファイルへの完全パス。
start_time	timestamp	トランザクションが開始された時刻。
end_time	timestamp	トランザクションが完了した時刻。
line_count	bigint	ファイルにアンロードされた行数。
transfer_size	bigint	転送バイト数。
file_format	character (10)	アンロードされたファイルのファイル形式。

サンプルクエリ

次のクエリは、アンロードコマンドの形式、行、ファイル数を含め、アンロードされたクエリの詳細を表示します。

```
select query_id, substring(file_name, 0, 50), transfer_size, file_format from
 sys_unload_detail;
```

サンプル出力。

```

 query_id |                       substring                               | transfer_size
 | file_format
----------+------------------------------------------------------------------
+---------------+-------------
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0000_part_00.gz  |
 395886 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0001_part_00.gz  |
 406444 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0002_part_00.gz  |
 409431 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0003_part_00.gz  |
 403051 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0004_part_00.gz  |
 413592 | Text
     9272 | s3://amzn-s3-demo-bucket/my_unload_doc_venue0005_part_00.gz  |
 395689 | Text
(6 rows)
```

SYS_UNLOAD_HISTORY

SYS_UNLOAD_HISTORY では、UNLOAD コマンドの詳細を表示します。いくつかのフィールドの累積統計情報を含む UNLOAD コマンドが、それぞれの行に対応し表示されます。この情報には、実行中および終了した UNLOAD コマンドの両方が含まれます。

SYS_UNLOAD_HISTORY はすべてのユーザーが表示可能です。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	アンロードを送信したユーザーの識別子。

列名	データ型	説明
query_id	bigint	UNLOAD コマンドのクエリ識別子。
transaction_id	bigint	トランザクション識別子。
session_id	integer	アンロードを実行しているプロセスのプロセス識別子。
database_name	text	対象のオペレーションが発行された時点で、ユーザーが接続されていたデータベースの名前。
status	text	UNLOAD コマンドのステータス。有効な値は、`running`、`completed`、`aborted`、および unknown です。
start_time	timestamp	アンロードが開始した時刻。
end_time	timestamp	アンロードが完了した時刻。
duration	bigint	UNLOAD コマンド中に消費された時間 (マイクロ秒)。
file_format	text	出力ファイルのファイル形式。
compression_type	text	圧縮タイプ
unloaded_location	text	アンロードされたファイル用の Amazon S3 の場所。
unloaded_rows	bigint	行の数。
unloaded_files_count	bigint	出力ファイルとして数えられるファイルの数。

列名	データ型	説明
unloaded_files_size	bigint	出力ファイルのファイルサイズ。
error_message	text	UNLOAD コマンドのエラーメッセージ。

サンプルクエリ

次のクエリは、アンロードコマンドの形式、行、ファイル数を含め、アンロードされたクエリの詳細を表示します。

```
SELECT query_id,
       file_format,
       start_time,
       duration,
       unloaded_rows,
       unloaded_files_count
FROM sys_unload_history
ORDER BY query_id,
file_format limit 100;
```

サンプル出力。

```
 query_id | file_format |         start_time         | duration | unloaded_rows |
 unloaded_files_count
----------+-------------+----------------------------+----------+---------------
+----------------------
   527067 | Text        | 2022-02-09 05:18:35.844452 |  5932478 |            10 |
           1
```

SYS_USERLOG

データベースユーザーに対する次の変更の詳細のレコード。

- ユーザーの作成
- ユーザーの削除
- ユーザーの変更 (名前の変更)

- ユーザーの変更 (プロパティの変更)

このビューをクエリして、サーバーレスワークグループとプロビジョニングされたクラスターに関する情報を確認できます。

SYS_USERLOG はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	アンロードを送信したユーザーの識別子。
user_name	character(50)	変更の影響を受けたユーザーのユーザー名。
original_user_name	character(50)	名前の変更アクションでの変更前のユーザー名。他のすべてのアクションの場合、このフィールドは空です。
アクション	character(10)	実行されたアクション。有効な値は、変更、作成、削除、および名前変更です。
has_create_db_privs	integer	true (値 1) の場合、ユーザーにはデータベースを作成するアクセス許可があります。
is_superuser	integer	true (値 1) の場合、ユーザーはシステムカタログを更新できます。
has_update_catalog_privs	integer	true (値 1) の場合、ユーザーはシステムカタログを更新できます。

列名	データ型	説明
password_expiration	timestamp	パスワードの有効期限日。
session_id	integer	プロセス ID。
transaction_id	bigint	トランザクション ID。
record_time	timestamp	クエリを開始した UTC 時刻。

サンプルクエリ

次の例は、4 つのユーザーアクションを実行してから、SYS_USERLOG ビューをクエリします。

```
CREATE USER userlog1 password 'Userlog1';
ALTER USER userlog1 createdb createuser;
ALTER USER userlog1 rename  to userlog2;
DROP user userlog2;

SELECT user_id, user_name, original_user_name, action, has_create_db_privs,
 is_superuser from SYS_USERLOG order by record_time desc;
```

```
user_id |  user_name | original_user_name |  action | has_create_db_privs |
 is_superuser
--------+------------+--------------------+---------+---------------------+------------
    108 |  userlog2  |                    | drop    |                   1 |   1
    108 |  userlog2  |     userlog1       | rename  |                   1 |   1
    108 |  userlog1  |                    | alter   |                   1 |   1
    108 |  userlog1  |                    | create  |                   0 |   0
 (4 rows)
```

SYS_VACUUM_HISTORY

SYS_VACUUM_HISTORY は、バキュームクエリの詳細を表示するために使用します。VACUUM コマンドの詳細については、「VACUUM」を参照してください。

SYS_VACUUM_HISTORY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
user_id	integer	クエリを開始したユーザーの ID。
transaction_id	long	VACUUM ステートメントのトランザクション ID。
query_id	long	VACUUM ステートメントのクエリ識別子。このテーブルを SYS_QUERY_DETAIL ビューに結合することにより、特定の VACUUM トランザクションで実行された個々の SQL ステートメントを確認できます。データベース全体をバキュームする場合、各テーブルが別々のトランザクションでバキュームされます。自動化された VACUUM 操作の場合、この値は NULL です。
database_name	text	データベースの名前。
schema_name	text	スキーマの名前。
table_name	text	テーブルの名前。
table_id	integer	テーブルの ID。
vacuum_type	文字	VACUUM 操作のタイプ。可能な値は以下のとおりです。

列名	データ型	説明
		• **Delete** • **Sort** • **Reindex** • **Recluster** • **Full** バキュームタイプの詳細については、「VACUUM」を参照してください。
is_automatic	ブール値	操作が自動バキュームの場合は `true`。そうでない場合は、`false` です。
status	文字	バキュームオペレーションの一環で実行された現在のアクティビティの説明。 • Initialize • Sort • Merge • 削除 • 選択 • 失敗 • 完了 • Skipped • INTERLEAVED SORTKEY の順序の構築
start_time	timestamp	バキューム操作が開始した時刻。

列名	データ型	説明
end_time	timestamp	バキューム操作が終了した時刻。操作が進行中の場合、このフィールドは空白になります。
record_time	timestamp	バキューム操作が SYS_VACUUM_HISTORY に記録された時刻。
duration	integer	バキューム操作の開始から終了までのマイクロ秒数。バキューム操作が進行中の場合、このフィールドは空白になります。
rows_before_vacuum	bigint	テーブル内にある実際の行と、削除されてまだディスクに保存されている (バキューム待ちの) 行の数の合計。
size_before_vacuum	integer	バキューム操作が開始する前のテーブルのサイズ (MB 単位)。
reclaimable_rows	bigint	バキューム操作の開始前に推定した行の再利用数。
reclaimed_rows	bigint	バキューム操作で再利用した行の数。
reclaimed_blocks	bigint	バキューム操作で再利用したブロックの数。
sortedrows_before_vacuum	integer	バキューム操作の開始前にテーブル内でソートされた行の数。

列名	データ型	説明
sortedrows_after_vacuum	integer	バキューム操作の終了後にテーブル内で追加でソートされた行の数。これには、`sortedrows_before_vacuum` にカウントされた行は含まれません。

SYS モニタリングビューに移行するためのシステムビューマッピング

Amazon Redshift のプロビジョニングされたクラスターを Amazon Redshift Serverless に移行すると、モニタリングまたは診断クエリは、プロビジョニングされたクラスターでのみ利用可能なシステムビューを参照する場合があります。SYS モニタリングビューを使用するようにクエリを更新できます。このページでは、クエリの更新時に参照できる、プロビジョニング専用ビューから SYS ビューへのマッピングを提供します。

トピック

- SYS_QUERY_HISTORY
- SYS_QUERY_DETAIL
- SYS_RESTORE_LOG
- SYS_RESTORE_STATE
- SYS_TRANSACTION_HISTORY
- SYS_QUERY_TEXT
- SYS_CONNECTION_LOG
- SYS_SESSION_HISTORY
- SYS_LOAD_DETAIL
- SYS_LOAD_HISTORY
- SYS_LOAD_ERROR_DETAIL
- SYS_UNLOAD_HISTORY
- SYS_UNLOAD_DETAIL
- SYS_COPY_REPLACEMENTS

- SYS_DATASHARE_USAGE_CONSUMER
- SYS_DATASHARE_USAGE_PRODUCER
- SYS_DATASHARE_CROSS_REGION_USAGE
- SYS_DATASHARE_CHANGE_LOG
- SYS_EXTERNAL_QUERY_DETAIL
- SYS_EXTERNAL_QUERY_ERROR
- SYS_VACUUM_HISTORY
- SYS_ANALYZE_HISTORY
- SYS_ANALYZE_COMPRESSION_HISTORY
- SYS_MV_REFRESH_HISTORY
- SYS_MV_STATE
- SYS_PROCEDURE_CALL
- SYS_PROCEDURE_MESSAGES
- SYS_UDF_LOG
- SYS_USERLOG
- SYS_SCHEMA_QUOTA_VIOLATIONS
- SYS_SPATIAL_SIMPLIFY

SYS_QUERY_HISTORY

次の表の一部またはすべての列は、SYS_QUERY_HISTORY にも定義されています。

- STL_DDLTEXT
- STL_ERROR
- STL_QUERY
- STL_UTILITYTEXT
- STL_WLM_QUERY
- STV_INFLIGHT
- STV_RECENTS
- STV_WLM_QUERY_STATE
- SVL_COMPILE
- SVL_MULTI_STATEMENT_VIOLATIONS

- SVL_QLOG
- SVL_QUERY_QUEUE_INFO
- SVL_STATEMENTTEXT
- SVL_TERMINATE

SYS_QUERY_DETAIL

次の表の一部またはすべての列は、SYS_QUERY_DETAIL にも定義されています。

- STL_AGGR
- STL_ALERT_EVENT_LOG
- STL_BCAST
- STL_DELETE
- STL_DIST
- STL_EXPLAIN
- STL_HASH
- STL_HASHJOIN
- STL_INSERT
- STL_LIMIT
- STL_MERGE
- STL_MERGEJOIN
- STL_NESTLOOP
- STL_PARSE
- STL_PLAN_INFO
- STL_PROJECT
- STL_QUERY_METRICS
- STL_RETURN
- STL_SAVE
- STL_SCAN
- STL_SORT
- STL_STREAM_SEGS
- STL_UNIQUE

- STL_WINDOW
- STV_EXEC_STATE
- STV_QUERY_METRICS
- SVCS_QUERY_SUMMARY
- SVL_QUERY_METRICS
- SVL_QUERY_METRICS_SUMMARY
- SVL_QUERY_REPORT
- SVL_QUERY_SUMMARY
- SVV_QUERY_STATE

SYS_RESTORE_LOG

次の表の一部またはすべての列は、SYS_RESTORE_LOG にも定義されています。

- SVL_RESTORE_ALTER_TABLE_PROGRESS

SYS_RESTORE_STATE

次の表の一部またはすべての列は、SYS_RESTORE_STATE にも定義されています。

- STV_XRESTORE_ALTER_QUEUE_STATE

SYS_TRANSACTION_HISTORY

次の表の一部またはすべての列は、SYS_TRANSACTION_HISTORY にも定義されています。

- STL_COMMIT_STATS
- STL_TR_CONFLICT
- STL_UNDONE

SYS_QUERY_TEXT

次の表の一部またはすべての列は、SYS_QUERY_TEXT にも定義されています。

- STL_QUERYTEXT

SYS_CONNECTION_LOG

次の表の一部またはすべての列は、SYS_CONNECTION_LOG にも定義されています。

- STL_CONNECTION_LOG

SYS_SESSION_HISTORY

次の表の一部またはすべての列は、SYS_SESSION_HISTORY にも定義されています。

- STL_SESSIONS
- STL_RESTARTED_SESSIONS
- STV_SESSIONS

SYS_LOAD_DETAIL

次の表の一部またはすべての列は、SYS_LOAD_DETAIL にも定義されています。

- STL_LOAD_COMMITS

SYS_LOAD_HISTORY

次の表の一部またはすべての列は、SYS_LOAD_HISTORY にも定義されています。

- STL_LOAD_COMMITS

SYS_LOAD_ERROR_DETAIL

次の表の一部またはすべての列は、SYS_LOAD_ERROR_DETAIL にも定義されています。

- STL_LOADERROR_DETAIL
- STL_LOAD_ERRORS

SYS_UNLOAD_HISTORY

次の表の一部またはすべての列は、SYS_UNLOAD_HISTORY にも定義されています。

- STL_UNLOAD_LOG

SYS_UNLOAD_DETAIL

次の表の一部またはすべての列は、SYS_UNLOAD_DETAIL にも定義されています。

- STL_UNLOAD_LOG

SYS_COPY_REPLACEMENTS

次の表の一部またはすべての列は、SYS_COPY_REPLACEMENTS にも定義されています。

- STL_REPLACEMENTS

SYS_DATASHARE_USAGE_CONSUMER

次の表の一部またはすべての列は、SYS_DATASHARE_USAGE_CONSUMER にも定義されています。

- SVL_DATASHARE_USAGE_CONSUMER

SYS_DATASHARE_USAGE_PRODUCER

次の表の一部またはすべての列は、SYS_DATASHARE_USAGE_PRODUCER にも定義されています。

- SVL_DATASHARE_USAGE_PRODUCER

SYS_DATASHARE_CROSS_REGION_USAGE

次の表の一部またはすべての列は、SYS_DATASHARE_CROSS_REGION_USAGE にも定義されています。

- SVL_DATASHARE_CROSS_REGION_USAGE

SYS_DATASHARE_CHANGE_LOG

次の表の一部またはすべての列は、SYS_DATASHARE_CHANGE_LOG にも定義されています。

- SVL_DATASHARE_CHANGE_LOG

SYS_EXTERNAL_QUERY_DETAIL

次の表の一部またはすべての列は、SYS_EXTERNAL_QUERY_DETAIL にも定義されています。

- SVL_FEDERATED_QUERY
- SVL_S3LIST
- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY

SYS_EXTERNAL_QUERY_ERROR

次の表の一部またはすべての列は、SYS_EXTERNAL_QUERY_ERROR にも定義されています。

- SVL_SPECTRUM_SCAN_ERROR

SYS_VACUUM_HISTORY

次の表の一部またはすべての列は、SYS_VACUUM_HISTORY にも定義されています。

- STL_VACUUM
- SVL_VACUUM_PERCENTAGE
- SVV_VACUUM_PROGRESS
- SVV_VACUUM_SUMMARY

SYS_ANALYZE_HISTORY

次の表の一部またはすべての列は、SYS_ANALYZE_HISTORY にも定義されています。

- STL_ANALYZE

SYS_ANALYZE_COMPRESSION_HISTORY

次の表の一部またはすべての列は、SYS_ANALYZE_COMPRESSION_HISTORY にも定義されています。

- STL_ANALYZE_COMPRESSION

SYS_MV_REFRESH_HISTORY

次の表の一部またはすべての列は、SYS_MV_REFRESH_HISTORY にも定義されています。

- SVL_MV_REFRESH_STATUS

SYS_MV_STATE

次の表の一部またはすべての列は、SYS_MV_STATE にも定義されています。

- STL_MV_STATE

SYS_PROCEDURE_CALL

次の表の一部またはすべての列は、SYS_PROCEDURE_CALL にも定義されています。

- SVL_STORED_PROC_CALL

SYS_PROCEDURE_MESSAGES

次の表の一部またはすべての列は、SYS_PROCEDURE_MESSAGES にも定義されています。

- SVL_STORED_PROC_MESSAGES

SYS_UDF_LOG

次の表の一部またはすべての列は、SYS_UDF_LOG にも定義されています。

- SVL_UDF_LOG

SYS_USERLOG

次の表の一部またはすべての列は、SYS_USERLOG にも定義されています。

- STL_USERLOG

SYS_SCHEMA_QUOTA_VIOLATIONS

次の表の一部またはすべての列は、SYS_SCHEMA_QUOTA_VIOLATIONS にも定義されています。

- STL_SCHEMA_QUOTA_VIOLATIONS

SYS_SPATIAL_SIMPLIFY

次の表の一部またはすべての列は、SYS_SPATIAL_SIMPLIFY にも定義されています。

- SVL_SPATIAL_SIMPLIFY

システムモニタリング (プロビジョニングのみ)

次のシステムテーブルとビューは、プロビジョニングされたクラスターでクエリできます。STL および STV のテーブルおよびビューには、複数のシステムテーブルから集めたデータのサブセットが含まれます。これらを使用すると、それらのテーブルで検索された一般的なクエリデータにすばやく簡単にアクセスできます。

SVCS ビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。SVL ビューは、SVL_STATEMENTTEXT を除いて、メインクラスターで実行されたクエリについてのみ情報を表示します。SVL_STATEMENTTEXT には、同時実行スケーリングクラスターおよびメインクラスターで実行されるクエリの情報を含めることができます。

トピック

- ログ記録のための STL ビュー
- スナップショットデータの STV テーブル
- メインおよび同時実行スケーリングクラスターの SVCS ビュー
- メインクラスターの SVL ビュー

ログ記録のための STL ビュー

STL システムビューは、システムの履歴を提供するために Amazon Redshift ログファイルから生成されます。

これらのファイルは、データウェアハウスクラスター内の各ノードに置かれます。STL ビューは、ログから取得した情報を、システム管理者が使用できる形式のビューにしたものです。

ログの保持 — STL システムビューは 7 日間のログ履歴を保持します。ログの保持は、すべてのクラスターサイズとノードタイプで保証されており、クラスターワークロードの変化による影響を受けません。また、ログの保持は、クラスターの一時停止などのクラスターの状態からも影響を受けません。クラスターが新しい場合のみ、ログ履歴が 7 日未満になります。ログを保持するために必要なアクションはありませんが、7 日以上前のログデータを保持するには、ログを定期的に他のテーブルにコピーするか、Amazon S3 にアンロードする必要があります。

トピック

- STL_USERLOG
- STL_UTILITYTEXT
- STL_VACUUM
- STL_WINDOW
- STL_WLM_ERROR
- STL_WLM_RULE_ACTION
- STL_WLM_QUERY

STL_AGGR

集計を実行するステップのクエリについて分析します。集計を実行するステップは、集計関数および GROUP BY 句の実行時に発生します。

STL_AGGR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_AGGR には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。

列名	データ型	説明
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
slots	integer	ハッシュバケットの数。
occupied	integer	レコードが含まれているスロットの数。
maxlength	integer	最も大きなスロットのサイズ。
tbl	integer	テーブル ID。
is_diskbased	character(1)	true (t) の場合、クエリはディスクベースのオペレーションとして実行されました。false (f) の場合、クエリはメモリ内で実行されました。
workmem	bigint	ステップに割り当てられた作業メモリのバイト数。

列名	データ型	説明
type	character(6)	ステップの種類。有効な値は次のとおりです。 • HASHED。ステップは、グループ化され、ソートされていない集計を使用しました。 • PLAIN。ステップは、グループ化されていない、スカラーの集計を使用しました。 • SORTED。ステップは、グループ化され、ソートされた集計を使用しました。
サイズ変更	integer	この情報は、内部使用に限定されています。
フラッシュ可能	integer	この情報は、内部使用に限定されています。

サンプルクエリ

SLICE 1、TBL 239 について、集計を実行するステップに関する情報を返します。

```
select query, segment, bytes, slots, occupied, maxlength, is_diskbased, workmem, type
from stl_aggr where slice=1 and tbl=239
order by rows
limit 10;
```

```
 query | segment | bytes |  slots   | occupied | maxlength | is_diskbased |  workmem  |
 type
-------+---------+-------+---------+----------+-----------+--------------+-----------
+--------
   562 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   616 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   546 |       1 |     0 | 4194304 |        0 |         0 | f            | 383385600 |
 HASHED
   547 |       0 |     8 |       0 |        0 |         0 | f            |         0 |
 PLAIN
   685 |       1 |    32 | 4194304 |        1 |         0 | f            | 383385600 |
 HASHED
```

```
    652 |         0 |      8 |         0 |          0 |          0 | f               |         0 |
  PLAIN
    680 |         0 |      8 |         0 |          0 |          0 | f               |         0 |
  PLAIN
    658 |         0 |      8 |         0 |          0 |          0 | f               |         0 |
  PLAIN
    686 |         0 |      8 |         0 |          0 |          0 | f               |         0 |
  PLAIN
    695 |         1 |     32 | 4194304 |          1 |          0 | f               | 383385600 |
  HASHED
(10 rows)
```

STL_ALERT_EVENT_LOG

パフォーマンスの問題を示している可能性のある条件がクエリオプティマイザによって特定された場合にアラートを記録します。クエリパフォーマンスを向上させる機会を特定するには、STL_ALERT_EVENT_LOG ビューを使用します。

複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。詳細については、「クエリ処理」を参照してください。

STL_ALERT_EVENT_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_ALERT_EVENT_LOG には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。

列名	データ型	説明
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
pid	integer	ステートメントとスライスに関連付けられるプロセス ID。複数のスライスで実行される場合、同じクエリに複数の PID がある可能性があります。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
event	character (1024)	アラートイベントの説明。
solution	character (1024)	推奨される解決策。
event_time	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。

使用に関する注意事項

STL_ALERT_EVENT_LOG を使用してクエリの潜在的な問題を特定し、「クエリパフォーマンスのチューニング」の説明に従ってデータベース設計を最適化して、クエリを再作成できます。STL_ALERT_EVENT_LOG は以下のアラートを記録します。

- 見つからない統計

  統計が見つかりません。データロードまたは大規模な更新の後で ANALYZE を実行し、COPY 操作で STATUPDATE を使用します。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。
- ネステッドループ

通常、ネステッドループは直積集合です。クエリを評価して、関与しているすべてのテーブルが効率的に結合されていることを確認します。

- 非常に選択的なフィルター

  スキャンされた行に対する返された行の比率が 0.05 未満です。スキャンされる行数は rows_pre_user_filter の値であり、返される行数は STL_SCAN システムビューの行の値です。結果セットを決定するために、クエリが著しく大量の行数をスキャンしていることを示します。この問題は、ソートキーが見つからない場合や正しくない場合に起こります。詳細については、「ソートキー」を参照してください。

- 過剰な数の非実体行

  削除済みだがバキューム未処理としてマークされた比較的多数の行、または挿入済みだがコミットされていない比較的多数の行がスキャンによってスキップされました。詳細については、「テーブルのバキューム処理」を参照してください。

- サイズの大きな分散

  ハッシュ結合または集計で 100 万を超える行が再分散されました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

- サイズの大きなブロードキャスト

  ハッシュ結合で 100 万を超える行がブロードキャストされました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

- 直列実行

  内部テーブル全体が単一ノードに再分散されたために直列実行を強制する、DS_DIST_ALL_INNER 再分散スタイルがクエリプランで指定されました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

サンプルクエリ

次のクエリは、4 つのクエリに関するアラートイベントを表示します。

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from stl_alert_event_log order by query;

 query |             event             |          solution            |     event_time
```

```
-------+-------------------------------+------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command      | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

STL_ANALYZE

ANALYZE オペレーションの詳細を記録します。

STL_ANALYZE はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_ANALYZE_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	long	トランザクション ID。
データベース	char(30)	データベース名。
table_id	integer	テーブル ID。
status	char(15)	ANALYZE コマンドの結果。指定できる値は `Full`、`Skipped`、および `PredicateColumn` です。
rows	double	テーブル内の合計行数。

列名	データ型	説明
modified_rows	double	最後の ANALYZE オペレーション以降に変更された合計行数。
threshold_percent	integer	`analyze_threshold_percent` パラメータの値。
is_auto	char(1)	オペレーションにデフォルトで Amazon Redshift 分析オペレーションが含まれている場合、値は true (`t`) です。ANALYZE コマンドが明示的に実行された場合、値は false (`f`) です。
starttime	timestamp	分析オペレーションの実行を開始した時刻 (UTC 時間)。
endtime	timestamp	分析オペレーションの実行を終了した時刻 (UTC 時間)。
prevtime	timestamp	テーブルが前回分析された時間 (UTC)。
num_predicate_cols	integer	現在テーブルにある述語の列の数。
num_new_predicate_cols	integer	前回の分析オペレーションから加わった、テーブルにある新しい述語の列の数。
is_background	character(1)	分析が自動分析操作によって実行された場合、値は true (`t`) です。それ以外の場合、値は false (`f`) に設定されます。
auto_analyze_phase	character(100)	内部で使用するために予約しています。
schema_name	char(128)	テーブルのスキーマ名。
table_name	char(136)	テーブルの名前。

サンプルクエリ

次の例では、STV_TBL_PERM を結合して、テーブル名と実行の詳細を表示します。

```
select distinct a.xid, trim(t.name) as name, a.status, a.rows, a.modified_rows,
 a.starttime, a.endtime
from stl_analyze a
join stv_tbl_perm t  on t.id=a.table_id
where name = 'users'
order by starttime;

xid    | name  | status          | rows  | modified_rows | starttime           |
 endtime
-------+-------+-----------------+-------+---------------+---------------------
+--------------------
  1582 | users | Full            | 49990 |         49990 | 2016-09-22 22:02:23 |
 2016-09-22 22:02:28
244287 | users | Full            | 24992 |         74988 | 2016-10-04 22:50:58 |
 2016-10-04 22:51:01
244712 | users | Full            | 49984 |         24992 | 2016-10-04 22:56:07 |
 2016-10-04 22:56:07
245071 | users | Skipped         | 49984 |             0 | 2016-10-04 22:58:17 |
 2016-10-04 22:58:17
245439 | users | Skipped         | 49984 |          1982 | 2016-10-04 23:00:13 |
 2016-10-04 23:00:13
(5 rows)
```

STL_ANALYZE_COMPRESSION

COPY または ANALYZE COMPRESSION コマンドの実行中に圧縮分析オペレーションの詳細を記録します。

STL_ANALYZE_COMPRESSION はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_ANALYZE_COMPRESSION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
start_time	timestamp	圧縮分析オペレーションを開始した時刻。
xid	bigint	圧縮分析オペレーションのトランザクション ID。
tbl	integer	分析されたテーブルのテーブル ID。
tablename	character(128)	分析されたテーブルの名前。
col	integer	圧縮エンコードを決定するために分析されたテーブルの列のインデックス。
old_encoding	character(15)	圧縮分析前のエンコードタイプ。
new_encoding	character(15)	圧縮分析後のエンコードタイプ。
mode	character(14)	指定できる値は以下のとおりです。 PRESET `new_encoding` が、列のデータタイプに基づいて Amazon Redshift COPY コマンドによって決定されることを指定します。サンプリングされているデータはありません。 ON `new_encoding` が、サンプルデータの分析に基づいて Amazon Redshift COPY コマンドによって決定されることを指定します。 ANALYZE ONLY `new_encoding` が、サンプルデータの分析に基づいて Amazon Redshift ANALYZE COMPRESSION コマンドに

列名	データ型	説明
		よって決定されることを指定します。ただし、分析された列のエンコードタイプは変更されません。
best_compression_encoding	character(15)	最適な圧縮率を提供するエンコードタイプ。
recommended_bytes	character(15)	新しいエンコードを採用して使用されたバイト数。
best_compression_bytes	character(15)	最適なエンコードを採用して使用されたバイト数。
ndv	bigint	サンプリングされた行内の異なる値の数。

サンプルクエリ

次の例では、同じセッションで実行された最後の COPY コマンドで、`lineitem` テーブルの圧縮分析の詳細を検査します。

```
select xid, tbl, btrim(tablename) as tablename, col, old_encoding, new_encoding,
 best_compression_encoding, mode
from stl_analyze_compression
where xid = (select xid from stl_query where query = pg_last_copy_id()) order by col;

 xid  |  tbl   | tablename | col |  old_encoding   |  new_encoding   |
 best_compression_encoding |      mode
------+--------+-----------+-----+-----------------+-----------------
+---------------------------+----------------
 5308 | 158961 | $lineitem |   0 | mostly32        | az64            | delta
          | ON
 5308 | 158961 | $lineitem |   1 | mostly32        | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   2 | lzo             | az64            | az64
          | ON
 5308 | 158961 | $lineitem |   3 | delta           | az64            | az64
          | ON
```

```
 5308 | 158961 | $lineitem |    4 | bytedict          | az64              | bytedict
          | ON
 5308 | 158961 | $lineitem |    5 | mostly32          | az64              | az64
          | ON
 5308 | 158961 | $lineitem |    6 | delta             | az64              | az64
          | ON
 5308 | 158961 | $lineitem |    7 | delta             | az64              | az64
          | ON
 5308 | 158961 | $lineitem |    8 | lzo               | lzo               | lzo
          | ON
 5308 | 158961 | $lineitem |    9 | runlength         | runlength         | runlength
          | ON
 5308 | 158961 | $lineitem |   10 | delta             | az64              | az64
          | ON
 5308 | 158961 | $lineitem |   11 | delta             | az64              | az64
          | ON
 5308 | 158961 | $lineitem |   12 | delta             | az64              | az64
          | ON
 5308 | 158961 | $lineitem |   13 | bytedict          | bytedict          | bytedict
          | ON
 5308 | 158961 | $lineitem |   14 | bytedict          | bytedict          | bytedict
          | ON
 5308 | 158961 | $lineitem |   15 | text255           | text255           | text255
          | ON
(16 rows)
```

STL_BCAST

データをブロードキャストするクエリステップが実行されている間のネットワークアクティビティに関する情報を記録します。ネットワークトラフィックは、特定のスライス上の特定のステップについて、そのステップの間にネットワークで送信される行、バイト、およびパケットの数によって把握されます。ステップの実行時間は、ログの開始時刻と終了時刻の差です。

クエリ内のブロードキャストステップを識別するには、SVL_QUERY_SUMMARY ビュー内で bcast ラベルを検索するか、EXPLAIN コマンドを実行してから bcast を含む step 属性を検索します。

STL_BCAST はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_BCAST には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
packets	integer	ネットワークを介して送信されたパケットの総数。

サンプルクエリ

次の例では、クエリのブロードキャスト情報が返されます。1 つまたは複数のパケットがあり、クエリの開始と終了の差は 1 秒以上でした。

```
select query, slice, step, rows, bytes, packets, datediff(seconds, starttime, endtime)
from stl_bcast
where packets>0 and datediff(seconds, starttime, endtime)>0;
```

```
 query | slice | step | rows | bytes | packets | date_diff
-------+-------+------+------+-------+---------+-----------
   453 |     2 |    5 |    1 |   264 |       1 |         1
   798 |     2 |    5 |    1 |   264 |       1 |         1
  1408 |     2 |    5 |    1 |   264 |       1 |         1
  2993 |     0 |    5 |    1 |   264 |       1 |         1
  5045 |     3 |    5 |    1 |   264 |       1 |         1
  8073 |     3 |    5 |    1 |   264 |       1 |         1
  8163 |     3 |    5 |    1 |   264 |       1 |         1
  9212 |     1 |    5 |    1 |   264 |       1 |         1
  9873 |     1 |    5 |    1 |   264 |       1 |         1
(9 rows)
```

STL_COMMIT_STATS

コミットのさまざまなステージのタイミングやコミットされるブロックの数など、コミットのパフォーマンスに関連するメトリクスを提供します。トランザクションのどの部分がコミットに費やされ、どのぐらいのキューイングが発生しているかを特定するには、STL_COMMIT_STATS に対してクエリを実行します。

STL_COMMIT_STATS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_TRANSACTION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
xid	bigint	コミットされているトランザクション ID。
node	integer	ノード番号。-1 はリーダーノードです。
startqueue	timestamp	コミット用キューイングの開始。
startwork	timestamp	コミットの開始。
endflush	timestamp	ダーティブロックフラッシュフェーズの終了。
endstage	timestamp	メタデータステージングフェーズの終了。
endlocal	timestamp	ローカルコミットフェーズの終了。
startglobal	timestamp	グローバルフェーズの開始。
endtime	timestamp	コミットの終了。
queuelen	bigint	このトランザクションより先にコミットキューに入れられたトランザクションの数。
permblocks	bigint	このコミットの時点における既存の永続ブロックの数。
newblocks	bigint	このコミットの時点における新しい永続ブロックの数。
dirtyblocks	bigint	このコミットの一部として書き込む必要があったブロックの数。
headers	bigint	このコミットの一部として書き込む必要があったブロックヘッダーの数。
numxids	integer	アクティブな DML トランザクションの数。
oldestxid	bigint	最も古いアクティブな DML のトランザクションの XID。
extwritelatency	bigint	この情報は、内部使用に限定されています。

列名	データ型	説明
metadatawritten	int	この情報は、内部使用に限定されています。
tombstonedblocks	bigint	この情報は、内部使用に限定されています。
tossedblocks	bigint	この情報は、内部使用に限定されています。
batched_by	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

```
select node, datediff(ms,startqueue,startwork) as queue_time,
datediff(ms, startwork, endtime) as commit_time, queuelen
from stl_commit_stats
where xid = 2574
order by node;

node | queue_time   | commit_time | queuelen
-----+--------------+-------------+---------
  -1 |            0 |         617 |        0
   0 | 444950725641 |         616 |        0
   1 | 444950725636 |         616 |        0
```

STL_CONNECTION_LOG

認証の試みと、接続および切断をログに記録します。

STL_CONNECTION_LOG はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_CONNECTION_LOG でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
event	character(50)	接続または認証イベント。
recordtime	timestamp	イベントが発生した時刻。
remotehost	character(45)	リモートホストの名前または IP アドレス。
remoteport	character(32)	リモートホストのポート番号。
pid	integer	ステートメントに関連付けられるプロセス ID。
dbname	character(50)	データベース名。
username	character(50)	ユーザー名。
authmethod	character(32)	認証方法。
duration	integer	接続時間 (マイクロ秒)。
sslversion	character(50)	Secure Sockets Layer (SSL) バージョン。
sslcipher	character(128)	SSL 暗号。
mtu	integer	最大送信単位 (MTU)。
sslcompression	character(64)	SSL 圧縮タイプ。
sslexpansion	character(64)	SSL 拡張タイプ。
iamauthguid	character(36)	CloudTrail リクエストの IAM 認証 ID。
application_name	character(250)	セッションのアプリケーションの初期名または更新名。
os_version	character(64)	Amazon Redshift クラスターに接続するクライアントマシン上にあるオペレーティングシステムのバージョン。

列名	データ型	説明
driver_version	character(64)	サードパーティーの SQL クライアントツールから Amazon Redshift クラスターに接続する ODBC または JDBC ドライバーのバージョン。
plugin_name	character(32)	Amazon Redshift クラスターへの接続に使用されるプラグインの名前。
protocol_version	integer	Amazon Redshift ドライバーが、サーバーとの接続を確立する際に使用する内部プロトコルのバージョン。プロトコルのバージョンは、ドライバとサーバ間でネゴシエートされます。バージョンは、利用可能な機能を説明します。有効な値を次に示します。 • 0 (BASE_SERVER_PROTOCOL_VERSION) • 1 (EXTENDED_RESULT_METADATA_SERVERSION) – クエリごとのラウンドトリップを省くため、サーバーが追加の結果セットのメタデータ情報を送信します。 • 2 (BINARY_PROTOCLO_VERSION) – 結果セットのデータ型に応じて、サーバーがバイナリ形式でデータを送信します。 • 3 (EXTENDED2_RESULT_METADATA_SERVERSION) – サーバーが列の大文字と小文字の区別 (照合順序) 情報を送信します。
sessionid	character(36)	現在のセッションのグローバル一意識別子。セッション ID は、ノード障害による再起動後も存続します。
compression	character(16)	接続に使用されている圧縮アルゴリズム。

サンプルクエリ

オープン接続の詳細を表示するには、以下のクエリを実行します。

```
select recordtime, username, dbname, remotehost, remoteport
from stl_connection_log
```

```
where event = 'initiating session'
and pid not in
(select pid from stl_connection_log
where event = 'disconnecting session')
order by 1 desc;

recordtime          | username    | dbname     | remotehost    | remoteport

--------------------+-------------+------------+---------------+------------
2014-11-06 20:30:06 | rdsdb       | dev        | [local]       |

2014-11-06 20:29:37 | test001     | test       | 10.49.42.138  | 11111

2014-11-05 20:30:29 | rdsdb       | dev        | 10.49.42.138  | 33333

2014-11-05 20:28:35 | rdsdb       | dev        | [local]       |
(4 rows)
```

以下の例は、失敗した認証の試みと、成功した接続および切断を反映しています。

```
select event, recordtime, remotehost, username
from stl_connection_log order by recordtime;

            event      |         recordtime        |  remotehost  | username

-----------------------+---------------------------+--------------+---------
authentication failure | 2012-10-25 14:41:56.96391 | 10.49.42.138 | john

authenticated          | 2012-10-25 14:42:10.87613 | 10.49.42.138 | john

initiating session     | 2012-10-25 14:42:10.87638 | 10.49.42.138 | john

disconnecting session  | 2012-10-25 14:42:19.95992 | 10.49.42.138 | john

(4 rows)
```

以下の例は、ODBC ドライバーのバージョン、クライアントマシンのオペレーティングシステム、および Amazon Redshift クラスターへの接続に使用されるプラグインを示しています。この例では、使用されるプラグインは、ログイン名とパスワードを使用した標準の ODBC ドライバー認証に使用されます。

```
select driver_version, os_version, plugin_name from stl_connection_log;
```

```
driver_version                          |  os_version                       |
 plugin_name
----------------------------------------+----------------------------------
+--------------------
Amazon Redshift ODBC Driver 1.4.15.0001 | Darwin 18.7.0 x86_64              | none
Amazon Redshift ODBC Driver 1.4.15.0001 | Linux 4.15.0-101-generic x86_64   | none
```

次の例では、クライアントマシン上のオペレーティングシステムのバージョン、ドライバのバージョン、およびプロトコルのバージョンを表示します。

```
select os_version, driver_version, protocol_version from stl_connection_log;

os_version                      |  driver_version              | protocol_version
--------------------------------+------------------------------+--------------------
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
Linux 4.15.0-101-generic x86_64 | Redshift JDBC Driver 2.0.0.0 | 2
```

STL_DDLTEXT

システムで実行された以下の DDL ステートメントを取得します。

これらの DDL ステートメントには、以下のクエリとオブジェクトが含まれます。

- CREATE SCHEMA、TABLE、VIEW
- DROP SCHEMA、TABLE、VIEW
- ALTER SCHEMA、TABLE

STL_QUERYTEXT、STL_UTILITYTEXT、SVL_STATEMENTTEXT も参照してください。これらのビューには、システムで実行される SQL コマンドのタイムラインが示されます。この履歴は、トラブルシューティング、およびすべてのシステムアクティビティの監査追跡の作成に役立ちます。

STARTTIME および ENDTIME 列を使用すると、一定の時間内に記録されたステートメントがわかります。SQL テキストの長いブロックは、200 文字の長さに分割されます。SEQUENCE 列により、1 つのステートメントに属する複数のフラグメントのテキストを識別できます。

STL_DDLTEXT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントに関連付けられるプロセス ID。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドは空になります。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
sequence	integer	1 つのステートメントに含まれる文字数が 200 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
text	character(200)	200 文字単位の SQL テキスト。このフィールドには、バックスラッシュ (\\) や改行 (\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

次のクエリは、以前に実行した DDL ステートメントを含むレコードを返します。

```
select xid, starttime, sequence, substring(text,1,40) as text
from stl_ddltext order by xid desc, sequence;
```

4 つの CREATE TABLE ステートメントを表示するサンプル出力を次に示します。DDL ステートメントは読みやすくするために切り詰められ、text列に表示されます。

```
 xid  |         starttime          | sequence |                  text
------+----------------------------+----------
+-----------------------------------------
 1806 | 2013-10-23 00:11:14.709851 |        0 | CREATE TABLE supplier ( s_suppkey int4
 N
 1806 | 2013-10-23 00:11:14.709851 |        1 |  s_comment varchar(101) NOT NULL )
 1805 | 2013-10-23 00:11:14.496153 |        0 | CREATE TABLE region ( r_regionkey int4
 N
 1804 | 2013-10-23 00:11:14.285986 |        0 | CREATE TABLE partsupp ( ps_partkey int8
 1803 | 2013-10-23 00:11:14.056901 |        0 | CREATE TABLE part ( p_partkey int8 NOT
 N
 1803 | 2013-10-23 00:11:14.056901 |        1 | ner char(10) NOT NULL , p_retailprice
 nu
(6 rows)
```

ストアド SQL の再構築

次の SQL は、STL_DDLTEXT の text 列に保存されている行を一覧表示します。行の順序は、xid および sequence によって決まります。元の SQL が 200 文字より長い複数行の場合、STL_DDLTEXT は sequence により複数の行を含むことができます。

```
SELECT xid, sequence, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text)
 END, '') WITHIN GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, sequence ORDER BY xid, sequence;
```

```


xid     |  sequence | query_statement
--------+-----------+-----------------
7886671    0          create external schema schema_spectrum_uddh\nfrom data catalog
\ndatabase 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
7886752    0          CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n
  league_rank smallint,\n  prev_rank   smallint,\n  club_name   varchar(15),\n
 league_name varchar(20),\n  league_off  decimal(6,2),\n  le
```

```
7886752    1          ague_def  decimal(6,2),\n  league_spi  decimal(6,2),\n
 league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
 LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's
7886752    2          3://mybucket-spectrum-uddh/'\ntable properties
 ('skip.header.line.count'='1');
...
```

STL_DDLTEXT の text 列に保存された SQL を再構築するには、次の SQL ステートメントを実行します。これにより、text列の 1 つ以上のセグメントからの DDL ステートメントが 1 つに統合されます。再構築された SQL を実行する前に、特殊文字 (\n) がある場合は、SQL クライアントで改行に置き換えます。次の SELECT ステートメントの結果は、query_statement フィールドの SQL を再構築するために 3 つの行を順にまとめたものです。

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_ddltext GROUP BY xid, endtime order by xid, endtime;
```

```


query_statement
--------------
create external schema schema_spectrum_uddh\nfrom data catalog\ndatabase
 'spectrum_db_uddh'\niam_role ''\ncreate external database if not exists;
CREATE EXTERNAL TABLE schema_spectrum_uddh.soccer_league\n(\n  league_rank smallint,
\n  prev_rank   smallint,\n  club_name   varchar(15),\n  league_name varchar(20),\n
 league_off  decimal(6,2),\n  league_def  decimal(6,2),\n  league_spi  decimal(6,2),
\n  league_nspi smallint\n)\nROW FORMAT DELIMITED \n    FIELDS TERMINATED BY ',' \n
  LINES TERMINATED BY '\\n\\l'\nstored as textfile\nLOCATION 's3://mybucket-spectrum-
uddh/'\ntable properties ('skip.header.line.count'='1');
```

STL_DELETE

削除を実行するステップをクエリについて分析します。

STL_DELETE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_DELETE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同

時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
tbl	integer	テーブル ID。

サンプルクエリ

STL_DELETE に行を作成するために、次の例では EVENT テーブルに行を挿入してから、その行を削除しています。

最初に、EVENT テーブルに行を 1 つ挿入し、挿入されたことを確認します。

```
insert into event(eventid,venueid,catid,dateid,eventname)
values ((select max(eventid)+1 from event),95,9,1857,'Lollapalooza');
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
    7972 |      92 |     9 |   2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |      65 |     9 |   1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |      48 |     9 |   1904 | Lollapalooza | 2008-03-19 15:00:00
    8799 |      95 |     9 |   1857 | Lollapalooza |
(10 rows)
```

次に、EVENT テーブルに追加した行を削除し、削除されたことを確認します。

```
delete from event
where eventname='Lollapalooza' and eventid=(select max(eventid) from event);
```

```
select * from event
where eventname='Lollapalooza'
order by eventid;
```

```
 eventid | venueid | catid | dateid |  eventname   |      starttime
---------+---------+-------+--------+--------------+---------------------
    4274 |     102 |     9 |   1965 | Lollapalooza | 2008-05-01 19:00:00
    4684 |     114 |     9 |   2105 | Lollapalooza | 2008-10-06 14:00:00
    5673 |     128 |     9 |   1973 | Lollapalooza | 2008-05-01 15:00:00
    5740 |      51 |     9 |   1933 | Lollapalooza | 2008-04-17 15:00:00
    5856 |     119 |     9 |   1831 | Lollapalooza | 2008-01-05 14:00:00
    6040 |     126 |     9 |   2145 | Lollapalooza | 2008-11-15 15:00:00
```

```
    7972 |       92 |      9 |    2026 | Lollapalooza | 2008-07-19 19:30:00
    8046 |       65 |      9 |    1840 | Lollapalooza | 2008-01-14 15:00:00
    8518 |       48 |      9 |    1904 | Lollapalooza | 2008-03-19 15:00:00
(9 rows)
```

その後、stl_delete をクエリして、削除を実行するステップを確認します。この例ではクエリによって 300 行以上が返されるため、以下に示す出力は表示の都合上、一部を省略しています。

```
select query, slice, segment, step, tasknum, rows, tbl from stl_delete order by query;
```

```
 query | slice | segment | step | tasknum | rows |  tbl
-------+-------+---------+------+---------+------+--------
     7 |     0 |       0 |    1 |       0 |    0 | 100000
     7 |     1 |       0 |    1 |       0 |    0 | 100000
     8 |     0 |       0 |    1 |       2 |    0 | 100001
     8 |     1 |       0 |    1 |       2 |    0 | 100001
     9 |     0 |       0 |    1 |       4 |    0 | 100002
     9 |     1 |       0 |    1 |       4 |    0 | 100002
    10 |     0 |       0 |    1 |       6 |    0 | 100003
    10 |     1 |       0 |    1 |       6 |    0 | 100003
    11 |     0 |       0 |    1 |       8 |    0 | 100253
    11 |     1 |       0 |    1 |       8 |    0 | 100253
    12 |     0 |       0 |    1 |       0 |    0 | 100255
    12 |     1 |       0 |    1 |       0 |    0 | 100255
    13 |     0 |       0 |    1 |       2 |    0 | 100257
    13 |     1 |       0 |    1 |       2 |    0 | 100257
    14 |     0 |       0 |    1 |       4 |    0 | 100259
    14 |     1 |       0 |    1 |       4 |    0 | 100259
    ...
```

STL_DISK_FULL_DIAG

ディスクがいっぱいになったときに記録されたエラーに関する情報をログに記録します。

STL_DISK_FULL_DIAG は、スーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明		
currenttime	bigint	2000 年 1 月 1 日以降、エラーがマイクロ秒単位で生成された日時。		
node_num	bigint	ノードの識別子。		
query_id	bigint	エラーの原因となったクエリの識別子。		
temp_blocks	bigint	クエリによって作成された一時ブロックの数。		

サンプルクエリ

次の例では、「ディスクがいっぱい」エラーが発生したときに格納されたデータの詳細を返します。

```
select * from stl_disk_full_diag
```

次の例では、currenttime をタイムスタンプに変換します。

```
select '2000-01-01'::timestamp + (currenttime/1000000.0)* interval '1 second' as
 currenttime,node_num,query_id,temp_blocks from pg_catalog.stl_disk_full_diag;
```

```
        currenttime         | node_num | query_id | temp_blocks
----------------------------+----------+----------+-------------
 2019-05-18 19:19:18.609338 |        0 |   569399 |       70982
 2019-05-18 19:37:44.755548 |        0 |   569580 |       70982
 2019-05-20 13:37:20.566916 |        0 |   597424 |       70869
```

STL_DIST

データを配布するクエリステップが実行されている間のネットワークアクティビティに関する情報を記録。ネットワークトラフィックは、特定のスライス上の特定のステップについて、そのステップの

間にネットワークで送信される行、バイト、およびパケットの数によって把握されます。ステップの実行時間は、ログの開始時刻と終了時刻の差です。

クエリ内の配布ステップを識別するには、QUERY_SUMMARY ビュー内で dist ラベルを検索するか、EXPLAIN コマンドを実行してから dist を含む step 属性を検索します。

STL_DIST はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_DIST には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。

列名	データ型	説明
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
packets	integer	ネットワークを介して送信されたパケットの総数。

サンプルクエリ

次の例は、1 つ以上のパケットとゼロより長い期間を持つクエリの分散情報を返します。

```
select query, slice, step, rows, bytes, packets,
datediff(seconds, starttime, endtime) as duration
from stl_dist
where packets>0 and datediff(seconds, starttime, endtime)>0
order by query
limit 10;
```

```
 query  | slice | step |  rows  |  bytes  | packets | duration
--------+-------+------+--------+---------+---------+-----------
    567 |     1 |    4 |  49990 | 6249564 |     707 |          1
    630 |     0 |    5 |   8798 |  408404 |      46 |          2
    645 |     1 |    4 |   8798 |  408404 |      46 |          1
    651 |     1 |    5 | 192497 | 9226320 |    1039 |          6
    669 |     1 |    4 | 192497 | 9226320 |    1039 |          4
    675 |     1 |    5 |   3766 |  194656 |      22 |          1
    696 |     0 |    4 |   3766 |  194656 |      22 |          1
    705 |     0 |    4 |    930 |   44400 |       5 |          1
 111525 |     0 |    3 |     68 |   17408 |       2 |          1
(9 rows)
```

STL_ERROR

Amazon Redshift データベースエンジンによって生成される内部処理エラーを記録します。STL_ERROR では SQL エラーやメッセージは記録されません。STL_ERROR の情報は、特定のエラーのトラブルシューティングに役立ちます。AWS サポートエンジニアが、トラブルシューティングプロセスを進める上で、この情報の提供を要求する場合があります。

STL_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

Copy コマンドを使用したデータのロード中に生成できるエラーコードのリストについては、「ロードエラー参照」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
process	character(12)	例外をスローしたプロセス。
recordtime	timestamp	エラーが発生した時刻。
pid	integer	プロセス ID。STL_QUERY テーブルには、完了したクエリのプロセス ID と、一意のクエリ ID が含まれています。
errcode	integer	エラーカテゴリに対応するエラーコード。
file	character(90)	エラーが発生したソースファイルの名前。
linenum	integer	ソースファイル内の、エラーが発生した行の番号。
context	character(100)	エラーの原因。
error	character(512)	エラーメッセージ。

サンプルクエリ

次の例では、STL_ERROR からエラー情報を取得します。

```
select process, errcode, linenum as line,
trim(error) as err
from stl_error;

   process    | errcode | line |                               err
--------------+---------+------
+------------------------------------------------------------------
 padbmaster   |    8001 |  194 | Path prefix: s3://redshift-downloads/testnulls/
venue.txt*
 padbmaster   |    8001 |  529 | Listing bucket=redshift-downloads prefix=tests/
category-csv-quotes
 padbmaster   |       2 |  190 | database "template0" is not currently accepting
 connections
 padbmaster   |      32 | 1956 | pq_flush: could not send data to client: Broken pipe
(4 rows)
```

STL_EXPLAIN

実行するために送信されたクエリの EXPLAIN プランを表示します。

STL_EXPLAIN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_EXPLAIN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
nodeid	integer	クエリの実行における 1 つ以上のステップに対応するノードの計画ノード識別子。
parentid	integer	親ノードの計画ノード識別子。親ノードには、いくつかの子ノードがあります。例えば、merge join は、結合されたテーブルに対する複数の scan の親です。
plannode	character(400)	EXPLAIN の出力のノードテキスト。コンピューティングノードでの実行を参照する計画ノードは、EXPLAIN 出力の先頭に **XN** が付けられます。
info	character(400)	計画ノードの修飾子およびフィルタ情報。例えば、この列には join の条件と WHERE 句の制限が含まれます。

サンプルクエリ

集計 join クエリの次の EXPLAIN 出力について考えます。

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
                                  QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

このクエリを実行し、そのクエリ ID が 10 の場合、EXPLAIN コマンドから返される情報のうち同じ種類のものを、STL_EXPLAIN テーブルを使用して確認できます。

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from stl_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring             |    substring
-----+--------+--------+--------------------------------+-------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO   | Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis    |
10   |      4 |      2 |       -> XN Seq Scan on sal    |
(4 rows)
```

次のクエリについて考えます。

```
select event.eventid, sum(pricepaid)
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

このクエリの ID が 15 の場合、次のシステムビュークエリは、完了された計画ノードを返します。この場合、ノードの順番は、実際の実行順序を示すために逆順にされます。

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from stl_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid|                          substring
-----+------+--------+--------------------------------------------------------
15   |    8 |      7 |                                -> XN Seq Scan on eve
15   |    7 |      5 |                          -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                          -> XN Seq Scan on sales(cos
```

```
15    |     5 |       4 |                    -> XN Hash Join DS_DIST_OUTER(cos
15    |     4 |       3 |                 -> XN HashAggregate(cost=862286577.07..
15    |     3 |       2 |           -> XN Sort(cost=1000862287175.47..10008622871
15    |     2 |       1 |   -> XN Network(cost=1000862287175.47..1000862287197.
15    |     1 |       0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

次のクエリは、ウィンドウ関数を含むすべてのクエリプランのクエリ ID を取得します。

```
select query, trim(plannode) from stl_explain
where plannode like '%Window%';

query|                                     btrim
-----+------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

STL_FILE_SCAN

COPY コマンドを使用してデータをロードした際に、Amazon Redshift により読み込まれたファイルを返します。

このビューをクエリすることで、データのロード時のエラーをトラブルシューティングすることができます。一般的にデータの並行ロードでは、1 つの COPY コマンドによって多くのファイルがロードされるため、STL_FILE_SCAN は、データの並行ロードにおける問題を特定する際に特に役立ちます。

STL_FILE_SCAN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_FILE_SCAN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_LOAD_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
name	character(90)	ロードされたファイルのフルパスおよび名前。
lines	bigint	ファイルから読み込まれた行数。
バイト	bigint	ファイルから読み込まれたバイト数。
loadtime	bigint	ファイルのロードにかかった時間 (マイクロ秒)。
curtime	タイムスタンプ	Amazon Redshift がファイルの処理を開始した時刻を表すタイムスタンプ。
is_partial	integer	COPY オペレーションの実行中に入力されたファイルが、いくつかの範囲で分割されていることを (true (1) で) 示す値。この値が false (0) の場合、入力ファイルは分割されていません。
start_offset	bigint	COPY オペレーション中に入力されたファイルが分割されていた場合、分割のオフセット値をバイト単位で示す値。ファイルが分割されていない場合、この値は 0 になります。

サンプルクエリ

次のクエリは、Amazon Redshift による読み込み時間が 1,000,000 マイクロ秒を超えたすべてのファイルの名前およびロード時間を取得します。

```
select trim(name)as name, loadtime from stl_file_scan
where loadtime > 1000000;
```

このクエリは、次の例のような出力を返します。

```
           name            | loadtime
---------------------------+----------
 listings_pipe.txt         |  9458354
 allusers_pipe.txt         |  2963761
 allevents_pipe.txt        |  1409135
 tickit/listings_pipe.txt  |  7071087
 tickit/allevents_pipe.txt |  1237364
 tickit/allusers_pipe.txt  |  2535138
 listings_pipe.txt         |  6706370
 allusers_pipe.txt         |  3579461
 allevents_pipe.txt        |  1313195
 tickit/allusers_pipe.txt  |  3236060
 tickit/listings_pipe.txt  |  4980108
(11 rows)
```

STL_HASH

ハッシュを実行するステップをクエリについて分析します。

STL_HASH はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_HASH には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。

列名	データ型	説明
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
slots	integer	ハッシュバケットの総数。
occupied	integer	レコードが含まれているスロットの総数。
maxlength	integer	最も大きなスロットのサイズ。
tbl	integer	テーブル ID。
is_diskbased	character(1)	true (t) の場合、クエリはディスクベースのオペレーションとして実行されました。false (f) の場合、クエリはメモリ内で実行されました。
workmem	bigint	ステップに割り当てられた作業メモリの総バイト数。
num_parts	integer	ハッシュステップでハッシュテーブルが分割された後のパーティションの総数。
est_rows	bigint	ハッシュされる行数の推定値。

列名	データ型	説明
num_blocks_permitted	integer	この情報は、内部使用に限定されています。
サイズ変更	integer	この情報は、内部使用に限定されています。
checksum	bigint	この情報は、内部使用に限定されています。
runtime_filter_size	integer	ランタイムフィルターのサイズ (バイト単位)。
max_runtime_filter_size	integer	ランタイムフィルターの最大サイズ (バイト単位)。

サンプルクエリ

次の例では、クエリ 720 のハッシュで使用されたパーティションの数に関する情報が返され、ディスク上で実行されたステップがないことが示されます。

```
select slice, rows, bytes, occupied, workmem, num_parts, est_rows,
 num_blocks_permitted, is_diskbased
from stl_hash
where query=720 and segment=5
order by slice;
```

```
 slice | rows | bytes  | occupied | workmem  | num_parts | est_rows |
 num_blocks_permitted | is_diskbased
-------+------+--------+----------+----------+-----------+----------
+----------------------+--------------
     0 |  145 | 585800 |        1 | 88866816 |        16 |        1 |
 52              f
     1 |    0 |      0 |        0 |        0 |        16 |        1 |
 52              f
(2 rows)
```

STL_HASHJOIN

ハッシュ結合を実行するステップをクエリについて分析します。

STL_HASHJOIN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_HASHJOIN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。

列名	データ型	説明
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
tbl	integer	テーブル ID。
num_parts	integer	ハッシュステップでハッシュテーブルが分割された後のパーティションの総数。
join_type	integer	ステップの結合タイプ。 • 0. クエリは内部結合を使用。 • 1. クエリは左外部結合を使用。 • 2. クエリは完全外部結合を使用。 • 3. クエリは右外部結合を使用。 • 4. クエリは UNION 演算子を使用。 • 5. クエリは IN 条件を使用。 • 6. この情報は、内部使用に限定されています。 • 7. この情報は、内部使用に限定されています。 • 8. この情報は、内部使用に限定されています。 • 9. この情報は、内部使用に限定されています。 • 10. この情報は、内部使用に限定されています。 • 11. この情報は、内部使用に限定されています。 • 12. この情報は、内部使用に限定されています。
hash_looped	character(1)	この情報は、内部使用に限定されています。
switched_parts	character(1)	ビルド (外部) 側とプローブ (内部) 側が入れ替えられたかどうかを示します。
used_prefetching	character(1)	この情報は、内部使用に限定されています。

列名	データ型	説明
hash_segment	integer	対応するハッシュステップのセグメント。
hash_step	integer	対応するハッシュステップのステップ番号。
checksum	bigint	この情報は、内部使用に限定されています。
ディストリビューション	integer	この情報は、内部使用に限定されています。

サンプルクエリ

次の例では、クエリ 720 のハッシュ結合で使用されたパーティションの数が返されます。

```
select query, slice, tbl, num_parts
from stl_hashjoin
where query=720 limit 10;
```

```
 query | slice | tbl | num_parts
-------+-------+-----+-----------
   720 |     0 | 243 |         1
   720 |     1 | 243 |         1
(2 rows)
```

STL_INSERT

挿入を実行するステップをクエリについて分析します。

STL_INSERT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_INSERT には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同

時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
tbl	integer	テーブル ID。
inserted_mega_value	character(1)	この情報は、内部使用に限定されています。この情報は、特定の挿入ステップで大きな値を挿入したかどうかを示します。大きな値は複数のブロックに格納されます。ブロックサイズはデフォルトで 1 MB で、大きな値とは、デフォルト設定で 1 MB より大きい値です。

サンプルクエリ

次の例は、最後に実行されたクエリの挿入実行ステップを返します。

```
select slice, segment, step, tasknum, rows, tbl
from stl_insert
where query=pg_last_query_id();
```

```
 slice | segment | step | tasknum | rows  |  tbl
-------+---------+------+---------+-------+--------
     0 |       2 |    2 |      15 | 24958 | 100548
     1 |       2 |    2 |      15 | 25032 | 100548
(2 rows)
```

STL_LIMIT

SELECT クエリ内で LIMIT 句が使用されるときに発生する実行ステップを分析します。

STL_LIMIT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

Note

STL_LIMIT には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。

列名	データ型	説明
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

STL_LIMIT に行を作成するために、この例ではまず、LIMIT 句を使用する次のクエリを VENUE テーブルに対して実行します。

```
select * from venue
order by 1
limit 10;
```

```
 venueid |         venuename          |    venuecity    | venuestate | venueseats
---------+----------------------------+-----------------+------------+------------
       1 | Toyota Park                | Bridgeview      | IL         |          0
       2 | Columbus Crew Stadium      | Columbus        | OH         |          0
       3 | RFK Stadium                | Washington      | DC         |          0
       4 | CommunityAmerica Ballpark  | Kansas City     | KS         |          0
       5 | Gillette Stadium           | Foxborough      | MA         |      68756
       6 | New York Giants Stadium    | East Rutherford | NJ         |      80242
       7 | BMO Field                  | Toronto         | ON         |          0
```

```
        8 | The Home Depot Center       | Carson          | CA          |           0
        9 | Dick's Sporting Goods Park | Commerce City   | CO          |           0
       10 | Pizza Hut Park              | Frisco          | TX          |           0
(10 rows)
```

次に、VENUE テーブルに対して最後に実行したクエリのクエリ ID を取得するために、次のクエリを実行します。

```
select max(query)
from stl_query;
```

```
  max
--------
 127128
(1 row)
```

オプションで次のクエリを実行することにより、このクエリ ID が実行済みの LIMIT クエリに対応していることを確認できます。

```
select query, trim(querytxt)
from stl_query
where query=127128;
```

```
 query  |                  btrim
--------+------------------------------------------
 127128 | select * from venue order by 1 limit 10;
(1 row)
```

最後に、次のクエリを実行して、LIMIT クエリに関する情報を STL_LIMIT テーブルから返します。

```
select slice, segment, step, starttime, endtime, tasknum
from stl_limit
where query=127128
order by starttime, endtime;
```

```
  slice | segment | step |         starttime          |          endtime           |
 tasknum
 -------+---------+------+----------------------------+----------------------------
+---------
```

```
       1 |         1 |     3 | 2013-09-06 22:56:43.608114 | 2013-09-06 22:56:43.609383 |
    15
       0 |         1 |     3 | 2013-09-06 22:56:43.608708 | 2013-09-06 22:56:43.609521 |
    15
   10000 |         2 |     2 | 2013-09-06 22:56:43.612506 | 2013-09-06 22:56:43.612668 |
     0
(3 rows)
```

STL_LOAD_COMMITS

データのロードを追跡またはトラブルシューティングするための情報を返します。

このビューには、各データファイルがデータベーステーブルにロードされるのに合わせて進捗が記録されます。

STL_LOAD_COMMITS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_LOAD_COMMITS には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_LOAD_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	このエントリでロードされるスライス。
name	character(256)	システム定義の値。

列名	データ型	説明
filename	character(256)	追跡されるファイルの名前。
byte_offset	integer	この情報は、内部使用に限定されています。
lines_scanned	integer	ロードされたファイルからスキャンされた行数。この数は、実際にロードされた行数とは一致しない可能性があります。例えば、ロードによってスキャンが実行されたとき、COPY コマンドの MAXERROR オプションに基づいて、いくつかの不良レコードが許容される可能性があります。
エラー	integer	この情報は、内部使用に限定されています。
curtime	timestamp	このエントリが最後に更新された時刻。
status	integer	この情報は、内部使用に限定されています。
file_format	character(16)	ロードファイルの形式。指定できる値は次のとおりです。 • Avro • JSON • ORC • Parquet • テキスト
is_partial	integer	COPY オペレーションの実行中に入力されたファイルが、いくつかの範囲で分割されていることを (true (1) で) 示す値。この値が false (0) の場合、入力ファイルは分割されていません。
start_offset	bigint	COPY オペレーション中に入力されたファイルが分割されていた場合、分割のオフセット値をバイト単位で示す値。各ファイル分割は、対応する start_offset 値を持つ個別のレコードとしてログ記録されます。ファイルが分割されていない場合、この値は 0 になります。
copy_job_id	bigint	コピージョブ識別子。`0` はジョブ ID がないことを示します。

サンプルクエリ

次の例は、前回の COPY 操作の詳細を返します。

```
select query, trim(filename) as file, curtime as updated
from stl_load_commits
where query = pg_last_copy_id();

 query |               file               |          updated
-------+----------------------------------+----------------------------
 28554 | s3://dw-tickit/category_pipe.txt | 2013-11-01 17:14:52.648486
(1 row)
```

次のクエリでは、TICKIT データベース内のテーブルについて、最新のロード状況を示すエントリが得られます。

```
select query, trim(filename), curtime
from stl_load_commits
where filename like '%tickit%' order by query;
```

```
 query |           btrim           |          curtime
-------+---------------------------+----------------------------
 22475 | tickit/allusers_pipe.txt  | 2013-02-08 20:58:23.274186
 22478 | tickit/venue_pipe.txt     | 2013-02-08 20:58:25.070604
 22480 | tickit/category_pipe.txt  | 2013-02-08 20:58:27.333472
 22482 | tickit/date2008_pipe.txt  | 2013-02-08 20:58:28.608305
 22485 | tickit/allevents_pipe.txt | 2013-02-08 20:58:29.99489
 22487 | tickit/listings_pipe.txt  | 2013-02-08 20:58:37.632939
 22593 | tickit/allusers_pipe.txt  | 2013-02-08 21:04:08.400491
 22596 | tickit/venue_pipe.txt     | 2013-02-08 21:04:10.056055
 22598 | tickit/category_pipe.txt  | 2013-02-08 21:04:11.465049
 22600 | tickit/date2008_pipe.txt  | 2013-02-08 21:04:12.461502
 22603 | tickit/allevents_pipe.txt | 2013-02-08 21:04:14.785124
 22605 | tickit/listings_pipe.txt  | 2013-02-08 21:04:20.170594

(12 rows)
```

レコードがこのシステムビューのログファイルに書き込まれていても、ロードがそれを含むトランザクションの一部として正しくコミットされているとは限りません。ロードのコミットを確認するには、STL_UTILITYTEXT ビューをクエリして、COPY トランザクションに対応する COMMIT

レコードを探します。例えば、このクエリは、STL_UTILITYTEXT に対するサブクエリに基づいて STL_LOAD_COMMITS と STL_QUERY を結合します。

```
select l.query,rtrim(l.filename),q.xid
from stl_load_commits l, stl_query q
where l.query=q.query
and exists
(select xid from stl_utilitytext where xid=q.xid and rtrim("text")='COMMIT');

 query |           rtrim           |  xid
-------+---------------------------+-------
 22600 | tickit/date2008_pipe.txt  | 68311
 22480 | tickit/category_pipe.txt  | 68066
  7508 | allusers_pipe.txt         | 23365
  7552 | category_pipe.txt         | 23415
  7576 | allevents_pipe.txt        | 23429
  7516 | venue_pipe.txt            | 23390
  7604 | listings_pipe.txt         | 23445
 22596 | tickit/venue_pipe.txt     | 68309
 22605 | tickit/listings_pipe.txt  | 68316
 22593 | tickit/allusers_pipe.txt  | 68305
 22485 | tickit/allevents_pipe.txt | 68071
  7561 | allevents_pipe.txt        | 23429
  7541 | category_pipe.txt         | 23415
  7558 | date2008_pipe.txt         | 23428
 22478 | tickit/venue_pipe.txt     | 68065
   526 | date2008_pipe.txt         |  2572
  7466 | allusers_pipe.txt         | 23365
 22482 | tickit/date2008_pipe.txt  | 68067
 22598 | tickit/category_pipe.txt  | 68310
 22603 | tickit/allevents_pipe.txt | 68315
 22475 | tickit/allusers_pipe.txt  | 68061
   547 | date2008_pipe.txt         |  2572
 22487 | tickit/listings_pipe.txt  | 68072
  7531 | venue_pipe.txt            | 23390
  7583 | listings_pipe.txt         | 23445
(25 rows)
```

次の例では、is_partial および start_offset 列の値が強調表示されています。

```
-- Single large file copy without scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
1
```

```
-- Single large uncompressed, delimited file copy with scan range
SELECT count(*) FROM stl_load_commits WHERE query = pg_last_copy_id();
16

-- Scan range offset logging in the file at 64MB boundary.
SELECT start_offset FROM stl_load_commits
WHERE query = pg_last_copy_id() ORDER BY start_offset;
0
67108864
134217728
201326592
268435456
335544320
402653184
469762048
536870912
603979776
671088640
738197504
805306368
872415232
939524096
1006632960
```

STL_LOAD_ERRORS

すべての Amazon Redshift ロードエラーのレコードを表示します。

STL_LOAD_ERRORS には、Amazon Redshift ロードエラーの履歴が含まれます。発生する可能性があるロードエラーと説明の包括的なリストについては、「ロードエラー参照」を参照してください。

STL_LOAD_ERRORS をクエリしてエラーに関する一般的な情報を得た後で、解析エラーが発生したデータの正確な行と列などの追加的な詳細を得るために STL_LOADERROR_DETAIL をクエリします。

STL_LOAD_ERRORS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_LOAD_ERRORS には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_LOAD_ERROR_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
slice	integer	エラーが発生したスライス。
tbl	integer	テーブル ID。
starttime	timestamp	UTC で表されたロードの開始時間。
session	integer	ロードを実行するセッションのセッション ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
filename	character(256)	ロードの入力ファイルへの完全なパス。
line_number	bigint	ロードファイル内の、エラーが発生した行の番号。JSON からの COPY の場合は、エラーのあった JSON オブジェクトの最終行の行番号。
colname	character(127)	エラーが発生したフィールド。
type	character(10)	フィールドのデータ型。
col_length	character(10)	定義されている場合、列の長さ。このフィールドは、データ型に長さの制限がある場合、値を持ちます。例えば、

列名	データ型	説明
		データ型が「character(3)」の場合、この列の値は「3」になります。
position	integer	フィールド内でのエラーの位置。
raw_line	character(1024)	エラーを含む生のロードデータ。ロードデータ内のマルチバイト文字は、ピリオドで置換されます。
raw_field_value	char(1024)	解析エラーを引き起こす「colname」フィールドの事前解析値。
err_code	integer	エラーコード。
err_reason	character(100)	エラーの説明。
is_partial	integer	COPY オペレーションの実行中に入力されたファイルが、いくつかの範囲で分割されていることを (true (1) で) 示す値。この値が false (0) の場合、入力ファイルは分割されていません。
start_offset	bigint	COPY オペレーション中に入力されたファイルが分割されていた場合、分割のオフセット値をバイト単位で示す値。ファイル内で不明な行番号は、「-1」として表示されます。ファイルが分割されていない場合、この値は 0 になります。
copy_job_id	bigint	コピージョブ識別子。0 はジョブ ID がないことを示します。

サンプルクエリ

次のクエリは、STL_LOAD_ERRORS を STL_LOADERROR_DETAIL に結合して、最後に実行されたロード中に発生したエラーの詳細を表示します。

```
select d.query, substring(d.filename,14,20),
d.line_number as line,
substring(d.value,1,16) as value,
substring(le.err_reason,1,48) as err_reason
```

```
from stl_loaderror_detail d, stl_load_errors le
where d.query = le.query
and d.query = pg_last_copy_id();

 query |    substring      | line |  value   |              err_reason
-------+-------------------+------+----------+----------------------------
    558| allusers_pipe.txt |  251 | 251      | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | ZRU29FGR | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Kaitlin  | String contains invalid or
                                               unsupported UTF8 code
    558| allusers_pipe.txt |  251 | Walter   | String contains invalid or
                                               unsupported UTF8 code
```

次の例では、STV_TBL_PERM と共に STL_LOAD_ERRORS を使用することにより、新しいビューを作成し、EVENT テーブルにデータをロードするときに発生したエラーをそのビューで確認します。

```
create view loadview as
(select distinct tbl, trim(name) as table_name, query, starttime,
trim(filename) as input, line_number, colname, err_code,
trim(err_reason) as reason
from stl_load_errors sl, stv_tbl_perm sp
where sl.tbl = sp.id);
```

次に、EVENT テーブルをロードするときに発生した最新のエラーを、次のクエリが返します。

```
select table_name, query, line_number, colname, starttime,
trim(reason) as error
from loadview
where table_name ='event'
order by line_number limit 1;
```

このクエリは、EVENT テーブルで発生した最新のロードエラーを返します。ロードエラーが発生していない場合、このクエリが返す行の数は 0 です。この例では、クエリは次のように 1 つのエラーを返します。

```
 table_name | query | line_number | colname | error | starttime
------+-----+----+----+--------------------------------------------------------
+----------------------
```

```
event | 309 |  0 |  5 | Error in Timestamp value or format [%Y-%m-%d %H:%M:%S] |
 2014-04-22 15:12:44

(1 row)
```

並列処理を容易にするために、COPY コマンドが自動的に非圧縮でテキスト区切りされた大きなファイルデータを分割する場合は、line_number、is_partial、および start_offset 列には、その分割処理に関する情報が表示されます。(元のファイルの行番号が使用できない場合は、行番号が不定になる可能性があります。)

```
--scan ranges information
SELECT line_number, POSITION, btrim(raw_line), btrim(raw_field_value),
btrim(err_reason), is_partial, start_offset FROM stl_load_errors
WHERE query = pg_last_copy_id();

--result
-1,51,"1008771|13463413|463414|2|28.00|38520.72|0.06|0.07|NO|1998-08-30|1998-09-25|
1998-09-04|TAKE BACK RETURN|RAIL|ans cajole sly","NO","Char length exceeds DDL
 length",1,67108864
```

STL_LOADERROR_DETAIL

COPY コマンドを使用したテーブルのロード中に発生したデータ解析エラーのログを表示します。ディスク領域を節約するために、各ロード操作に関してノードスライスあたり最大 20 件のエラーがログに記録されます。

解析エラーは、Amazon Redshift がデータ行をテーブルにロードするときに、データ行内のフィールドを解析できない場合に発生します。例えば、テーブルの中のある列が整数データ型を前提としており、データファイルではそのフィールドに文字列が含まれている場合、解析エラーが発生します。

STL_LOADERROR_DETAIL は、STL_LOAD_ERRORS をクエリしてエラーに関する一般的な情報を得た後で、解析エラーが発生したデータの正確な行と列などの追加的な詳細を得るためにクエリします。

STL_LOADERROR_DETAIL ビューには、解析エラーが発生した列を含む、そこまでのすべてのデータ列が含まれています。VALUE フィールドを使用すると、エラーまでに正しく解析された列と、エラーの列で実際に解析されたデータ値を確認することができます。

このビューはすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_LOADERROR_DETAIL には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_LOAD_ERROR_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
slice	integer	エラーが発生したスライス。
session	integer	ロードを実行するセッションのセッション ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
filename	character(256)	ロードの入力ファイルへの完全なパス。
line_number	bigint	ロードファイル内の、エラーが発生した行の番号。
field	integer	エラーが発生したフィールド。
colname	character(1024)	列名。
値	character(1024)	フィールドの解析済みデータ値 (切り詰められることがあります)。ロードデータ内のマルチバイト文字は、ピリオドで置換されます。
is_null	integer	解析された値が null であるかどうか。
type	character(10)	フィールドのデータ型。

列名	データ型	説明
col_length	character(10)	定義されている場合、列の長さ。このフィールドは、データ型に長さの制限がある場合、値を持ちます。例えば、データ型が「character(3)」の場合、この列の値は「3」になります。

サンプルクエリ

次のクエリは、STL_LOAD_ERRORS を STL_LOADERROR_DETAIL に結合して、テーブル ID が 100133 である EVENT テーブルのロード時に発生した解析エラーの詳細を表示します。

```
select d.query, d.line_number, d.value,
le.raw_line, le.err_reason
from stl_loaderror_detail d, stl_load_errors le
where
d.query = le.query
and tbl = 100133;
```

次のサンプル出力は、正常にロードされた列と、エラーのある列を示しています。この例では、3 番目の列で解析エラーが発生するまでに 2 つの列が正しくロードされ、3 番目の列では整数が前提となっているにもかかわらず文字列が解析されています。フィールドは整数であることが前提なので、初期化されていないデータである文字列「aaa」は null と解析されて、解析エラーが発生します。出力は、生の値、解析された値、およびエラー理由を示しています。

```
query  | line_number | value | raw_line | err_reason
-------+-------------+-------+----------+----------------
4      |      3      |  1201 |  1201    | Invalid digit
4      |      3      |   126 |   126    | Invalid digit
4      |      3      |       |   aaa    | Invalid digit
(3 rows)
```

クエリによって STL_LOAD_ERRORS と STL_LOADERROR_DETAIL が結合されると、データ行の各列にエラー理由が示されます。これは、その行でエラーが発生したことを示しているだけです。実際に解析エラーが発生しているのは、結果の最後の行です。

STL_MERGE

マージを実行するステップをクエリについて分析します。これらのステップは、並行した操作 (ソートと結合など) の結果が、その後の処理に備えてマージされると発生します。

STL_MERGE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_MERGE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。

列名	データ型	説明
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。

サンプルクエリ

次の例は、10 件のマージ実行結果を返します。

```
select query, step, starttime, endtime, tasknum, rows
from stl_merge
limit 10;
```

```
 query | step |       starttime       |        endtime        | tasknum | rows
-------+------+---------------------+---------------------+---------+------
     9 |    0 | 2013-08-12 20:08:14 | 2013-08-12 20:08:14 |       0 |    0
    12 |    0 | 2013-08-12 20:09:10 | 2013-08-12 20:09:10 |       0 |    0
    15 |    0 | 2013-08-12 20:10:24 | 2013-08-12 20:10:24 |       0 |    0
    20 |    0 | 2013-08-12 20:11:27 | 2013-08-12 20:11:27 |       0 |    0
    26 |    0 | 2013-08-12 20:12:28 | 2013-08-12 20:12:28 |       0 |    0
    32 |    0 | 2013-08-12 20:14:33 | 2013-08-12 20:14:33 |       0 |    0
    38 |    0 | 2013-08-12 20:16:43 | 2013-08-12 20:16:43 |       0 |    0
    44 |    0 | 2013-08-12 20:17:05 | 2013-08-12 20:17:05 |       0 |    0
    50 |    0 | 2013-08-12 20:18:48 | 2013-08-12 20:18:48 |       0 |    0
    56 |    0 | 2013-08-12 20:20:48 | 2013-08-12 20:20:48 |       0 |    0
(10 rows)
```

STL_MERGEJOIN

マージ結合を実行するステップをクエリについて分析します。

STL_MERGEJOIN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_MERGEJOIN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
tbl	integer	テーブル ID。これは、マージ結合で使用された内部テーブルの ID です。

列名	データ型	説明
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

次の例は、最後に実行したクエリのマージ結合結果を返します。

```
select sum(s.qtysold), e.eventname
from event e, listing l, sales s
where e.eventid=l.eventid
and l.listid= s.listid
group by e.eventname;

select * from stl_mergejoin where query=pg_last_query_id();
```

```
 userid | query | slice | segment | step |         starttime      |          endtime       |
 tasknum | rows | tbl
--------+-------+-------+---------+------+---------------------+---------------------
+---------+------+-----
    100 | 27399 |     3 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43428 | 240
    100 | 27399 |     0 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43159 | 240
    100 | 27399 |     2 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |42778 | 240
    100 | 27399 |     1 |       4 |    4 | 2013-10-02 16:30:41 | 2013-10-02 16:30:41 |
     19 |43091 | 240
```

STL_MV_STATE

STL_MV_STATE ビューには、マテリアライズドビューのすべてのステータス遷移の行が含まれています。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

STL_MV_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_MV_STATE でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	bigint	イベントを作成したユーザーの ID。
starttime	timestamp	イベントの開始時間。
xid	bigint	イベントのトランザクション ID。
event_desc	char(500)	ステータスの変更を促したイベント。いくつかの値の例は次のとおりです。 • 列のタイプが変更されました • 列が削除されました • 列の名前が変更されました • スキーマ名が変更されました • 小さいテーブルの変換 • TRUNCATE • Vacuum この列には他にも可能な値があることに注意してください。
db_name	char(128)	マテリアライズドビューを含むデータベース。
base_table_schema	char(128)	ベーステーブルのスキーマ。
base_table_name	char(128)	ベーステーブルの名前。
mv_schema	char(128)	マテリアライズドビューのスキーマ。

列名	データ型	説明
mv_name	char(128)	マテリアライズドビューの名前。
state	character(32)	マテリアライズドビューの変更済みステータスは次のとおりです。 • 再計算 • 更新不可

次の表は、event_desc および state の組み合わせ例を表しています。

```
          event_desc     |     state
-------------------------+---------------
 TRUNCATE                | Recompute
 TRUNCATE                | Recompute
 Small table conversion  | Recompute
 Vacuum                  | Recompute
 Column was renamed      | Unrefreshable
 Column was dropped      | Unrefreshable
 Table was renamed       | Unrefreshable
 Column type was changed | Unrefreshable
 Schema name was changed | Unrefreshable
```

サンプルクエリ

マテリアライズドビューのステータス遷移ログを表示するには、次のクエリを実行します。

```
select * from stl_mv_state;
```

このクエリは、次のサンプル出力を返します。

```
 userid |         starttime          | xid  |              event_desc        | db_name |
 base_table_schema   |    base_table_name     |       mv_schema        | mv_name         |
  state
--------+----------------------------+------+------------------------------+---------
+----------------------+----------------------+----------------------+---------------
+----------------
```

```
    138 | 2020-02-14 02:21:25.578885 | 5180 | TRUNCATE                    | dev      |
public              | mv_base_table        | public              | mv_test        |
Recompute
    138 | 2020-02-14 02:21:56.846774 | 5275 | Column was dropped          | dev      |
                    | mv_base_table        | public              | mv_test        |
Unrefreshable
    100 | 2020-02-13 22:09:53.041228 | 1794 | Column was renamed          | dev      |
                    | mv_base_table        | public              | mv_test        |
Unrefreshable
      1 | 2020-02-13 22:10:23.630914 | 1893 | ALTER TABLE ALTER SORTKEY   | dev      |
public              | mv_base_table_sorted | public              | mv_test        |
Recompute
      1 | 2020-02-17 22:57:22.497989 | 8455 | ALTER TABLE ALTER DISTSTYLE | dev      |
public              | mv_base_table        | public              | mv_test        |
Recompute
    173 | 2020-02-17 22:57:23.591434 | 8504 | Table was renamed           | dev      |
                    | mv_base_table        | public              | mv_test        |
Unrefreshable
    173 | 2020-02-17 22:57:27.229423 | 8592 | Column type was changed     | dev      |
                    | mv_base_table        | public              | mv_test        |
Unrefreshable
    197 | 2020-02-17 22:59:06.212569 | 9668 | TRUNCATE                    | dev      |
schemaf796e415850f4f | mv_base_table        | schemaf796e415850f4f | mv_test        |
Recompute
    138 | 2020-02-14 02:21:55.705655 | 5226 | Column was renamed          | dev      |
                    | mv_base_table        | public              | mv_test        |
Unrefreshable
      1 | 2020-02-14 02:22:26.292434 | 5325 | ALTER TABLE ALTER SORTKEY   | dev      |
public              | mv_base_table_sorted | public              | mv_test        |
Recompute
```

STL_NESTLOOP

ネステッドループ結合を実行するステップをクエリについて分析します。

STL_NESTLOOP はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_NESTLOOP には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
tbl	integer	テーブル ID。
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

次のクエリは CATEGORY テーブルの結合を無視するため、部分的なデカルト積を生成し、推奨されません。次の例はネステッドループを示しています。

```
select count(event.eventname), event.eventname, category.catname, date.caldate
from event, category, date
where event.dateid = date.dateid
group by event.eventname, category.catname, date.caldate;
```

次のクエリは、前のクエリの結果を stl_nestloop ビューから取得して表示します。

```
select query, slice, segment as seg, step,
datediff(msec, starttime, endtime) as duration, tasknum, rows, tbl
from stl_nestloop
where query = pg_last_query_id();
```

```
 query | slice | seg | step | duration | tasknum | rows  | tbl
-------+-------+-----+------+----------+---------+-------+-----
  6028 |     0 |   4 |    5 |       41 |      22 | 24277 | 240
  6028 |     1 |   4 |    5 |       26 |      23 | 24189 | 240
  6028 |     3 |   4 |    5 |       25 |      23 | 24376 | 240
  6028 |     2 |   4 |    5 |       54 |      22 | 23936 | 240
```

STL_PARSE

文字列をロード用のバイナリ値に解析するクエリステップを分析します。

STL_PARSE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_PARSE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。

サンプルクエリ

次の例では、文字列がバイナリ値に解析された、スライス 1、セグメント 0 のすべてのクエリステップの結果が返されます。

```
select query, step, starttime, endtime, tasknum, rows
from stl_parse
where slice=1 and segment=0;
```

```
 query | step |      starttime       |        endtime       | tasknum |  rows
-------+------+----------------------+----------------------+---------+--------
   669 |    1 | 2013-08-12 22:35:13 | 2013-08-12 22:35:17 |      32 | 192497
   696 |    1 | 2013-08-12 22:35:49 | 2013-08-12 22:35:49 |      32 |      0
```

```
    525 |     1 | 2013-08-12 22:32:03 | 2013-08-12 22:32:03 |       13 |  49990
    585 |     1 | 2013-08-12 22:33:18 | 2013-08-12 22:33:19 |       13 |    202
    621 |     1 | 2013-08-12 22:34:03 | 2013-08-12 22:34:03 |       27 |    365
    651 |     1 | 2013-08-12 22:34:47 | 2013-08-12 22:34:53 |       35 | 192497
    590 |     1 | 2013-08-12 22:33:28 | 2013-08-12 22:33:28 |       19 |      0
    599 |     1 | 2013-08-12 22:33:39 | 2013-08-12 22:33:39 |       31 |     11
    675 |     1 | 2013-08-12 22:35:26 | 2013-08-12 22:35:27 |       38 |   3766
    567 |     1 | 2013-08-12 22:32:47 | 2013-08-12 22:32:48 |       23 |  49990
    630 |     1 | 2013-08-12 22:34:17 | 2013-08-12 22:34:17 |       36 |      0
    572 |     1 | 2013-08-12 22:33:04 | 2013-08-12 22:33:04 |       29 |      0
    645 |     1 | 2013-08-12 22:34:37 | 2013-08-12 22:34:38 |       29 |   8798
    604 |     1 | 2013-08-12 22:33:47 | 2013-08-12 22:33:47 |       37 |      0
(14 rows)
```

STL_PLAN_INFO

行のセットに関するクエリの EXPLAIN 出力を確認するには、STL_PLAN_INFO ビューを使用します。これは、クエリプランを確認する代替的な方法となります。

STL_PLAN_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_PLAN_INFO には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。

列名	データ型	説明
nodeid	integer	クエリの実行における 1 つ以上のステップに対応するノードの計画ノード識別子。
segment	integer	クエリセグメントを識別する番号。
step	integer	クエリステップを識別する番号。
locus	integer	ステップが実行される場所。コンピューティングノード上にある場合は 0、リーダーノード上にある場合は 1。
plannode	integer	計画ノードの列挙値。計画ノードの列挙値については、次の表を参照してください (STL_EXPLAIN の PLANNODE 列には計画ノードのテキストが格納されます)。
startupcost	double precision	このステップの最初の行を返すのにかかる相対的コストの推定値。
totalcost	double precision	ステップの実行にかかる相対的コストの推定値。
rows	bigint	ステップによって生成される行数の推定値。
バイト	bigint	ステップによって生成されるバイト数の推定値。

サンプルクエリ

次の例は、EXPLAIN コマンドの使用と STL_PLAN_INFO ビューに対するクエリの実行によって返されるシンプルな SELECT クエリのクエリプランを比較します。

```
explain select * from category;
QUERY PLAN
-------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+--------------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
```

```
5 | Sports | MLS | Major League Soccer
...

select * from stl_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+-----------+------
+-------
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

この例では、PLANNODE 104 は CATEGORY テーブルのシーケンシャルスキャンを参照します。

```
select distinct eventname from event order by 1;

eventname
------------------------------------------------------------------------
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
-------------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from stl_plan_info where query=240 order by nodeid desc;
```

```
query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+------------------
+------------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
(10 rows)
```

STL_PROJECT

式を評価するために使用されるクエリステップの行が含まれます。

STL_PROJECT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_PROJECT には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。

列名	データ型	説明
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

次の例は、スライス 0、セグメント 1 の式を評価するために使用されたクエリステップのすべての行が返されます。

```
select query, step, starttime, endtime, tasknum, rows
from stl_project
where slice=0 and segment=1;
```

```
 query  | step |      starttime      |       endtime       | tasknum | rows
--------+------+---------------------+---------------------+---------+------
  86399 |    2 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
  86399 |    3 | 2013-08-29 22:01:21 | 2013-08-29 22:01:21 |      25 |   -1
    719 |    1 | 2013-08-12 22:38:33 | 2013-08-12 22:38:33 |       7 |   -1
  86383 |    1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |       7 |   -1
```

```
     714 |     1 | 2013-08-12 22:38:17 | 2013-08-12 22:38:17 |        2 |   -1
   86375 |     1 | 2013-08-29 21:57:59 | 2013-08-29 21:57:59 |        2 |   -1
   86397 |     2 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |       19 |   -1
     627 |     1 | 2013-08-12 22:34:13 | 2013-08-12 22:34:13 |       34 |   -1
   86326 |     2 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |       34 |   -1
   86326 |     3 | 2013-08-29 21:45:28 | 2013-08-29 21:45:28 |       34 |   -1
   86325 |     2 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |       28 |   -1
   86371 |     1 | 2013-08-29 21:57:42 | 2013-08-29 21:57:42 |        4 |   -1
  111100 |     2 | 2013-09-03 19:04:45 | 2013-09-03 19:04:45 |       12 |   -1
     704 |     2 | 2013-08-12 22:36:34 | 2013-08-12 22:36:34 |       37 |   -1
     649 |     2 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |       38 |   -1
     649 |     3 | 2013-08-12 22:34:47 | 2013-08-12 22:34:47 |       38 |   -1
     632 |     2 | 2013-08-12 22:34:22 | 2013-08-12 22:34:22 |       13 |   -1
     705 |     2 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |       13 |   -1
     705 |     3 | 2013-08-12 22:36:48 | 2013-08-12 22:36:49 |       13 |   -1
       3 |     1 | 2013-08-12 20:07:40 | 2013-08-12 20:07:40 |        3 |   -1
   86373 |     1 | 2013-08-29 21:57:58 | 2013-08-29 21:57:58 |        3 |   -1
  107976 |     1 | 2013-09-03 04:05:12 | 2013-09-03 04:05:12 |        3 |   -1
   86381 |     1 | 2013-08-29 21:58:35 | 2013-08-29 21:58:35 |        8 |   -1
   86396 |     1 | 2013-08-29 22:01:20 | 2013-08-29 22:01:20 |       15 |   -1
     711 |     1 | 2013-08-12 22:37:10 | 2013-08-12 22:37:10 |       20 |   -1
   86324 |     1 | 2013-08-29 21:45:27 | 2013-08-29 21:45:27 |       24 |   -1
(26 rows)
```

STL_QUERY

データベースクエリに関する実行情報を返します。

> **Note**
>
> STL_QUERY および STL_QUERYTEXT ビューには、クエリに関する情報だけが含まれており、他のユーティリティや DDL コマンドは含まれていません。Amazon Redshift によって実行されるすべてのステートメントのリストと情報については、STL_DDLTEXT および STL_UTILITYTEXT ビューもクエリできます。Amazon Redshift によって実行されるすべてのステートメントの完全なリストについては、SVL_STATEMENTTEXT ビューをクエリできます。

STL_QUERY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドの値は `default` になります。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。通常、セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、通常、この値は一定です。Amazon Redshift は特定の内部イベントに続いてアクティブなセッションを再起動し、新しい PID を割り当てる場合があります。詳細については、「STL_RESTARTED_SESSIONS」を参照してください。
database	character(32)	クエリが発行されたときにユーザーが接続されたデータベースの名前。
querytxt	character(4000)	クエリの実際のクエリテキスト。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁

列名	データ型	説明
		の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
aborted	integer	クエリがシステムによって停止されたかユーザーによってキャンセルされた場合、この列は **1** になります。クエリが (クライアントに結果を返すことも含めて) 最後まで実行された場合、この列は **0** になります。クライアントが結果を受け取る前に接続を解除した場合、クエリはバックエンドで正常に完了した場合でも、キャンセルされたものとしてマーク (**1**) されます。
insert_pristine	integer	現在のクエリの実行中に書き込みクエリが実行可能であるかどうか。1 = 書き込みクエリが許可されていません。0 = 書き込みクエリが許可されています。この列は、デバッグで使用することが意図されています。
concurrency_scaling_status	integer	クエリがメインクラスター、または同時実行スケーリングクラスターのどちらで実行されたかを示します。指定できる値は次のとおりです。 0 - メインクラスターで実行 1 - 同時実行スケーリングクラスターで実行 1 より大きい - メインクラスターで実行

サンプルクエリ

次のクエリは、最近実行された 5 つのクエリを表示します。

```
select query, trim(querytxt) as sqlquery
from stl_query
order by query desc limit 5;

query |                                    sqlquery
------+--------------------------------------------------
129 | select query, trim(querytxt) from stl_query order by query;
128 | select node from stv_disk_read_speeds;
```

```
127 | select system_status from stv_gui_status
126 | select * from systable_topology order by slice
125 | load global dict registry
(5 rows)
```

次のクエリは、2013 年 2 月 15 日に実行されたクエリを所要時間の降順で返します。

```
select query, datediff(seconds, starttime, endtime),
trim(querytxt) as sqlquery
from stl_query
where starttime >= '2013-02-15 00:00' and endtime < '2013-02-16 00:00'
order by date_diff desc;

 query | date_diff |  sqlquery
-------+-----------+-------------------------------------------
 55    |       119 | padb_fetch_sample: select count(*) from category
121    |         9 | select * from svl_query_summary;
181    |         6 | select * from svl_query_summary where query in(179,178);
172    |         5 | select * from svl_query_summary where query=148;
...
(189 rows)
```

以下のクエリは、クエリのキュー時間および実行時間を表示します。concurrency_scaling_status = 1 を使用したクエリは、同時実行スケーリングクラスターで実行されました。他のすべてのクエリは、メインクラスターで実行されました。

```
SELECT w.service_class AS queue
     , q.concurrency_scaling_status
     , COUNT( * ) AS queries
     , SUM( q.aborted )  AS aborted
     , SUM( ROUND( total_queue_time::NUMERIC / 1000000,2 ) ) AS queue_secs
     , SUM( ROUND( total_exec_time::NUMERIC / 1000000,2 ) )  AS exec_secs
FROM stl_query q
     JOIN stl_wlm_query w
          USING (userid,query)
WHERE q.userid > 1
  AND service_class > 5
  AND q.starttime > '2019-03-01 16:38:00'
  AND q.endtime   < '2019-03-01 17:40:00'
GROUP BY 1,2
ORDER BY 1,2;
```

STL_QUERY_METRICS

ユーザー定義のクエリキュー (サービスクラス) でアクティブに実行され、完了したクエリについて、処理される列数、CPU 使用率、入出力、ディスク使用率などのメトリクス情報を含みます。現在実行されているアクティブなクエリのメトリクスを表示するには、STV_QUERY_METRICS システムビューを参照してください。

クエリメトリクスは、1 秒間隔でサンプリングされます。結果として、同じクエリが複数実行され、わずかに異なる時刻を返す場合があります。また、1 秒未満で実行されるクエリセグメントは記録されない場合があります。

STL_QUERY_METRICS は、クエリレベル、セグメントレベル、およびステップレベルでメトリクスを追跡および集計します。クエリセグメントとステップの詳細については、「クエリプランと実行ワークフロー」を参照してください。多くのメトリクス (`max_rows`、`cpu_time`など) は、ノードスライスを超えて合計されます。ノードスライスの詳細については、「データウェアハウスシステムのアーキテクチャ」を参照してください。

列がメトリクスをレポートするレベルを確認するには、`segment`列および `step_type` 列を調べます。

- `segment` と `step_type` の両方が `-1` である場合、列はクエリレベルでメトリクスをレポートします。
- `segment` が `-1` でなく、`step_type`が `-1` である場合、列はメトリクスをセグメントレベルでレポートします。
- `segment` と `step_type` の両方が `-1` でない場合、列はステップレベルでメトリクスをレポートします。

SVL_QUERY_METRICS ビューと SVL_QUERY_METRICS_SUMMARY ビューは、このビューのデータを集計し、より便利な形式で情報を表示します。

STL_QUERY_METRICS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したクエリを実行したユーザーの ID。
service_class	integer	サービスクラスの ID。クエリキューは WLM 設定で定義されます。メトリクスはユーザー定義のキューについてのみレポートされます。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。セグメント値が -1 である場合は、メトリクスセグメントの値はクエリレベルまでロールアップされます。
step_type	integer	実行されたステップのタイプ。ステップタイプの説明については、「ステップタイプ」を参照してください。
starttime	timestamp	UTC で表されたクエリの実行開始時刻。秒の小数部の精度 (6 桁) を使用します。例: `2009-06-12 11:29:19.131358` 。
スライス	integer	クラスターのスライスの数。
max_rows	bigint	ステップの列出力の最大数 (すべてのスライスにわたる集計値)。
rows	bigint	ステップに処理された行数。
max_cpu_time	bigint	CPU の最長使用時間 (ミリ秒)。セグメントレベルでは、すべてのスライスにわたるセグメントによる CPU の最長使用時間。クエリレベルでは、任意のクエリセグメントによる CPU の最長使用時間。

列名	データ型	説明
cpu_time	bigint	CPU の使用時間 (ミリ秒)。セグメントレベルでは、すべてのスライスにわたるセグメントの CPU 時間の合計。クエリレベルでは、すべてのスライスとセグメントにわたるクエリの CPU 時間の合計。
max_blocks_read	bigint	セグメントに読み取られた 1 MB ブロックの最大数 (すべてのスライスにわたる集計値)。セグメントレベルでは、すべてのスライスにわたるセグメントによって読み取られた 1 MB ブロックの最大数。クエリレベルでは、任意のクエリセグメントによって読み取られた 1 MB ブロックの最大数。
blocks_read	bigint	クエリまたはセグメントに読み取られた 1 MB ブロックの数。
max_run_time	bigint	セグメントの最長経過時間 (ミリ秒) セグメントレベルでは、すべてのスライスにわたるセグメントの最長実行時間。クエリレベルでは、任意のクエリセグメントの最長実行時間。
run_time	bigint	すべてのスライスにわたって集計された合計実行時間。実行時間に待機時間は含まれません。 セグメントレベルでは、すべてのスライスにわたって合計されたセグメントの実行時間。クエリレベルでは、全てのスライスとセグメントにわたって合計されたクエリの実行時間。合計値であるため、実行時間はクエリの実行時間とは関連しません。
max_blocks_to_disk	bigint	中間結果の書き込みに使用する最大ディスク容量 (MB ブロック)。セグメントレベルでは、全てのスライスにわたるセグメントによって使用された最大ディスク容量。クエリレベルでは、任意のクエリセグメントによって使用された最大ディスク容量。
blocks_to_disk	bigint	クエリまたはセグメントが中間結果の書き込みに使用するディスク容量 (MB ブロック)。
step	integer	実行されたクエリステップ。

列名	データ型	説明
max_query_scan_size	bigint	クエリによってスキャンされた最大データサイズ (MB)。セグメントレベルでは、すべてのスライスにわたるセグメントによってスキャンされたデータの最大サイズ。クエリレベルでは、任意のクエリセグメントによってスキャンされたデータの最大サイズ。
query_scan_size	bigint	クエリによってスキャンされたデータサイズ (MB)。
query_priority	integer	クエリの優先度です。有効な値は、-1、0、1、2、3、および 4 です。ここで、-1 は、クエリ優先度がサポートされていないことを意味します。
query_queue_time	bigint	クエリがキューに入れられた時間 (マイクロ秒)。
service_class_name	character (64)	サービスクラスの名前。

サンプルクエリ

CPU 時間の長いクエリ (1,000 秒以上) を検索するには、次のクエリを実行します。

```
Select query, cpu_time / 1000000 as cpu_seconds
from stl_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
25775 |        9540
```

ネストされたループ結合を持ち、100 万以上の行を返したクエリを検索するには、次のクエリを実行します。

```
select query, rows
from stl_query_metrics
where step_type = 15 and rows > 1000000
order by rows;
```

```
query | rows
------+-----------
25775 | 2621562702
```

60 秒以上実行されていて CPU 時間が 10 秒以下のアクティブなクエリを検索するには、次のクエリを実行します。

```
select query, run_time/1000000 as run_time_seconds
from stl_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
------+-----------------
25775 |             114
```

STL_QUERYTEXT

SQL コマンドのクエリテキストを取得します。

STL_QUERYTEXT ビューをクエリして、次のステートメントに関して記録された SQL を取得します。

- SELECT、SELECT INTO
- INSERT、UPDATE、DELETE
- COPY
- UNLOAD
- VACUUM と ANALYZE を実行することによって生成されたクエリ
- CREATE TABLE AS (CTAS)

これらのステートメントに関する一定期間のアクティビティをクエリするには、STL_QUERYTEXT ビューと STL_QUERY ビューを結合します。

> **Note**
>
> STL_QUERY および STL_QUERYTEXT ビューには、クエリに関する情報だけが含まれており、他のユーティリティや DDL コマンドは含まれていません。Amazon Redshift によっ

て実行されるすべてのステートメントのリストと情報については、STL_DDLTEXT および STL_UTILITYTEXT ビューもクエリできます。Amazon Redshift によって実行されるすべてのステートメントの完全なリストについては、SVL_STATEMENTTEXT ビューをクエリできます。

STL_DDLTEXT、STL_UTILITYTEXT、SVL_STATEMENTTEXT も参照してください。

STL_QUERYTEXT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_TEXT でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。通常、セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、通常、この値は一定です。Amazon Redshift は特定の内部イベントに続いてアクティブなセッションを再起動し、新しい PID を割り当てる場合があります。詳細については、「STL_RESTARTED_SESSIONS」を参照してください。この列を使用して、STL_ERROR ビューに結合できます。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
sequence	integer	1 つのステートメントに含まれる文字数が 200 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。

列名	データ型	説明
text	character(200)	200 文字単位の SQL テキスト。このフィールドには、バックスラッシュ (\\) や改行 (\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

PG_BACKEND_PID() 関数を使用して、現在のセッションに関する情報を取得できます。例えば、以下のクエリは、現在のセッションで完了されたクエリのクエリ ID とクエリテキストの一部を返します。

```
select query, substring(text,1,60)
from stl_querytext
where pid = pg_backend_pid()
order by query desc;

 query |                         substring
-------+--------------------------------------------------------------
 28262 | select query, substring(text,1,80) from stl_querytext where
 28252 | select query, substring(path,0,80) as path from stl_unload_l
 28248 | copy category from 's3://dw-tickit/manifest/category/1030_ma
 28247 | Count rows in target table
 28245 | unload ('select * from category') to 's3://dw-tickit/manifes
 28240 | select query, substring(text,1,40) from stl_querytext where
(6 rows)
```

ストアド SQL の再構築

STL_QUERYTEXT の `text` 列に保存されている SQL を再構築するには、SELECT ステートメントを実行して、`text`列の 1 つ以上の部分から SQL を作成します。再構築された SQL を実行する前に、特殊文字 (\n) がある場合は、改行に置き換えます。次の SELECT ステートメントの結果は、`query_statement`フィールドに再構築された SQL の行です。

```
SELECT query, LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END)
 WITHIN GROUP (ORDER BY sequence) as query_statement, COUNT(*) as row_count
FROM stl_querytext GROUP BY query ORDER BY query desc;
```

例えば、次のクエリでは 3 つの列を選択します。クエリ自体は 200 文字より長く、STL_QUERYTEXT の部分に保存されます。

```
select
1 AS a0123456789012345678901234567890123456789012345678901234567890,
2 AS b0123456789012345678901234567890123456789012345678901234567890,
3 AS b012345678901234567890123456789012345678901234
FROM stl_querytext;
```

この例では、クエリは STL_QUERYTEXT の `text` 列の 2 つの部分 (行) に保存されます。

```
select query, sequence, text
from stl_querytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```


query  | sequence |
                         text

-------+----------
+----------------------------------------------------------------------------------------------
    45 |        0 | select\n1 AS
 a0123456789012345678901234567890123456789012345678901234567890,\n2 AS
 b0123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234
    45 |        1 | \nFROM stl_querytext;
```

STL_QUERYTEXT に保存された SQL を再構築するには、次の SQL を実行します。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
from stl_querytext where query=pg_last_query_id();
```

再構築された SQL をクライアントで使用するには、特殊文字 (\n) を改行に置き換えます。

```



                      text

-----------------------------------------------------------------------------------------------
 select\n1 AS a0123456789012345678901234567890123456789012345678901234567890,
\n2 AS b0123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234\nFROM stl_querytext;
```

STL_REPLACEMENTS

無効な UTF-8 文字が COPY コマンドの ACCEPTINVCHARS オプションによって置き換えられたときのログを表示します。STL_REPLACEMENTS へのログのエントリは、少なくとも 1 つの置き換えが必要だった各ノードスライスについて、それぞれ最初の 100 行について 1 件追加されます。

STL_REPLACEMENTS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_NESTLOOP には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_COPY_REPLACEMENTS を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	置き換えが発生したノードスライス番号。
tbl	integer	テーブル ID。
starttime	timestamp	UTC で表された COPY コマンドの開始時刻。
session	integer	COPY コマンドを実行するセッションのセッション ID。
filename	character (256)	COPY コマンドの入力ファイルへの完全なパス。

列名	データ型	説明
line_number	bigint	入力データファイル内で無効な UTF-8 文字を含んでいた行の番号。-1 は、列データファイルからコピーする場合など、行番号を使用できないことを示します。
colname	character (127)	無効な UTF-8 文字を含んでいた最初のフィールド。
raw_line	character (1024)	無効な UTF-8 文字を含んでいた生のロードデータ。

サンプルクエリ

次の例は、最後に実行された COPY 操作で置き換えられた文字を返します。

```
select query, session, filename, line_number, colname
from stl_replacements
where query = pg_last_copy_id();

 query | session |   filename                                       | line_number | colname
 ------+---------+--------------------------------------------------+-------------+--------
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt        |         251 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt        |         317 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt        |         569 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt        |         623 | city
    96 |    6314 | s3://DOC-EXAMPLE-BUCKET/allusers_pipe.txt        |         694 | city
...
```

STL_RESTARTED_SESSIONS

特定の内部イベント後に継続的な可用性を維持するため、Amazon Redshift は新しいプロセス ID (PID) でアクティブなセッションを再開する場合があります。Amazon Redshift がセッションを再起動すると、STL_RESTARTED_SESSIONS は新しい PID と古い PID を記録します。

詳細については、このセクションに続く例を参照してください。

STL_RESTARTED_SESSIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_SESSION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
currenttime	timestamp	イベントの時刻。
dbname	character (50)	セッションに関連付けられたデータベースの名前。
newpid	integer	再起動したセッションのプロセス ID。
oldpid	integer	元のセッションのプロセス ID。
username	character (50)	セッションに関連付けられたユーザーの名前。
remotehost	character (45)	リモートホストの名前または IP アドレス。
remoteport	character (32)	リモートホストのポート番号。
parkedtime	timestamp	この情報は、内部使用に限定されています。
session_vars	character (2000)	この情報は、内部使用に限定されています。

サンプルクエリ

次の例では、STL_RESTARTED_SESSIONS を STL_SESSIONS に結合し、再起動したセッションのユーザー名を表示します。

```
select process, stl_restarted_sessions.newpid, user_name
from stl_sessions
inner join stl_restarted_sessions on stl_sessions.process =
 stl_restarted_sessions.oldpid
```

```
order by process;

...
```

STL_RETURN

クエリに含まれるリターンステップの詳細を格納します。リターンステップは、コンピューティングノードで完了されたクエリの結果をリーダーノードに返します。リーダーノードは受け取ったデータをマージして、その結果を要求元クライアントに返します。リーダーノードで完了されたクエリについて、リターンステップはクライアントに結果を返します。

複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。詳細については、「クエリ処理」を参照してください。

STL_RETURN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_RETURN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。

列名	データ型	説明
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
packets	integer	ネットワークを介して送信されたパケットの総数。
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

以下のクエリは、各スライスで、最近のクエリのどのステップが実行されたかを表示します。

```
SELECT query, slice, segment, step, endtime, rows, packets
from stl_return where query = pg_last_query_id();

 query |  slice | segment | step |          endtime           | rows | packets
-------+--------+---------+------+----------------------------+------+---------
     4 |      2 |       3 |    2 | 2013-12-27 01:43:21.469043 |    3 |       0
     4 |      3 |       3 |    2 | 2013-12-27 01:43:21.473321 |    0 |       0
     4 |      0 |       3 |    2 | 2013-12-27 01:43:21.469118 |    2 |       0
     4 |      1 |       3 |    2 | 2013-12-27 01:43:21.474196 |    0 |       0
     4 |      4 |       3 |    2 | 2013-12-27 01:43:21.47704  |    2 |       0
     4 |      5 |       3 |    2 | 2013-12-27 01:43:21.478593 |    0 |       0
     4 |   12811|       4 |    1 | 2013-12-27 01:43:21.480755 |    0 |       0
(7 rows)
```

STL_S3CLIENT

転送時間などのパフォーマンスメトリクスを記録します。

STL_S3CLIENT テーブルは、Amazon S3 からデータを転送するときにかかった時間を調べるときに使用します。

STL_S3CLIENT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
recordtime	timestamp	レコードが記録された時刻。
pid	integer	プロセス ID。セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、この値は一定です。
http_method	character (64)	Amazon S3 リクエストに対応する HTTP メソッド名。
bucket	character (64)	S3 バケット名。
key	character (256)	Amazon S3 オブジェクトに対応するキー。
transfer_size	bigint	転送バイト数。

列名	データ型	説明
data_size	bigint	データのバイト数。非圧縮データの場合、この値は transfer_size と同じです。圧縮が使用された場合、これは圧縮前のデータのサイズです。
start_time	bigint	転送が開始された時刻 (マイクロ秒単位、2000 年 1 月 1 日から)。
end_time	bigint	転送が終了した時刻 (マイクロ秒単位、2000 年 1 月 1 日から)。
transfer_time	bigint	転送にかかった時間 (マイクロ秒単位)。
compression_time	bigint	転送時間のうち、データの展開にかかった時間 (マイクロ秒単位)。
connect_time	bigint	開始から、リモートサーバーへの接続が完了するまでの時間 (マイクロ秒単位)。
app_connect_time	bigint	開始から、リモートホストとの SSL 接続/ハンドシェイクが完了するまでの時間 (マイクロ秒単位)。
retries	bigint	転送が再試行された回数。
request_id	char(32)	Amazon S3 HTTP レスポンスヘッダー内のリクエスト ID
extended_request_id	char(128)	Amazon S3 HTTP ヘッダーのレスポンスから拡張されたリクエスト ID (x-amz-id-2)。
ip_address	char(64)	サーバーの IP アドレス (IPv4 または IPv6)。
is_partial	integer	COPY オペレーションの実行中に入力されたファイルが、いくつかの範囲で分割されていることを (true (1) で) 示す値。この値が false (0) の場合、入力ファイルは分割されていません。
start_offset	bigint	COPY オペレーション中に入力されたファイルが分割されていた場合、分割のオフセット値をバイト単位で示す値。ファイルが分割されていない場合、この値は 0 になります。

サンプルクエリ

次のクエリは、COPY コマンドを使用してファイルをロードするためにかかった時間を返します。

```
select slice, key, transfer_time
from stl_s3client
where query = pg_last_copy_id();
```

結果

```
 slice |   key                         | transfer_time
 ------+-----------------------------+---------------
     0 | listing10M0003_part_00      |    16626716
     1 | listing10M0001_part_00      |    12894494
     2 | listing10M0002_part_00      |    14320978
     3 | listing10M0000_part_00      |    11293439
  3371 | prefix=listing10M;marker=   |       99395
```

次の例では、start_timeおよび end_time をタイムスタンプに変換します。

```
select userid,query,slice,pid,recordtime,start_time,end_time,
'2000-01-01'::timestamp + (start_time/1000000.0)* interval '1 second' as start_ts,
'2000-01-01'::timestamp + (end_time/1000000.0)* interval '1 second' as end_ts
from stl_s3client where query> -1 limit 5;
```

```
 userid | query | slice |  pid  |         recordtime         |   start_time    |
 end_time      |          start_ts          |           end_ts
--------+-------+-------+-------+----------------------------+-----------------
+-----------------+----------------------------+----------------------------
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.207839 | 616436837154256 |
 616436837207838 | 2019-07-14 16:27:17.154256 | 2019-07-14 16:27:17.207838
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.252521 | 616436837208208 |
 616436837252520 | 2019-07-14 16:27:17.208208 | 2019-07-14 16:27:17.25252
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.284376 | 616436837208460 |
 616436837284374 | 2019-07-14 16:27:17.20846  | 2019-07-14 16:27:17.284374
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.285307 | 616436837208980 |
 616436837285306 | 2019-07-14 16:27:17.20898  | 2019-07-14 16:27:17.285306
      0 |     0 |     0 | 23449 | 2019-07-14 16:27:17.353853 | 616436837302216 |
 616436837353851 | 2019-07-14 16:27:17.302216 | 2019-07-14 16:27:17.353851
```

STL_S3CLIENT_ERROR

スライスで Amazon S3 からファイルをロードするときに発生したエラーを記録します。

STL_S3CLIENT_ERROR は、COPY コマンドで Amazon S3 からデータを転送するときに発生したエラーの詳細を調べるために使用します。

STL_S3CLIENT_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。クエリ ID -1 は内部用です。
sliceid	integer	クエリが実行されていたスライスを識別する番号。
recordtime	timestamp	レコードが記録された時刻。
pid	integer	プロセス ID。セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、この値は一定です。
http_method	character (64)	Amazon S3 リクエストに対応する HTTP メソッド名。
bucket	character (64)	Amazon S3 バケット名。
key	character (256)	Amazon S3 オブジェクトに対応するキー。
error	character (1024)	エラーメッセージ。

列名	データ型	説明
is_partial	integer	COPY オペレーションの実行中に入力されたファイルが、いくつかの範囲で分割されていることを (true (1) で) 示す値。この値が false (0) の場合、入力ファイルは分割されていません。
start_offset	bigint	COPY オペレーション中に入力されたファイルが分割されていた場合、分割のオフセット値をバイト単位で示す値。ファイルが分割されていない場合、この値は 0 になります。

使用に関する注意事項

「接続タイムアウト」として複数のエラーが発生する場合は、ネットワークに問題がある可能性があります。拡張された VPC のルーティングを使用している場合は、クラスターの VPC とデータリソースの間のネットワークパスが有効であることを確認してください。詳細については、「Amazon Redshift の拡張 VPC ルーティング」を参照してください。

サンプルクエリ

以下のクエリは、現在のセッション中に完了された COPY コマンドからのエラーを返します。

```
select query, sliceid, substring(key from 1 for 20) as file,
substring(error from 1 for 35) as error
from stl_s3client_error
where pid = pg_backend_pid()
order by query desc;
```

結果

```
 query  | sliceid |     file           |              error

--------+---------+--------------------+------------------------------------
 362228 |      12 | part.tbl.25.159.gz | transfer closed with 1947655 bytes
 362228 |      24 | part.tbl.15.577.gz | transfer closed with 1881910 bytes
 362228 |       7 | part.tbl.22.600.gz | transfer closed with 700143 bytes r
 362228 |      22 | part.tbl.3.34.gz   | transfer closed with 2334528 bytes
 362228 |      11 | part.tbl.30.274.gz | transfer closed with 699031 bytes r
 362228 |      30 | part.tbl.5.509.gz  | Unknown SSL protocol error in conne
```

```
 361999 |       10 | part.tbl.23.305.gz | transfer closed with 698959 bytes r
 361999 |       19 | part.tbl.26.582.gz | transfer closed with 1881458 bytes
 361999 |        4 | part.tbl.15.629.gz | transfer closed with 2275907 bytes
 361999 |       20 | part.tbl.6.456.gz  | transfer closed with 692162 bytes r
(10 rows)
```

STL_SAVE

クエリに含まれる保存ステップの詳細を格納します。保存ステップは、入力ストリームを一時テーブルに保存します。一時テーブルは、クエリ実行中の中間結果が格納される一時的なテーブルです。

複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。詳細については、「クエリ処理」を参照してください。

STL_SAVE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_SAVE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。

列名	データ型	説明
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
tbl	integer	マテリアライズされた一時テーブルの ID。
is_diskbased	character(1)	クエリのこのステップがディスクベースのオペレーションとして実行されたかどうか: true (t) または false (f)。
workmem	bigint	ステップに割り当てられた作業メモリのバイト数。

サンプルクエリ

以下のクエリは、各スライスで、最近のクエリのどの保存ステップが実行されたかを表示します。

```
select query, slice, segment, step, tasknum, rows,  tbl
from stl_save where query = pg_last_query_id();

 query | slice | segment | step | tasknum | rows | tbl
-------+-------+---------+------+---------+------+-----
 52236 |     3 |       0 |    2 |      21 |    0 | 239
 52236 |     2 |       0 |    2 |      20 |    0 | 239
 52236 |     2 |       2 |    2 |      20 |    0 | 239
 52236 |     3 |       2 |    2 |      21 |    0 | 239
 52236 |     1 |       0 |    2 |      21 |    0 | 239
 52236 |     0 |       0 |    2 |      20 |    0 | 239
 52236 |     0 |       2 |    2 |      20 |    0 | 239
```

```
  52236 |      1 |        2 |     2 |       21 |     0 | 239
(8 rows)
```

STL_SCAN

テーブルをスキャンするステップをクエリについて分析します。スキャンは 1 つのセグメント内で最初のステップなので、このテーブルに含まれる行のステップ番号は常に 0 です。

STL_SCAN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_SCAN には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。

列名	データ型	説明
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
fetches	bigint	この情報は、内部使用に限定されています。
type	integer	スキャンタイプの ID。有効な値のリストについては、次の表を参照してください。
tbl	integer	テーブル ID。
is_rrscan	character(1)	true (t) の場合は、ステップで範囲限定スキャンが使用されたことを示します。
is_delayed_scan	character(1)	この情報は、内部使用に限定されています。
rows_pre_filter	bigint	永続テーブルのスキャンの場合は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前でユーザー定義のクエリフィルタが適用される前に出力された合計行数。
rows_pre_user_filter	bigint	永続テーブルのスキャンの場合は、削除対象としてマークされた行 (非実体の行) をフィルタリングした後でユーザー定義のクエリフィルタが適用される前に処理された行数。
perm_table_name	character(136)	永続テーブルのスキャンの場合は、スキャンされたテーブルの名前。
is_rlf_scan	character(1)	true (t) の場合は、ステップで低レベルフィルタリングが使用されたことを示します。

列名	データ型	説明
is_rlf_scan_reason	integer	この情報は、内部使用に限定されています。
num_em_blocks	integer	この情報は、内部使用に限定されています。
checksum	bigint	この情報は、内部使用に限定されています。
runtime_filtering	character(1)	true(t) の場合、ランタイムフィルタが適用されていることを示します。
scan_region	integer	この情報は、内部使用に限定されています。
num_sortkey_as_predicate	integer	この情報は、内部使用に限定されています。
row_fetcher_state	integer	この情報は、内部使用に限定されています。
consumed_scan_ranges	bigint	この情報は、内部使用に限定されています。
work_stealing_reason	bigint	この情報は、内部使用に限定されています。
is_vectorized_scan	character(1)	この情報は、内部使用に限定されています。
is_vectorized_scan_reason	integer	この情報は、内部使用に限定されています。

列名	データ型	説明
row_fetcher_reason	bigint	この情報は、内部使用に限定されています。
topology_signature	bigint	この情報は、内部使用に限定されています。
use_tpm_partition	character(1)	この情報は、内部使用に限定されています。
is_rrscan_expr	character(1)	この情報は、内部使用に限定されています。
scanned_mega_value	character(1)	この情報は、内部使用に限定されています。この情報は、特定のスキャンステップで大きな値をスキャンしたかどうかを示します。大きな値は複数のブロックに格納されます。ブロックサイズはデフォルトで 1 MB で、大きな値とは、デフォルト設定で 1 MB より大きい値です。

スキャンタイプ

タイプ ID	Description
1	ネットワークからのデータ。
2	圧縮された共有メモリ内のパーマネントユーザーテーブル。
3	行が認識される一時テーブル。
21	Amazon S3 からファイルをロードします。
22	Amazon DynamoDB からテーブルをロードします。
23	リモート SSH 接続からのロードデータ。
24	リモートクラスター (ソート済みリージョン) からのロードデータ。これは、サイズを変更するために使用されます。

タイプ ID	Description
25	リモートクラスター (未ソートリージョン) からのロードデータ。これは、サイズを変更するために使用されます。
28	複数のテーブルの UNION ALL を使用して時系列ビューからデータを読み取ります。
29	Amazon S3 外部テーブルからデータを読み取ります。
30	Amazon S3 外部テーブルのパーティション情報を読み取ります。
33	リモート Postgres テーブルからデータを読み取ります。
36	リモート MySQL テーブルからデータを読み取ります。
37	リモート Kinesis ストリームからデータを読み取ります。

使用に関する注意事項

`rows` が `rows_pre_filter` と相対的に近いことが理想的です。`rows` と `rows_pre_filter` の間に大きな差異がある場合は、後で破棄される行を実行エンジンがスキャンしていることが考えられ、非効率です。`rows_pre_filter` と `rows_pre_user_filter` の差異は、スキャンに含まれる非実体行の数に相当します。削除対象としてマークされた行を削除するには、VACUUM を実行します。`rows` と `rows_pre_user_filter` の差異は、クエリによってフィルタリングされる行の数に相当します。多数の行がユーザーフィルタによって破棄される場合は、ソート列の選択を見直すか、未ソートリージョンが大きいことが原因である場合は、バキュームを実行します。

サンプルクエリ

次の例は、バキュームされていない行 (非実体行) がテーブルによって削除されたため、`rows_pre_filter` が `rows_pre_user_filter` よりも大きいことを示しています。

```
SELECT query, slice, segment,step,rows, rows_pre_filter, rows_pre_user_filter
from stl_scan where query = pg_last_query_id();

 query |  slice | segment | step | rows  | rows_pre_filter | rows_pre_user_filter
-------+--------+---------+------+-------+-----------------+----------------------
 42915 |      0 |       0 |    0 | 43159 |           86318 |                43159
 42915 |      0 |       1 |    0 |     1 |               0 |                    0
```

```
 42915 |      1 |        0 |      0 | 43091 |              86182 |                  43091
 42915 |      1 |        1 |      0 |     1 |                  0 |                      0
 42915 |      2 |        0 |      0 | 42778 |              85556 |                  42778
 42915 |      2 |        1 |      0 |     1 |                  0 |                      0
 42915 |      3 |        0 |      0 | 43428 |              86856 |                  43428
 42915 |      3 |        1 |      0 |     1 |                  0 |                      0
 42915 |  10000 |        2 |      0 |     4 |                  0 |                      0
(9 rows)
```

STL_SCHEMA_QUOTA_VIOLATIONS

スキーマのクォータを超過したときの発生、タイムスタンプ、XID、およびその他の有用な情報を記録します。

STL_SCHEMA_QUOTA_VIOLATIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_SCHEMA_QUOTA_VIOLATIONS でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
ownerid	integer	スキーマの所有者の ID。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントに関連付けられるプロセス ID。
userid	integer	エントリを生成したユーザーの ID。
schema_id	integer	名前空間またはスキーマの ID。
schema_name	character (128)	名前空間またはスキーマの名前。

列名	データ型	説明
クォータ	integer	スキーマが使用できるディスク容量 (MB)。
disk_usage	integer	スキーマによって現在使用されているディスク容量 (MB)。
disk_usage_pct	double precision	設定されたクォータのうち、スキーマによって現在使用されているディスク容量の割合。
timestamp	タイムゾーンなしのタイムスタンプ	違反が発生した時刻。

サンプルクエリ

次のクエリは、クォータ違反の結果を表示します。

```
SELECT userid, TRIM(SCHEMA_NAME) "schema_name", quota, disk_usage, disk_usage_pct,
 timestamp FROM
stl_schema_quota_violations WHERE SCHEMA_NAME = 'sales_schema' ORDER BY timestamp DESC;
```

このクエリは、指定したスキーマについて次のような出力を返します。

```

userid | schema_name  | quota | disk_usage | disk_usage_pct |timestamp
-------+--------------+-------+------------+----------------
+----------------------------
104    | sales_schema | 2048  | 2798       |  136.62        | 2020-04-20
 20:09:25.494723
(1 row)
```

STL_SESSIONS

ユーザーセッション履歴に関する情報を返します。

STL_SESSIONS と STV_SESSIONS の違いは、STL_SESSIONS にはセッション履歴が含まれ、STV_SESSIONS には現在のアクティブなセッションが含まれているという点です。

STL_SESSIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_SESSION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
starttime	timestamp	UTC で表されたセッションの開始時刻。
endtime	timestamp	UTC で表されたセッションの終了時刻。
process	integer	セッションのプロセス ID。
user_name	character(50)	セッションに関連付けられたユーザー名。
db_name	character(50)	セッションに関連付けられたデータベースの名前。
timeout_sec	int	タイムアウトするまで、セッションが非アクティブまたはアイドル状態を維持する最大秒数。0 の場合は、タイムアウトが設定されていないことを示します。
timed_out	整数	接続が終了した理由を示す値。以下のイベント値があります。 • `0`: 不明なエラーにより接続が終了しました。 • 1: 接続がタイムアウトしました。 • 2: クライアント側で接続が終了しました。 • 3: Amazon Redshift バックエンドの内部エラーにより接続が終了しました。

サンプルクエリ

TICKIT データベースのセッション履歴を表示するには、次のクエリを入力します。

```
select starttime, process, user_name, timeout_sec, timed_out
from stl_sessions
where db_name='tickit' order by starttime;
```

このクエリは、次のサンプル出力を返します。

```
    starttime              | process |  user_name              | timeout_sec | timed_out
---------------------------+---------+-------------------------+-------------
+-------------
2008-09-15 09:54:06.746705 |   32358 | dwuser                  | 120          | 1
2008-09-15 09:56:34.30275  |   32744 | dwuser                  | 60           | 1
2008-09-15 11:20:34.694837 |   14906 | dwuser                  | 0            | 0
2008-09-15 11:22:16.749818 |   15148 | dwuser                  | 0            | 0
2008-09-15 14:32:44.66112  |   14031 | dwuser                  | 0            | 0
2008-09-15 14:56:30.22161  |   18380 | dwuser                  | 0            | 0
2008-09-15 15:28:32.509354 |   24344 | dwuser                  | 0            | 0
2008-09-15 16:01:00.557326 |   30153 | dwuser                  | 120          | 1
2008-09-15 17:28:21.419858 |   12805 | dwuser                  | 0            | 0
2008-09-15 20:58:37.601937 |   14951 | dwuser                  | 60           | 1
2008-09-16 11:12:30.960564 |   27437 | dwuser                  | 60           | 1
2008-09-16 14:11:37.639092 |   23790 | dwuser                  | 3600         | 1
2008-09-16 15:13:46.02195  |    1355 | dwuser                  | 120          | 1
2008-09-16 15:22:36.515106 |    2878 | dwuser                  | 120          | 1
2008-09-16 15:44:39.194579 |    6470 | dwuser                  | 120          | 1
2008-09-16 16:50:27.02138  |   17254 | dwuser                  | 120          | 1
2008-09-17 12:05:02.157208 |    8439 | dwuser                  | 3600         | 0
(17 rows)
```

STL_SORT

ORDER BY 処理などを使用する、ソートを実行するステップのクエリを表示します。

STL_SORT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_SORT には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
バイト	bigint	ステップのすべての出力行のサイズ (バイト単位)。
tbl	integer	テーブル ID。

列名	データ型	説明
is_diskbased	character(1)	true (t) の場合、クエリはディスクベースのオペレーションとして実行されました。false (f) の場合、クエリはメモリ内で実行されました。
workmem	bigint	ステップに割り当てられた作業メモリの総バイト数。
checksum	bigint	この情報は、内部使用に限定されています。

サンプルクエリ

次の例は、スライス 0、セグメント 1 のソート結果を返します。

```
select query, bytes, tbl, is_diskbased, workmem
from stl_sort
where slice=0 and segment=1;
```

```
 query |  bytes   | tbl | is_diskbased |  workmem
-------+---------+-----+--------------+-----------
   567 | 3126968 | 241 | f            | 383385600
   604 |    5292 | 242 | f            | 383385600
   675 |  104776 | 251 | f            | 383385600
   525 | 3126968 | 251 | f            | 383385600
   585 |    5068 | 241 | f            | 383385600
   630 |  204808 | 266 | f            | 383385600
   704 |       0 | 242 | f            |         0
   669 | 4606416 | 241 | f            | 383385600
   696 |  104776 | 241 | f            | 383385600
   651 | 4606416 | 254 | f            | 383385600
   632 |       0 | 256 | f            |         0
   599 |     396 | 241 | f            | 383385600
 86397 |       0 | 242 | f            |         0
   621 |    5292 | 241 | f            | 383385600
 86325 |       0 | 242 | f            |         0
   572 |    5068 | 242 | f            | 383385600
   645 |  204808 | 241 | f            | 383385600
   590 |     396 | 242 | f            | 383385600
(18 rows)
```

STL_SSHCLIENT_ERROR

SSH クライアントに対して表示されるすべてのエラーを記録します。

STL_SSHCLIENT_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
recordtime	timestamp	エラーが記録された時刻。
pid	integer	エラーを記録したプロセス。
ssh_username	character (1024)	SSH ユーザー名。
endpoint	character (1024)	SSH エンドポイント。
コマンド	character (4,096)	完全な SSH コマンド。
error	character (1024)	エラーメッセージ。

STL_STREAM_SEGS

ストリームと並行セグメントの間の関係をリスト表示します。

このコンテキストのストリームは Amazon Redshift ストリームです。このシステムビューは マテリアライズドビューへのストリーミング取り込み には適用されません。

STL_STREAM_SEGS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

Note

STL_STREAM_SEGS には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
stream	integer	クエリの並行セグメントのセット。
segment	integer	クエリセグメントを識別する番号。

サンプルクエリ

最後に実行したクエリのストリームと並行セグメントの間の関係を表示するには、次のクエリを入力します。

```
select *
from stl_stream_segs
where query = pg_last_query_id();

 query | stream | segment
```

```
-------+--------+---------
    10 |       1 |        2
    10 |       0 |        0
    10 |       2 |        4
    10 |       1 |        3
    10 |       0 |        1
(5 rows)
```

STL_TR_CONFLICT

データベーステーブルのトランザクション競合を認識し、解決するための情報を表示します。

2 人以上のユーザーがクエリとデータ行の変更を実行し、これらのトランザクションを直列化できない場合、トランザクションの競合が発生します。直列化可能性を損なうステートメントを実行するトランザクションは停止され、ロールバックされます。Amazon Redshift は、トランザクションの競合が発生するたびに、キャンセルされたトランザクションの詳細が記載されている STL_TR_CONFLICT システムテーブルにデータ行を書き込みます。詳細については、「直列化可能分離」を参照してください。

STL_TR_CONFLICT はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_TRANSACTION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
xact_id	bigint	ロールバックされたトランザクションのトランザクション ID。
process_id	bigint	ロールバックされたトランザクションに関連付けられているプロセス。
xact_start_ts	timestamp	トランザクションが開始された時刻のタイムスタンプ (UTC)。

列名	データ型	説明
abort_time	timestamp	トランザクションが停止された時刻のタイムスタンプ (UTC)。
table_id	bigint	競合が発生したテーブルのテーブル ID。

サンプルクエリ

特定のテーブルの競合に関する情報を返すには、テーブル ID を指定したクエリを実行します。

```
select * from stl_tr_conflict where table_id=100234
order by xact_start_ts;

xact_id|process_|      xact_start_ts        |        abort_time         |table_
       |id      |                           |                           |id
-------+--------+---------------------------+---------------------------+------
  1876 |  8551  |2010-03-30 09:19:15.852326|2010-03-30 09:20:17.582499|100234
  1928 | 15034  |2010-03-30 13:20:00.636045|2010-03-30 13:20:47.766817|100234
  1991 | 23753  |2010-04-01 13:05:01.220059|2010-04-01 13:06:06.94098 |100234
  2002 | 23679  |2010-04-01 13:17:05.173473|2010-04-01 13:18:27.898655|100234
(4 rows)
```

テーブル ID は、連続性違反 (エラー 1023) のエラーメッセージの DETAIL セクションから取得できます。

STL_UNDONE

元に戻されたトランザクションに関する情報を表示します。

STL_UNDONE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_TRANSACTION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xact_id	bigint	元に戻すトランザクションの ID。
xact_id_undone	bigint	元に戻されたトランザクションの ID。
undo_start_ts	timestamp	元に戻すトランザクションの開始時刻。
undo_end_ts	timestamp	元に戻すトランザクションの終了時刻。
table_id	bigint	元に戻すトランザクションの影響を受けるテーブルの ID。

サンプルクエリ

元に戻されたすべてのトランザクションの簡潔なログを表示するには、次のコマンドを入力します。

```
select xact_id, xact_id_undone, table_id from stl_undone;
```

このコマンドは、次のサンプル出力を返します。

```
 xact_id | xact_id_undone | table_id
---------+----------------+----------
1344 |           1344 |   100192
1326 |           1326 |   100192
1551 |           1551 |   100192
(3 rows)
```

STL_UNIQUE

SELECT リストで DISTINCT 関数が使用されているとき、あるいは UNION または INTERSECT クエリから重複が除去されるときに発生する実行ステップを分析します。

STL_UNIQUE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

Note

STL_UNIQUE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: `2009-06-12 11:29:19.131358` 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。

列名	データ型	説明
`type`	character(6)	ステップの種類。有効な値は次のとおりです。 • HASHED。ステップは、グループ化され、ソートされていない集計を使用しました。 • PLAIN。ステップは、グループ化されていない、スカラーの集計を使用しました。 • SORTED。ステップは、グループ化され、ソートされた集計を使用しました。
is_diskbased	character(1)	true (t) の場合、クエリはディスクベースのオペレーションとして実行されました。false (f) の場合、クエリはメモリ内で実行されました。
slots	integer	ハッシュバケットの総数。
workmem	bigint	ステップに割り当てられた作業メモリの総バイト数。
max_buffers_used	bigint	ディスクに書き込まれる前にハッシュテーブル内で使用されるバッファの最大数。
サイズ変更	integer	この情報は、内部使用に限定されています。
occupied	integer	この情報は、内部使用に限定されています。
フラッシュ可能	integer	この情報は、内部使用に限定されています。
used_unique_prefetching	character(1)	この情報は、内部使用に限定されています。
bytes	biginit	ステップの総出力行のバイト数。

サンプルクエリ

以下のクエリを実行するとします。

```
select distinct eventname
from event order by 1;
```

次の例は、上のクエリの ID が 6313 であるという前提で、セグメント 0 および 1 内の各スライスにおけるそれぞれのステップで作成される行の数を表します。

```
select query, slice, segment, step, datediff(msec, starttime, endtime) as msec,
 tasknum, rows
from stl_unique where query = 6313
order by query desc, slice, segment, step;
```

```
 query | slice | segment | step | msec | tasknum | rows
-------+-------+---------+------+------+---------+------
  6313 |     0 |       0 |    2 |    0 |      22 |  550
  6313 |     0 |       1 |    1 |  256 |      20 |  145
  6313 |     1 |       0 |    2 |    1 |      23 |  540
  6313 |     1 |       1 |    1 |   42 |      21 |  127
  6313 |     2 |       0 |    2 |    1 |      22 |  540
  6313 |     2 |       1 |    1 |  255 |      20 |  158
  6313 |     3 |       0 |    2 |    1 |      23 |  542
  6313 |     3 |       1 |    1 |   38 |      21 |  146
(8 rows)
```

STL_UNLOAD_LOG

はアンロード処理の詳細を記録します。

STL_UNLOAD_LOG は、UNLOAD ステートメントによって作成される各ファイルについて 1 行ずつレコードが作成されます。例えば、UNLOAD で 12 個のファイルが作成されると、STL_UNLOAD_LOG にはそれに対応する 12 行が作成されます。

STL_UNLOAD_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_UNLOAD_LOG には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS

モニタリングビュー SYS_UNLOAD_HISTORY と SYS_UNLOAD_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。
slice	integer	クエリが実行されていたスライスを識別する番号。
pid	integer	クエリステートメントに関連付けられるプロセス ID。
パス	character(1280)	Amazon S3 オブジェクトのファイルへの完全パス。
start_time	timestamp	トランザクションの開始時間。
end_time	timestamp	トランザクションの終了時間。
line_count	bigint	ファイルにアンロードされた行数。
transfer_size	bigint	転送バイト数。
file_format	character(10)	アンロードされたファイルの形式。

サンプルクエリ

UNLOAD コマンドによって Amazon S3 に書き込まれたファイルのリストを取得するには、UNLOAD が完了した後に Amazon S3 リストのオペレーションを呼び出すことができます。STL_UNLOAD_LOG をクエリすることもできます。

以下のクエリでは、最後に完了されたクエリの UNLOAD によって作成されたファイルのパス名が返されます。

```
select query, substring(path,0,40) as path
```

```
from stl_unload_log
where query = pg_last_query_id()
order by path;
```

このコマンドは、次のサンプル出力を返します。

```
 query |             path
-------+--------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

STL_USAGE_CONTROL

STL_USAGE_CONTROL ビューには、使用制限に達したときに記録される情報が含まれています。使用制限の詳細については、「Amazon Redshift 管理ガイド」の「使用制限の管理」を参照してください。

STL_USAGE_CONTROL はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
eventtime	timestamp	クエリが使用制限を超えた時刻 (UTC)。
query	integer	クエリ識別子。この ID を使用して、他のさまざまなシステムテーブルやビューを結合できます。
xid	bigint	トランザクション識別子。
pid	integer	クエリに関連付けられたプロセス識別子。
usage_limit_id	character(40)	Amazon Redshift (例えば 25d9297e-3e7b-41c8-9f4d-c4b6eb731c09) によって生成される汎用一意識別子 (UUID)。

列名	データ型	説明
feature_type	character(30)	使用制限を超えた機能。指定できる値には、CONCURRENCY_SCALING　および SPECTRUM などがあります。

サンプルクエリ

次の SQL 例は、使用制限に達したときに記録される情報の一部を返します。

```
select query, pid, eventtime, feature_type
from stl_usage_control
order by eventtime desc
limit 5;
```

STL_USERLOG

データベースユーザーに対する次の変更の詳細を記録します。

- ユーザーの作成
- ユーザーの削除
- ユーザーの変更 (名前の変更)
- ユーザーの変更 (プロパティの変更)

STL_USERLOG はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_USERLOG でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	変更の影響を受けるユーザーの ID。
username	character(50)	変更の影響を受けるユーザーのユーザー名。

列名	データ型	説明
oldusername	character(50)	名前の変更アクションの場合、以前のユーザー名。その他のアクションの場合、このフィールドは空欄です。
action	character(10)	実行されたアクション。有効な値: • 変更 • 作成 • 削除 • 名前の変更
usecreatedb	integer	true (1) の場合、ユーザーに create database 権限があることを示します。
usesuper	integer	true (1) の場合、ユーザーがスーパーユーザーであることを示します。
usecatupd	integer	true (1) の場合、ユーザーはシステムカタログを更新できることを示します。
valuntil	timestamp	パスワードが失効する日付。
pid	integer	プロセス ID。
xid	bigint	トランザクション ID。
recordtime	timestamp	UTC で表されたクエリの開始時間。

サンプルクエリ

次の例は、4 つのユーザーアクションを実行してから、STL_USERLOG ビューをクエリします。

```
create user userlog1 password 'Userlog1';
alter user userlog1 createdb createuser;
alter user userlog1 rename  to userlog2;
drop user userlog2;
```

```
select userid, username, oldusername, action, usecreatedb, usesuper from stl_userlog
 order by recordtime desc;
```

```
 userid |  username | oldusername |  action | usecreatedb | usesuper
--------+-----------+-------------+---------+-------------+----------
    108 | userlog2  |             | drop    |           1 |   1
    108 | userlog2  | userlog1    | rename  |           1 |   1
    108 | userlog1  |             | alter   |           1 |   1
    108 | userlog1  |             | create  |           0 |   0
 (4 rows)
```

STL_UTILITYTEXT

データベースに対して実行された SELECT 以外の SQL コマンドを取得します。

STL_UTILITYTEXT ビューをクエリすると、システムで実行された SQL ステートメントのうち、次のサブセットを取得できます。

- ABORT、BEGIN、COMMIT、END、ROLLBACK
- ANALYZE
- CALL
- CANCEL
- COMMENT
- CREATE、ALTER、DROP DATABASE
- CREATE、ALTER、DROP USER
- EXPLAIN
- GRANT、REVOKE
- LOCK
- RESET
- SET
- SHOW
- TRUNCATE

STL_DDLTEXT、STL_QUERYTEXT、SVL_STATEMENTTEXT も参照してください。

STARTTIME および ENDTIME 列を使用すると、一定の時間内に記録されたステートメントがわかります。SQL テキストの長いブロックは、200 文字の長さに分割されます。SEQUENCE 列により、1 つのステートメントに属する複数のフラグメントのテキストを識別できます。

STL_UTILITYTEXT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	トランザクション ID。
pid	integer	クエリステートメントに関連付けられるプロセス ID。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドは空になります。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **`2009-06-12 11:29:19.131358`** 。

列名	データ型	説明
sequence	integer	1 つのステートメントに含まれる文字数が 200 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
text	character(200)	200 文字単位の SQL テキスト。このフィールドには、バックスラッシュ (\\) や改行 (\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

次のクエリは、2012 年 1 月 26 日に実行された「utility」コマンドのテキストを返します。ここでは、いくつかの SET コマンドと 1 つの SHOW ALL コマンドが実行されています。

```
select starttime, sequence, rtrim(text)
from stl_utilitytext
where starttime like '2012-01-26%'
order by starttime, sequence;

starttime           | sequence |              rtrim
---------------------------+-----+----------------------------------
2012-01-26 13:05:52.529235 |   0 | show all;
2012-01-26 13:20:31.660255 |   0 | SET query_group to ''
2012-01-26 13:20:54.956131 |   0 | SET query_group to 'soldunsold.sql'
...
```

ストアド SQL の再構築

STL_UTILITYTEXT の `text` 列に保存されている SQL を再構築するには、SELECT ステートメントを実行して、`text`列の 1 つ以上の部分から SQL を作成します。再構築された SQL を実行する前に、特殊文字 (\n) がある場合は、改行に置き換えます。次の SELECT ステートメントの結果は、`query_statement`フィールドに再構築された SQL の行です。

```
SELECT LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END) WITHIN
 GROUP (ORDER BY sequence) as query_statement
FROM stl_utilitytext GROUP BY xid order by xid;
```

例えば、次のクエリでは、query_group をゼロの文字列に設定します。クエリ自体は 200 文字より長く、STL_UTILITYTEXT の部分に保存されます。

```
set query_group to
 '00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
             000000';
```

この例では、クエリは STL_UTILITYTEXT の `text` 列の 2 つの部分 (行) に保存されます。

```
select query, sequence, text
from stl_utilitytext where query=pg_last_query_id() order by query desc, sequence limit
 10;
```

```


         starttime          | sequence |
                                                    text

----------------------------+----------
+-----------------------------------------------------------------------------------------------
 2019-07-23 22:55:34.926198 |        0 | set query_group to
 '00000000000000000000000000000000000000000000000000000000000000000000000000000000\n00000000000
 2019-07-23 22:55:34.926198 |        1 |                    000000';
```

STL_UTILITYTEXT に保存された SQL を再構築するには、次の SQL を実行します。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from stl_utilitytext where query=pg_last_query_id();
```

再構築された SQL をクライアントで使用するには、特殊文字 (\n) を改行に置き換えます。

```


                                               query_statement


-----------------------------------------------------------------------------------------------------
```

```
set query_group to
'000000000000000000000000000000000000000000000000000000000000000000000000000000000000\n00000000000
                  000000';
```

STL_VACUUM

バキュームされたテーブルの行およびブロックに関する統計を表示します。

このビューは、各バキュームオペレーションが開始および終了したときの固有の情報を保持し、バキュームオペレーションを実行する利点を示します。このコマンドを実行するための要件については、VACUUM コマンドの説明を参照してください。

STL_VACUUM はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_VACUUM_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	VACUUM ステートメントのトランザクション ID。このテーブルを STL_QUERY ビューに結合することにより、特定の VACUUM トランザクションで実行された個々の SQL ステートメントを確認できます。データベース全体をバキュームする場合、各テーブルが別々のトランザクションでバキュームされます。
table_id	integer	テーブル ID。
status	character(30)	各テーブルの VACUUM 操作のステータス。取り得る値には以下のものがあります。 • **`Started`** • **`Started Delete Only`**

列名	データ型	説明
		• **Started Delete Only (Sorted >= nn%)** VACUUM FULL の削除フェーズのみを開始しました。テーブルはすでにソートされているかソートしきい値を超えているため、ソートフェーズはスキップされました。 • **Started Sort Only** • **Started Ranged Partition** • **Started Reindex** • **Finished** テーブルのオペレーションが完了した時刻。特定のテーブルに対してバキューム操作にかかった時間を計算するために、特定のトランザクション ID およびテーブル ID について終了時刻から開始時刻を引きます。 • **Skipped** テーブルが完全にソートされ削除対象としてマークされた列はないため、テーブルはスキップされました。 • **Skipped (delete only)** DELETE ONLY が指定され、削除対象としてマークされた列はないため、テーブルはスキップされました。 • **Skipped (sort only)** SORT ONLY が指定され、テーブルはすでに完全にソートされたため、テーブルはスキップされました。 • **Skipped (sort only, sorted>=xx%)** SORT ONLY が指定され、テーブルはすでにソートされているかソートしきい値を超えているため、テーブルはスキップされました。 • **Skipped (0 rows)** テーブルは空であるため省略されました。

列名	データ型	説明
		• **VacuumBG** 自動バキューム操作がバックグラウンドで実行されます。このステータスは、自動的に実行されるときに他のステータスの前に追加されます。例えば、削除のみのバキュームを自動的に実行すると、開始行に `[VacuumBG] Started Delete Only` ステータスが表示されます。 VACUUM ソートしきい値を設定する方法の詳細については、VACUUMを参照してください。
rows	bigint	テーブル内にある実際の行と、削除されてまだディスクに保存されている (バキューム待ちの) 行の数の合計。この列は、バキューム開始前にステータスが **Started** だった行の数と、バキューム後にステータスが **Finished** だった行の数を示します。
sortedrows	integer	テーブル内でソートされる行の数。この列は、バキューム開始前に Status 列が **Started** だった行の数と、バキューム後に Status 列が **Finished** だった行の数を示します。
blocks	integer	バキューム操作前 (ステータスが **Started** の行) とバキューム操作後 (**Finished** の列) のそれぞれに、テーブルデータを保存するために使用されたデータブロックの総数。各データブロックのサイズは 1 MB です。

列名	データ型	説明
max_merge_partitions	integer	この列は、パフォーマンスの分析に使用され、バキュームが 1 回のマージフェーズで処理できるテーブルのパーティションの最大数を表します (バキュームでは、ソートされていないリージョンを 1 つ以上のソートされたパーティションにソートます。テーブルの列数と現在の Amazon Redshift の設定に応じて、マージフェーズでは繰り返しマージ処理のうち 1 回で最大数のパーティションを処理できます。マージフェーズは、ソートされたパーティションの数がマージパーティションの最大数を超えても引き続き有効ですが、繰り返しマージ処理をさらに続ける必要があります。)
eventtime	timestamp	バキューム操作が開始または終了した時刻。
reclaimable_rows	bigint	現在の cutoff_xid で再利用可能な行の数。この列は、ステータスが **Started** の行に対してバキュームを開始する前の Redshift が推定する再利用可能な行の数と、ステータスが **Finished** のバキューム後に残っている再利用可能な行の実数を示します。
reclaimable_space_mb	bigint	現在の cutoff_xid 用に再利用可能なスペース (MB 単位)。この列は、ステータスが **Started** の行に対してバキュームを開始する前の Redshift が推定する再利用可能な空き容量と、ステータスが **Finished** のバキューム後に残っている再利用可能な空き容量の実数を示します。
cutoff_xid	bigint	バキューム処理の cutoff トランザクション ID。cutoff 後のトランザクションはバキューム処理に含まれません。
is_recluster	integer	1 (true) の場合、バキューム処理で再クラスターアルゴリズムが実行され、0 (false) の場合は実行されませんでした。

サンプルクエリ

次のクエリは、テーブル 108313 のバキューム統計をレポートします。テーブルは一連の挿入、削除後にバキューム処理されています。

```
select xid, table_id, status, rows, sortedrows, blocks, eventtime,
                reclaimable_rows, reclaimable_space_mb
from stl_vacuum where table_id=108313 order by eventtime;

 xid   | table_id | status                  | rows | sortedrows | blocks |  eventtime
        | reclaimable_rows | reclaimable_space_mb
-------+----------+-------------------------+------+------------+--------
+---------------------+-------------------+----------------------
 14294 |   108313 | Started                 | 1950 |        408 |     28 |  2016-05-19
 17:36:01 |              984 |                     17
 14294 |   108313 | Finished                |  966 |        966 |     11 |  2016-05-19
 18:26:13 |                0 |                      0
 15126 |   108313 | Skipped(sorted>=95%)    |  966 |        966 |     11 |  2016-05-19
 18:26:38 |                0 |                      0
```

バキュームの開始時に、テーブルには 1,950 行が 28 個の 1 MB ブロックに含まれていました。Amazon Redshift は、バキューム処理で 984 行、17 ブロックのディスク容量を回収できると見積もっています。

ステータスが Finished の行では、ROWS 列は 966 の値になり、BLOCKS 列では値が 28 から 11 に減っています。バキューム処理が完了すると、推定量のディスク容量が回収され、再利用可能な行や空き容量がなくなりました。

ソートフェーズ (トランザクション 15126) で、行がソートキー順序で挿入されバキュームはテーブルをスキップできました。

次の例では、大きな INSERT 操作の後に SALES テーブル (この例ではテーブル 110116) の SORT ONLY のバキュームの統計が表示されます。

```
vacuum sort only sales;

select xid, table_id, status, rows, sortedrows, blocks, eventtime
from stl_vacuum order by xid, table_id, eventtime;

xid |table_id|     status      | rows  |sortedrows|blocks|     eventtime
----+--------+-----------------+-------+----------+------+--------------------
```

```
...
2925| 110116 |Started Sort Only|1379648|   172456 |   132 | 2011-02-24 16:25:21...
2925| 110116 |Finished         |1379648|  1379648 |   132 | 2011-02-24 16:26:28...
```

STL_WINDOW

Window 関数を実行するクエリステップを分析します。

STL_WINDOW はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STL_WINDOW には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	クエリが実行されていたスライスを識別する番号。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
starttime	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。

列名	データ型	説明
endtime	timestamp	クエリの実行が完了した時刻 (UTC)。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。
tasknum	integer	ステップ実行に割り当てられたクエリタスク処理の数。
rows	bigint	処理された合計行数。
is_diskbased	character(1)	true (t) の場合、クエリはディスクベースのオペレーションとして実行されました。false (f) の場合、クエリはメモリ内で実行されました。
workmem	bigint	ステップに割り当てられた作業メモリの総バイト数。

サンプルクエリ

次の例は、スライス 0、セグメント 3 のウィンドウ関数の結果を返します。

```
select query, tasknum, rows, is_diskbased, workmem
from stl_window
where slice=0 and segment=3;
```

```
 query | tasknum | rows | is_diskbased | workmem
-------+---------+------+--------------+----------
 86326 |      36 | 1857 | f            | 95256616
   705 |      15 | 1857 | f            | 95256616
 86399 |      27 | 1857 | f            | 95256616
   649 |      10 |    0 | f            | 95256616
(4 rows)
```

STL_WLM_ERROR

すべての WLM 関連のエラーを発生時に記録します。

STL_WLM_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
recordtime	timestamp	エラーが発生した時刻。
pid	integer	エラーを起こしたプロセスの ID。
error_string	character(256)	エラーの説明。

STL_WLM_RULE_ACTION

レコードは、ユーザー定義のキューに関連付けられた WLM クエリモニタリングルールによって生じたアクションの詳細を示します。詳細については、「WLM クエリモニタリングルール」を参照してください。

STL_WLM_RULE_ACTION はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	クエリを実行したユーザー。
query	integer	クエリ ID。
service_class	integer	サービスクラスの ID。クエリキューは WLM 設定で定義されます。サービスクラス 5 以上はユーザー定義のキューです。
ルール	character(256)	クエリモニタリングのルールの名前。
action	character(256)	結果として生じるアクション。指定できる値は次のとおりです。 • ログ

列名	データ型	説明
		• hop(reassign) • hop(restart) • abort • change_query_priority • なし none 値は、ルールの述語が満たされたものの、アクションの重大度が高い別のルールが優先されたことを示します。
recordtime	timestamp	アクションが記録された時刻 (UTC)。
action_value	character(256)	action が change_query_priority の場合、指定できる値は highest、high、normal、low、lowest です。 action が log、hop、または abort の場合、値は空です。
service_class_name	character(64)	サービスクラスの名前。

サンプルクエリ

以下の例は、クエリモニタリングルールによって停止されたクエリを検索します。

```
Select query, rule
from stl_wlm_rule_action
where action = 'abort'
order by query;
```

STL_WLM_QUERY

WLM が扱うサービスクラス内で実行が試みられたクエリごとに、レコードが 1 件作成されます。

STL_WLM_QUERY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	integer	クエリまたはサブクエリのトランザクション ID。
task	integer	ワークロードマネージャ全体を通じてクエリを追跡するために使用される ID。複数のクエリ ID と関連付けることができます。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
query	integer	クエリ ID。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
service_class	integer	サービスクラスの ID。サービスクラスの ID のリストについては、「WLM サービスクラス ID」を参照してください。
slot_count	integer	キューに設定された同時実行レベルに従ってクエリが使用する WLM クエリスロットの数。デフォルトは1です。詳細については、「wlm_query_slot_count」を参照してください。
service_class_start_time	timestamp	クエリがサービスクラスに割り当てられた時刻。この時刻は UTC タイムゾーンです。

列名	データ型	説明
queue_start_time	timestamp	クエリがサービスクラスのキューに入れられた時刻。この時刻は UTC タイムゾーンです。
queue_end_time	timestamp	クエリがサービスクラスのキューから解放された時刻。この時刻は UTC タイムゾーンです。
total_queue_time	bigint	クエリがキュー内で消費した時間の合計 (マイクロ秒単位)
exec_start_time	timestamp	サービスクラス内でクエリの実行が始まった時刻。この時刻は UTC タイムゾーンです。
exec_end_time	timestamp	サービスクラス内でクエリの実行が完了した時刻。この時刻は UTC タイムゾーンです。
total_exec_time	bigint	クエリが実行にかかった時間 (マイクロ秒単位)。
service_class_end_time	timestamp	クエリがサービスクラスから解放された時刻。この時刻は UTC タイムゾーンです。
final_state	character(16)	将来の利用のために予約されています。
est_peak_mem	bigint	将来の利用のために予約されています。
query_priority	char(20)	クエリの優先度です。有効な値は、`n/a`、`lowest`、`low`、`normal`、`high`、および `highest` です。ここで、`n/a` は、クエリ優先度がサポートされていないことを意味します。
service_class_name	character(64)	サービクラス名。サービスクラスの詳細については、「WLM システムテーブルとビュー」を参照してください。

サンプルクエリ

クエリのキューと実行にかかったクエリの平均時間の表示

次のクエリでは、4 を超えるサービスクラスの現在の設定を表示します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

次のクエリは、各クエリがクエリキュー内に置かれていた時間および実行にかかった時間の平均を、サービスクラス別にマイクロ秒単位で返します。

```
select service_class as svc_class, count(*),
avg(datediff(microseconds, queue_start_time, queue_end_time)) as avg_queue_time,
avg(datediff(microseconds, exec_start_time, exec_end_time )) as avg_exec_time
from stl_wlm_query
where service_class > 4
group by service_class
order by service_class;
```

このクエリは、次のサンプル出力を返します。

```
 svc_class | count | avg_queue_time | avg_exec_time
-----------+-------+----------------+---------------
         5 | 20103 |              0 |         80415
         5 |  3421 |          34015 |        234015
         6 |    42 |              0 |        944266
         7 |   196 |           6439 |       1364399
(4 rows)
```

クエリのキューと実行にかかったクエリの最大時間の表示

次のクエリは、クエリがいずれかのクエリキュー内に置かれていた時間と実行にかかった時間の最大値を、サービスクラスごとにマイクロ秒単位で返します。

```
select service_class as svc_class, count(*),
max(datediff(microseconds, queue_start_time, queue_end_time)) as max_queue_time,
max(datediff(microseconds, exec_start_time, exec_end_time )) as max_exec_time
from stl_wlm_query
where svc_class > 5
group by service_class
order by service_class;
```

```
 svc_class | count | max_queue_time | max_exec_time
-----------+-------+----------------+---------------
         6 |    42 |              0 |       3775896
         7 |   197 |          37947 |      16379473
```

```
(4 rows)
```

スナップショットデータの STV テーブル

STV テーブルとは、現在のシステムデータのスナップショットを含む仮想システムテーブルです

トピック

- STV_WLM_QUERY_STATE
- STV_WLM_QUERY_TASK_STATE
- STV_WLM_SERVICE_CLASS_CONFIG
- STV_WLM_SERVICE_CLASS_STATE
- STV_XRESTORE_ALTER_QUEUE_STATE

STV_ACTIVE_CURSORS

STV_ACTIVE_CURSORS は、現在開いているカーソルの詳細を表示します。詳細については、「DECLARE」を参照してください。

STV_ACTIVE_CURSORS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。ユーザーは、自身が開いたカーソルのみを表示できます。スーパーユーザーはすべてのカーソルを表示できます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
name	characte (256)	カーソル名。
xid	bigint	トランザクションコンテキスト。
pid	integer	クエリを実行しているリーダープロセス。
starttime	timestam	カーソルが宣言された時刻。
row_count	bigint	カーソル結果セット内の行数。
byte_count	bigint	カーソル結果セット内のバイト数。
fetched_rows	bigint	カーソル結果セットから現在取得されている行数。

STV_BLOCKLIST

STV_BLOCKLIST には、データベース内の各スライス、テーブル、または列で使用される 1 MB ディスクブロックの数が表示されます。

データベース、テーブル、または列ごとに割り当てられている 1 MB ディスクブロックの数を調べるには、以下の例に示すように、STV_BLOCKLIST で集計クエリを使用します。または STV_PARTITIONS を使用して、ディスク利用に関する概要を見ることができます。

STV_BLOCKLIST はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> STV_BLOCKLIST は、プロビジョニングされたクラスターまたはサーバーレス名前空間が所有するブロックのみを記録します。データベースにデータ共有プロデューサーから共有されたブロックが含まれている場合、それらのブロックは STV_BLOCKLIST に含まれません。データ共有に関する詳細については、「Amazon Redshift でのデータの共有」を参照してください。

テーブルの列

列名	データ型	説明
slice	integer	ノードスライス。
col	integer	列のゼロベースインデックス。作成するテーブルにはすべて、3 つの非表示列 (INSERT_XID、DELETE_XID、ROW_ID (OID)) が追加されます。例えばユーザー定義列が 3 つあるテーブルには、実際には 6 つの列が含まれます。ユーザー定義列の初期番号は 0、1、2 です。その場合、INSERT_XID、DELETE_XID、および ROW_ID 列はそれぞれ、3、4、および 5 となります。
tbl	integer	データベーステーブルのテーブル ID。
blocknum	integer	データブロックの ID。

列名	データ型	説明
num_values	integer	ブロックに含まれる値の数。
extended_limits	integer	内部使用を目的とします。
minvalue	bigint	ブロックの最小値。数値以外のデータの場合、最初の 8 文字を 64 ビット整数として格納します。ディスクスキャンに使用されます。
maxvalue	bigint	ブロックの最大値。数値以外のデータの場合、最初の 8 文字を 64 ビット整数として格納します。ディスクスキャンに使用されます。
sb_pos	integer	ディスクのスーパーブロックの位置の内部 Amazon Redshift 識別子。
pinned	integer	事前ロードの一環としてブロックをメモリにピンするかどうかを示します。0 = false、1 = true。デフォルトは false です。
on_disk	integer	ブロックを自動的にディスクに保存するかどうかを示します。0 = false、1 = true。デフォルトは false です。
modified	integer	ブロックが修正されたかどうかを示します。0 = false、1 = true。デフォルトは false です。
hdr_modified	integer	ブロックヘッダーが修正されたかどうかを示します。0 = false、1 = true。デフォルトは false です。
unsorted	integer	ブロックが未ソートかどうかを示します。0 = false、1 = true。デフォルトは true です。
tombstone	integer	内部使用を目的とします。
preferred_diskno	integer	ディスクにエラーがない場合に、ブロックが存在すべきディスクの番号。ディスクが修正されると、ブロックがそのディスクに戻されます。
temporary	integer	ブロックに、一時テーブルやクエリの中間結果などの一時データが含まれているかどうかを示します。0 = false、1 = true。デフォルトは false です。

列名	データ型	説明
newblock	integer	ブロックが新しい (true) かまたは一度もディスクにコミットされていない (false) かを示します。0 = false、1 = true。
num_readers	integer	各ブロックの参照数。
flags	integer	ブロックヘッダーの内部 Amazon Redshift フラグ。

サンプルクエリ

STV_BLOCKLIST には、割り当てられたディスクブロックごとに行が 1 つ含まれるため、すべての行を選択するようなクエリを実行すると、非常に多くの行が返される可能性があります。STV_BLOCKLIST では集計クエリのみを使用することをお勧めします。

SVV_DISKUSAGE ビューには同様の情報が見やすい形式で表示されますが、以下の例では STV_BLOCKLIST テーブルの利用方法の 1 つを示します。

VENUE テーブル内の各列で使用されている 1 MB ブロックの数を調べるには、以下のクエリを使用します。

```
select col, count(*)
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;
```

このクエリは、VENUE テーブル内の各列に割り当てられている 1 MB ブロックの数を返します。例:

```
 col | count
-----+-------
   0 |   4
   1 |   4
   2 |   4
   3 |   4
   4 |   4
```

```
   5 |  4
   7 |  4
   8 |  4
(8 rows)
```

以下のクエリは、テーブルデータが実際に全スライスに配分されているかどうかを示します。

```
select trim(name) as table, stv_blocklist.slice, stv_tbl_perm.rows
from stv_blocklist,stv_tbl_perm
where stv_blocklist.tbl=stv_tbl_perm.id
and stv_tbl_perm.slice=stv_blocklist.slice
and stv_blocklist.id > 10000 and name not like '%#m%'
and name not like 'systable%'
group by name, stv_blocklist.slice, stv_tbl_perm.rows
order by 3 desc;
```

このクエリは、以下のような出力を返します。出力には、最も多くの行を含む表に対する均等なデータ配分が表示されています。

```


table   | slice | rows
----------+-------+-------
listing  |    13 | 10527
listing  |    14 | 10526
listing  |     8 | 10526
listing  |     9 | 10526
listing  |     7 | 10525
listing  |     4 | 10525
listing  |    17 | 10525
listing  |    11 | 10525
listing  |     5 | 10525
listing  |    18 | 10525
listing  |    12 | 10525
listing  |     3 | 10525
listing  |    10 | 10525
listing  |     2 | 10524
listing  |    15 | 10524
listing  |    16 | 10524
listing  |     6 | 10524
listing  |    19 | 10524
listing  |     1 | 10523
listing  |     0 | 10521
...
```

```
(180 rows)
```

以下のクエリは、廃棄されたブロックの中に、ディスクにコミットされていたものがあるかどうかを返します。

```
select slice, col, tbl, blocknum, newblock
from stv_blocklist
where  tombstone > 0;

slice | col |   tbl  | blocknum | newblock
-------+-----+--------+----------+----------
4      |  0  | 101285 |    0     |   1
4      |  2  | 101285 |    0     |   1
4      |  4  | 101285 |    1     |   1
5      |  2  | 101285 |    0     |   1
5      |  0  | 101285 |    0     |   1
5      |  1  | 101285 |    0     |   1
5      |  4  | 101285 |    1     |   1
...
(24 rows)
```

STV_CURSOR_CONFIGURATION

STV_CURSOR_CONFIGURATION はカーソル設定の制約を表示します。詳細については、「カーソルの制約」を参照してください。

STV_CURSOR_CONFIGURATION はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
current_cursor_count	integer	現在開いているカーソルの数。
max_diskspace_usable	integer	カーソルに利用できるディスク領域の量 (メガバイト)。この制限は、クラスターのカーソル結果セットの最大サイズに基づいています。

列名	データ型	説明
current_diskspace_used	integer	現在カーソルで使用中のディスク領域の量 (メガバイト)。

STV_DB_ISOLATION_LEVEL

STV_DB_ISOLATION_LEVEL は、データベースの現在の分離レベルを表示します。分離レベルの詳細については、「CREATE DATABASE」を参照してください。

STV_DB_ISOLATION_LEVEL は、すべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
db_name	character(128)	データベース名。
isolation_level	character(20)	データベースの分離レベル。指定できる値には、`Serializable` および `Snapshot Isolation` などがあります。

STV_EXEC_STATE

コンピュートノードでアクティブに実行されているクエリおよびクエリ手順についての情報を取得するには、STV_EXEC_STATE テーブルを使用します。

この情報は通常、エンジニアリング上の問題を解決する際にのみ使用されます。SVV_QUERY_STATE ビューおよび SVL_QUERY_SUMMARY ビューは、STV_EXEC_STATE から情報を取得します。

STV_EXEC_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	ステップが完了した場合のノードスライスです。
segment	integer	実行したクエリのセグメントです。クエリセグメントとは、一連の手順です。
step	integer	完了したクエリセグメントのステップです。ステップとは、クエリにより実行される最小の単位です。
starttime	timestamp	ステップが実行された時刻。
currenttime	timestamp	現在の時刻。
tasknum	integer	ステップを完了するために割り当てられたクエリタスク処理です。
rows	bigint	処理された行数。
バイト	bigint	処理されたバイト数。
label	char(256)	クエリステップ名および該当する場合はテーブル ID とテーブル名 (`scan tbl=100448 name =user` など) からなるステップラベル。3 桁のテーブル ID は通常、一時

列名	データ型	説明
		テーブルのスキャンを照会します。`tbl=0` は、通常、定数値のスキャンを指します。
is_diskbased	char(1)	クエリのこのステップがディスクベースの処理として完了されたかどうかを示します: true (**t**) または false (**f**)。ハッシュ、ソート、集計といった特定のステップのみがディスクベースで実行できます。ステップの多くのタイプは、常にメモリ内で完了されます。
workmem	bigint	ステップに割り当てられた作業メモリのバイト数。
num_parts	integer	ハッシュテーブルがハッシュステップ中に分割されるパーティションの数。この列にある正の数は、ハッシュステップがディスクベースのオペレーションとして実行されることを示すものではありません。ハッシュステップがディスクベースであったかどうかを調べるには、IS_DISKBASED 列の値を見てください。
is_rrscan	char(1)	true (**t**) の場合は、ステップで範囲限定スキャンが使用されたことを示します。デフォルトは false (**f**) です。
is_delayed_scan	char(1)	true (**t**) の場合は、ステップで遅延スキャンが使用されたことを示します。デフォルトは false (**f**) です。

サンプルクエリ

Amazon Redshift では、STV_EXEC_STATE を直接クエリするのでなく、SVL_QUERY_SUMMARY または SVV_QUERY_STATE をクエリすることをお勧めしています。これにより、STV_EXEC_STATE の情報を見やすい形式で取得することができます。詳細については、SVL_QUERY_SUMMARY または SVV_QUERY_STATE テーブルのドキュメントを参照してください。

STV_INFLIGHT

クラスターで現在どのクエリが実行されているかを調べるには、STV_INFLIGHT テーブルを使用します。トラブルシューティングを行う場合、実行時間が長いクエリのステータスを確認するのに役立ちます。

STV_INFLIGHT はリーダーノードのみのクエリは表示しません。詳細については、「リーダーノード専用関数」を参照してください。STV_INFLIGHT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

STV_INFLIGHT によるトラブルシューティング

STV_INFLIGHT を使用してクエリまたはクエリのコレクションのパフォーマンスのトラブルシューティングを行う場合は、次の点に注意してください。

- 実行時間の長いオープントランザクションでは、一般的に負荷が増加します。これらのオープントランザクションにより、他のクエリの実行時間が長くなる可能性があります。
- 実行時間が長い COPY ジョブと ETL ジョブは、大量のコンピューティングリソースを消費している場合、クラスターで実行されている他のクエリに影響する可能性があります。ほとんどの場合、これらの実行時間の長いジョブをあまり使用しない時間帯に移動すると、レポートや分析のワークロードのパフォーマンスが向上します。
- STV_INFLIGHT に関連する情報を提供するビューがあります。これらには、SQL コマンドのクエリテキストをキャプチャする STL_QUERYTEXT と、STV_INFLIGHT を STL_QUERYTEXT に結合する SVV_QUERY_INFLIGHT が含まれます。トラブルシューティングのために、STV_RECENTS を STV_INFLIGHT で使用することもできます。例えば、STV_RECENTS は、特定のクエリの状態が実行中または完了であるかを示すことができます。この情報を STV_INFLIGHT の結果と組み合わせると、クエリのプロパティとコンピューティングリソースへの影響に関する詳細情報を得ることができます。

Amazon Redshift コンソールを使用して実行中のクエリをモニタリングすることもできます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
slice	integer	クエリが実行されているスライス。

列名	データ型	説明
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドは空になります。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、この値は一定です。この列を使用して、STL_ERROR テーブルに結合できます。
starttime	timestamp	クエリが開始された時刻。
text	character(100)	クエリテキスト。ステートメントが 100 文字を超えた場合は切り捨てられます。
suspended	integer	クエリが中断されているかどうかを示します。0 = false で、1 = true です。
insert_pristine	integer	現在のクエリの実行中に書き込みクエリが実行可能であるかどうか。1 = 書き込みクエリが許可されていません。0 = 書き込みクエリが許可されています。この列は、デバッグで使用することが意図されています。
concurrency_scaling_status	integer	クエリがメインクラスターで実行されたのか、または同時実行スケーリングクラスターで実行されたのかを示します。有効な値は次のとおりです。 0 - メインクラスターで実行 1 - 同時実行スケーリングクラスターで実行

サンプルクエリ

データベースで現在実行中のアクティブなクエリをすべて表示するには、以下のクエリを入力します。

```
select * from stv_inflight;
```

以下の出力例には、STV_INFLIGHT クエリ自体と、`avgwait.sql` というスクリプトから実行されているクエリが表示されています。

```
select slice, query, trim(label) querylabel, pid,
starttime, substring(text,1,20) querytext
from stv_inflight;

slice|query|querylabel | pid |        starttime          |      querytext
-----+-----+-----------+-----+--------------------------+--------------------
1011 |  21 |           | 646 |2012-01-26 13:23:15.645503|select slice, query,
1011 |  20 |avgwait.sql| 499 |2012-01-26 13:23:14.159912|select avg(datediff(
(2 rows)
```

次のクエリは、concurrency_scaling_status を含む複数の列を選択します。この列は、クエリが同時実行スケーリングクラスターに送信されているかどうかを示します。一部の結果で値が 1 である場合は、同時実行スケーリングコンピューティングリソースが使用されていることを示します。詳細については、「[同時実行スケーリング]」を参照してください。

```
select userid,
query,
pid,
starttime,
text,
suspended,
concurrency_scaling_status
 from STV_INFLIGHT;
```

サンプル出力は、1 つのクエリが同時実行スケーリングクラスターに送信されていることを示しています。

```
 query  | pid     |         starttime           |   text                      | suspended
 |  concurrency_scaling_status
--------+---------
+-----------------------------|------------------------|---------------|------------------------
```

```
1234567 | 123456  | 2012-01-26 13:23:15.645503 | select userid, query...  0
    1
2345678 | 234567  | 2012-01-26 13:23:14.159912 | select avg(datediff(...  0
    0
(2 rows)
```

クエリパフォーマンスのトラブルシューティングの詳細については、「クエリのトラブルシューティング」を参照してください。

STV_LOAD_STATE

実行中の COPY ステートメントの現在の状態についての情報を取得するには、STV_LOAD_STATE テーブルを使用します。

COPY コマンドは、100 万レコードがロードされるたびにこの表を更新します。

STV_LOAD_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
session	integer	ロードを実行している処理のセッション PID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	ノードスライス番号。
pid	integer	プロセス ID。セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、この値は一定です。
recordtime	timestamp	レコードが記録された時刻。
bytes_to_load	bigint	このスライスによってロードされるバイトの総数。ロードされるデータが圧縮されている場合は 0 になります。

列名	データ型	説明
bytes_loaded	bigint	このスライスによってロードされたバイト数。ロードされるデータが圧縮されている場合、この値は、データを解凍した後のバイト数になります。
bytes_to_load_compressed	bigint	このスライスによってロードされる圧縮データのバイトの総数。ロードされるデータが圧縮されていない場合は 0 になります。
bytes_loaded_compressed	bigint	このスライスによってロードされた圧縮データのバイト数。ロードされるデータが圧縮されていない場合は 0 になります。
lines	integer	このスライスによってロードされた行数。
num_files	integer	このスライスによってロードされるファイル数。
num_files_complete	integer	このスライスによってロードされたファイル数。
current_file	character (256)	このスライスによってロードされているファイルの名前。
pct_complete	integer	このスライスによるデータロードで完了している割合 (パーセント)。

サンプルクエリ

COPY コマンドの各スライスの進行状況を表示するには、以下のクエリを入力します。この例では、最後の COPY コマンドに関する情報を取得するために、PG_LAST_COPY_ID() 関数が使用されます。

```
select slice , bytes_loaded, bytes_to_load , pct_complete from stv_load_state where
 query = pg_last_copy_id();

 slice | bytes_loaded | bytes_to_load | pct_complete
-------+---------------+---------------+--------------
     2 |            0 |             0 |            0
     3 |     12840898 |      39104640 |           32
(2 rows)
```

STV_LOCKS

データベース内のテーブルに対する現時点での更新をすべて表示するには、STV_LOCKS テーブルを使用します。

Amazon Redshift は、複数のユーザーが 1 つのテーブルを同時に更新することを防ぐため、テーブルをロックします。STV_LOCKS テーブルに現在のテーブル更新がすべて表示されているときに、STL_TR_CONFLICT テーブルをクエリすると、ロックの衝突のログを見ることができます。SVV_TRANSACTIONS 表示を使用して、開いているトランザクションとロックの競合問題を識別します。

STV_LOCKS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
table_id	bigint	ロックを取得するテーブルのテーブル ID
last_commit	timestamp	テーブル内での最後のコミットのタイムスタンプ。
last_update	timestamp	最後のテーブル更新のタイムスタンプ。
lock_owner	bigint	ロックに関連付けられたトランザクション ID。
lock_owner_pid	bigint	ロックに関連付けられたプロセス ID。
lock_owner_start_ts	timestamp	トランザクション開始時刻のタイムスタンプ。
lock_owner_end_ts	timestamp	トランザクション終了時刻のタイムスタンプ。
lock_status	character (22)	ロックを待っているかまたは保持している処理の状態。

サンプルクエリ

現在のトランザクションで発生しているすべてのロックを表示するには、以下のコマンドを入力します。

```
select table_id, last_update, lock_owner, lock_owner_pid from stv_locks;
```

このクエリは、以下のような出力を返します。出力には、現在有効となっている 3 つのロックが表示されています。

```
 table_id |        last_update         | lock_owner | lock_owner_pid
----------+----------------------------+------------+----------------
100004   | 2008-12-23 10:08:48.882319 |       1043 |          5656
100003   | 2008-12-23 10:08:48.779543 |       1043 |          5656
100140   | 2008-12-23 10:08:48.021576 |       1043 |          5656
(3 rows)
```

STV_ML_MODEL_INFO

機械学習モデルの現在の状態に関する情報です。

STV_ML_MODEL_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
schema_name	char(128)	モデルの名前空間。
user_name	char(128)	モデルの所有者。
model_name	char(128)	モデルの名前です。
life_cycle	char(20)	モデルのライフサイクルステータス。
is_refreshable	integer	トレーニングクエリの元のテーブルと列がまだ存在し、ユーザーがそれらに対するアクセス許可を持っている場合、それが更新可能かどうかについ

列名	データ型	説明
		てのモデルの状態。指定できる値は 1 (更新可能) と 0 (更新不可) です。
model_state	char(128)	モデルの現在の状態。

サンプルクエリ

次のクエリでは、機械学習モデルの現在の状態を表示します。

```
SELECT schema_name, model_name, model_state
FROM stv_ml_model_info;

 schema_name |         model_name           |             model_state
-------------+------------------------------+--------------------------------------
 public      | customer_churn_auto_model    | Train Model On SageMaker In Progress
 public      | customer_churn_xgboost_model | Model is Ready
(2 row)
```

STV_MV_DEPS

STV_MV_DEPS テーブルには、Amazon Redshift 内の、異なるマテリアライズドビュー間に存在する依存関係が表示されます。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

STV_MV_DEPS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
db_name	char(128)	特定のマテリアライズドビューを含むデータベース。
schema	char(128)	マテリアライズドビューのスキーマ。

列名	データ型	説明
name	char(128)	マテリアライズドビューの名前。
ref_schema	char(128)	このマテリアライズドビューが依存するマテリアライズドビューのスキーマ。
ref_name	char(128)	このマテリアライズドビューが依存するマテリアライズドビューの名前。
ref_datab ase_name	char(128)	このマテリアライズドビューが依存するデータベースの名前。

サンプルクエリ

次のクエリは、マテリアライズドビュー mv_over_foo がその定義内の依存関係として、マテリアライズドビュー mv_foo を使用していることを示す行を出力に返します。

```
CREATE SCHEMA test_ivm_setup;
CREATE TABLE test_ivm_setup.foo(a INT);
CREATE MATERIALIZED VIEW test_ivm_setup.mv_foo AS SELECT * FROM test_ivm_setup.foo;
CREATE MATERIALIZED VIEW test_ivm_setup.mv_over_foo AS SELECT * FROM
 test_ivm_setup.mv_foo;

SELECT * FROM stv_mv_deps;

 db_name | schema          | name        |   ref_schema    | ref_name |
 ref_database_name
---------+-----------------+-------------+----------------+----------
+------------------
 dev     | test_ivm_setup  | mv_over_foo | test_ivm_setup | mv_foo   | dev
```

STV_MV_INFO

STV_MV_INFO テーブルには、すべてのマテリアライズドビューの行、データが古くなっているかどうか、およびステータス情報が含まれます。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

STV_MV_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
db_name	char(128)	マテリアライズドビューを含むデータベース。
schema	char(128)	データベースのスキーマ。
name	char(128)	マテリアライズドビューの名前。
updated_upto_xid	bigint	内部で使用するために予約しています。
is_stale	char(1)	t は、マテリアライズドビューが古くなっていることを示します。古いマテリアライズドビューとは、ベーステーブルは更新されているが、マテリアライズドビューは更新されていないビューのことを指します。前回の再起動以降に更新が実行されていない場合、この情報は正確ではない可能性があります。 マテリアライズドビューが可変関数に依存している場合、`is_stale` 列は常に `t` に設定されます。可変関数は、同じ引数 (1 つまたは複数) が与えられた場合、異なる結果を返します。例えば、日付やタイムスタンプを返すほとんどの関数は可変関数です。
owner_user_name	char(128)	マテリアライズドビューを所有するユーザー。
state	integer	マテリアライズドビューの状態。 • 0 – マテリアライズドビューは、更新時に完全に再計算されます。

列名	データ型	説明
		• 1 – マテリアライズドビューは増分です。 • 101 – 削除された列があるため、マテリアライズドビューを更新できません。この制約は、列がマテリアライズドビューで使用されていない場合でも適用されます。 • 102 – 列のタイプが変更されたため、マテリアライズドビューを更新できません。この制約は、列がマテリアライズドビューで使用されていない場合でも適用されます。 • 103 – テーブル名が変更されたため、マテリアライズドビューを更新できません。 • 104 – 列の名前が変更されたため、マテリアライズドビューを更新できません。この制約は、列がマテリアライズドビューで使用されていない場合でも適用されます。 • 105 – スキーマ名が変更されたため、マテリアライズドビューを更新できません。
autorewrite	char(1)	t は、マテリアライズドビューがクエリの自動書き換えに適格であることを示します。
autorefresh	char(1)	t は、マテリアライズドビューを自動的に更新できることを示します。

サンプルクエリ

すべてのマテリアライズドビューのステータスを表示するには、次のクエリを実行します。

```
select * from stv_mv_info;
```

このクエリは次のサンプル出力を返します。

```
 db_name |       schema        |    name   | updated_upto_xid | is_stale | owner_user_name
 | state | autorefresh | autorewrite
```

```
---------+--------------------+---------+------------------+----------
+----------------+-------+------------+------------
 dev      | test_ivm_setup     | mv      |             1031 | f         | catch-22
 |      1 |              1 |            0
 dev      | test_ivm_setup     | old_mv  |              988 | t         | lotr
 |      1 |              0 |            1
```

STV_NODE_STORAGE_CAPACITY

STV_NODE_STORAGE_CAPACITY テーブルには、クラスター内の各ノードについて、ストレージ容量の合計と使用容量の合計の詳細が表示されます。これには、各ノードの行が含まれます。

STV_NODE_STORAGE_CAPACITY は、スーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
node	integer	ノード番号。
used	integer	ノード上で現在使用されている 1 MB ディスクブロックの数。RA3 ノードタイプの場合、使用されるブロックには、ローカルにキャッシュされたブロックと Amazon S3 で永続化されたブロックの両方が含まれます。
capacity	integer	ノード用のプロビジョンド合計ストレージ容量 (1 MB ブロック)。容量には、内部使用のために DC2 ノードタイプ上の Amazon Redshift によって予約されている領域が含まれます。容量は、ユーザーデータに使用できるノードスペースの量である公称ノード容量よりも大きくなります。RA3 ノードタイプの場合、この容量はクラスターのマネージドストレージクォータの合計と同じです。ノードタイプ別の容量の詳細については、「Amazon Redshift 管理ガイド」の「ノードタイプの詳細」を参照してください。

サンプルクエリ

> **Note**
>
> 以下の例の結果は、クラスターのノード仕様によって異なります。列 `capacity` を SQL SELECT に追加して、クラスターの容量を確保します。

次のクエリでは、使用済み領域と合計容量を 1 MB のディスクブロックで返します。この例は、2 ノードの dc2.8xlarge クラスターで実行されました。

```
select node, used from stv_node_storage_capacity order by node;
```

このクエリは次のサンプル出力を返します。

```
 node | used
------+-------
    0 | 30597
    1 | 27089
```

次のクエリでは、使用済み領域と合計容量を 1 MB のディスクブロックで返します。この例は、2 ノードの ra3.16xlarge クラスターで実行されました。

```
select node, used from stv_node_storage_capacity order by node;
```

このクエリは次のサンプル出力を返します。

```
 node | used
------+-------
    0 | 30591
    1 | 27103
```

STV_PARTITIONS

Amazon Redshift のディスク速度の性能とディスクの利用状況を調べるには、STV_PARTITIONS テーブルを使用します。

STV_PARTITIONS は、論理ディスクボリュームごとに、ノードあたり 1 つの行を含んでいます。

STV_PARTITIONS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
owner	integer	パーティションを所有するディスクノード。
host	integer	パーティションに物理的にアタッチされているノード。
diskno	integer	パーティションを含むディスク。
part_begin	bigint	パーティションのオフセット。ミラーブロック用にスペースを開くため、raw デバイスは論理的にパーティション化されています。
part_end	bigint	パーティションの末尾。
used	integer	パーティション上で現在使用されている 1 MB ディスクブロックの数。
tossed	integer	削除の準備は整っているが、存在している箇所のディスクアドレスを解放することが安全でないためにまだ削除されていないブロックの数。アドレスがすぐに解放されると、保留中のトランザクションがディスク上の同じ場所に書き込むことができてしまいます。このため、これらの tossed ブロックは、次のコミット時に解放されます。INSERT 処理中やディスクベースのクエリの処理中にテーブル列がドロップされた場合などに、ディスクブロックが tossed とマークされることがあります。
capacity	integer	1 MB ディスクブロック内のパーティションの総容量
reads	bigint	最後のクラスターの再起動以降に発生した読み取りの数。
writes	bigint	最後のクラスターの再起動以降に発生した書き込みの数。

列名	データ型	説明
seek_forward	integer	以前のリクエストアドレスが与えられた場合に、リクエストが次のアドレスに対するものでない回数。
seek_back	integer	次のリクエストアドレスが与えられた場合に、リクエストが前のアドレスに対するものでない回数。
is_san	integer	パーティションが SAN に属しているかどうかを示します。有効な値は、**0** (false) または **1** (true) です。
失敗	integer	この列は廃止されました。
mbps	integer	1 秒あたりのメガバイト数で表したディスク速度。
mount	character (256)	デバイスへのディレクトリパス。

サンプルクエリ

以下のクエリは、1 MB ディスクブロック内での使用ディスクスペースと容量を返し、ディスク利用率を raw ディスク容量に対するパーセント数で表します。raw ディスク容量には、Amazon Redshift が内部用に予約するスペースの分も含まれます。このため、ユーザーが利用できるディスク容量として表示される名目上のディスク容量より大きな数字になります。Amazon Redshift マネジメントコンソールの [Performance] (パフォーマンス) タブにある [Percentage of Disk Space Used] (使用されているディスク容量の割合) メトリクスは、クラスターによって使用されている公称ディスク容量の割合を報告します。使用容量をクラスターの名目上のディスク容量以内に維持するために、[ディスク使用率] メトリクスを監視することをお勧めします。

> **Important**
>
> クラスターの名目上のディスク容量は超えないようにしてください。名目上のディスク容量を超えることは、状況によっては可能ではありますが、クラスターの耐障害性が低下し、データ損失の可能性が増加します。

この例は、2 ノードクラスターで、1 ノードあたりの論理ディスクパーティションが 6 個という環境で実行されています。スペースは各ディスク間で非常に均等に配分利用 (約 25% ずつ) されています。

```
select owner, host, diskno, used, capacity,
(used-tossed)/capacity::numeric *100 as pctused
from stv_partitions order by owner;

 owner | host | diskno |  used  | capacity | pctused
-------+------+--------+--------+----------+---------
   0   |  0   |    0   | 236480 |  949954  | 24.9
   0   |  0   |    1   | 236420 |  949954  | 24.9
   0   |  0   |    2   | 236440 |  949954  | 24.9
   0   |  1   |    2   | 235150 |  949954  | 24.8
   0   |  1   |    1   | 237100 |  949954  | 25.0
   0   |  1   |    0   | 237090 |  949954  | 25.0
   1   |  1   |    0   | 236310 |  949954  | 24.9
   1   |  1   |    1   | 236300 |  949954  | 24.9
   1   |  1   |    2   | 236320 |  949954  | 24.9
   1   |  0   |    2   | 237910 |  949954  | 25.0
   1   |  0   |    1   | 235640 |  949954  | 24.8
   1   |  0   |    0   | 235380 |  949954  | 24.8

(12 rows)
```

STV_QUERY_METRICS

ユーザー定義のクエリキュー (サービスクラス) でアクティブに実行されているクエリについて、処理される列数、CPU 使用率、入出力、ディスク使用率などのメトリクス情報を含みます。完了したクエリのメトリクスを確認するには、STL_QUERY_METRICS システムテーブルを表示します。

クエリメトリクスは、1 秒間隔でサンプリングされます。結果として、同じクエリが複数実行され、わずかに異なる時刻を返す場合があります。また、1 秒未満で実行されるクエリセグメントは記録されない場合があります。

STV_QUERY_METRICS は、クエリレベル、セグメントレベル、およびステップレベルでメトリクスを追跡および集計します。クエリセグメントとステップの詳細については、「クエリプランと実行ワークフロー」を参照してください。多くのメトリクス (`max_rows`、`cpu_time`など) は、ノードスライスを超えて合計されます。ノードスライスの詳細については、「データウェアハウスシステムのアーキテクチャ」を参照してください。

列がメトリクスをレポートするレベルを確認するには、segment 列および step_type 列を調べます。

- segment と step_type の両方が -1 である場合、列はクエリレベルでメトリクスをレポートします。
- segment が -1 でなく、step_typeが -1 である場合、列はメトリクスをセグメントレベルでレポートします。
- segment と step_type の両方が -1 でない場合、列はステップレベルでメトリクスをレポートします。

STV_QUERY_METRICS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したクエリを実行したユーザーの ID。
service_class	integer	WLM クエリキュー (サービスクラス) の ID。クエリキューは WLM 設定で定義されます。メトリクスはユーザー定義のキューについてのみレポートされます。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
starttime	timestamp	UTC で表されたクエリの実行開始時刻。秒の小数部の精度 (6 桁) を使用します。例: 2009-06-12 11:29:19.131358 。
スライス	integer	クラスターのスライスの数。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されま

列名	データ型	説明
		す。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。セグメント値が -1 である場合は、メトリクスセグメントの値はクエリレベルまでロールアップされます。
step_type	integer	実行されたステップのタイプ。ステップタイプの説明については、「ステップタイプ」を参照してください。
rows	bigint	ステップに処理された行数。
max_rows	bigint	ステップの列出力の最大数 (すべてのスライスにわたる集計値)。
cpu_time	bigint	CPU の使用時間 (ミリ秒)。セグメントレベルでは、すべてのスライスにわたるセグメントの CPU 時間の合計。クエリレベルでは、すべてのスライスとセグメントにわたるクエリの CPU 時間の合計。
max_cpu_time	bigint	CPU の最長使用時間 (ミリ秒)。セグメントレベルでは、すべてのスライスにわたるセグメントによる CPU の最長使用時間。クエリレベルでは、任意のクエリセグメントによる CPU の最長使用時間。
blocks_read	bigint	クエリまたはセグメントに読み取られた 1 MB ブロックの数。
max_blocks_read	bigint	セグメントに読み取られた 1 MB ブロックの最大数 (すべてのスライスにわたる集計値)。セグメントレベルでは、すべてのスライスにわたるセグメントによって読み取られた 1 MB ブロックの最大数。クエリレベルでは、任意のクエリセグメントによって読み取られた 1 MB ブロックの最大数。

列名	データ型	説明
run_time	bigint	すべてのスライスにわたって集計された合計実行時間。実行時間に待機時間は含まれません。 セグメントレベルでは、すべてのスライスにわたって合計されたセグメントの実行時間。クエリレベルでは、全てのスライスとセグメントにわたって合計されたクエリの実行時間。合計値であるため、実行時間はクエリの実行時間とは関連しません。
max_run_time	bigint	セグメントの最長経過時間 (ミリ秒) セグメントレベルでは、すべてのスライスにわたるセグメントの最長実行時間。クエリレベルでは、任意のクエリセグメントの最長実行時間。
max_blocks_to_disk	bigint	中間結果の書き込みに使用する最大ディスク容量 (1 MB ブロック)。セグメントレベルでは、全てのスライスにわたるセグメントによって使用された最大ディスク容量。クエリレベルでは、任意のクエリセグメントによって使用された最大ディスク容量。
blocks_to_disk	bigint	クエリまたはセグメントが中間結果の書き込みに使用するディスク容量 (1 MB ブロック)。
step	integer	実行されたクエリステップ。
max_query_scan_size	bigint	クエリによってスキャンされた最大データサイズ (MB)。セグメントレベルでは、すべてのスライスにわたるセグメントによってスキャンされたデータの最大サイズ。クエリレベルでは、任意のクエリセグメントによってスキャンされたデータの最大サイズ。
query_scan_size	bigint	クエリによってスキャンされたデータサイズ (MB)。
query_priority	integer	クエリの優先度です。有効な値は、`-1`、`0`、`1`、`2`、`3`、および `4` です。ここで、`-1` は、クエリ優先度がサポートされていないことを意味します。

列名	データ型	説明
query_queue_time	bigint	クエリがキューに入れられた時間 (マイクロ秒)。

ステップタイプ

次の表に、データベースユーザーに関連するステップタイプのリストを示します。内部利用のみを目的とするステップタイプは含まれません。ステップタイプが -1 の場合、メトリクスはステップレベルでレポートされません。

ステップタイプ	説明
1	テーブルのスキャン
2	行の挿入
3	行の集計
6	ソートステップ
7	マージステップ
8	分散ステップ
9	ブロードキャスト分散ステップ
10	ハッシュ結合
11	マージ結合
12	保存ステップ
14	ハッシュ
15	ネストされたループ結合
16	フィールドと表現の投影
17	返される行数の制限

ステップタイプ	説明
18	Unique
20	行の削除
26	返される行のソート数の制限
29	ウィンドウ関数のコンピューティング
32	UDF
33	Unique
37	リーダーノードからクライアントに行を返す
38	コンピュートノードからリーダーノードに行を返す
40	Spectrum スキャン

サンプルクエリ

CPU 時間の長いアクティブなクエリ (1,000 秒以上) を検索するには、次のクエリを実行します。

```
select query, cpu_time / 1000000 as cpu_seconds
from stv_query_metrics where segment = -1 and cpu_time > 1000000000
order by cpu_time;

query | cpu_seconds
------+------------
25775 |        9540
```

ネストされたループ結合を持ち、100 万以上の行を返したクエリを検索するには、次のクエリを実行します。

```
select query, rows
from stv_query_metrics
where step_type = 15 and rows > 1000000
order by rows;

query | rows
```

```
------+-----------
25775 | 1580225854
```

60 秒以上実行されていて CPU 時間が 10 秒以下のアクティブなクエリを検索するには、次のクエリを実行します。

```
select query, run_time/1000000 as run_time_seconds
from stv_query_metrics
where segment = -1 and run_time > 60000000 and cpu_time < 10000000;

query | run_time_seconds
------+-----------------
25775 |              114
```

STV_RECENTS

現在アクティブなクエリや、最近データベースに対して実行されたクエリに関する情報を取得するには、STV_RECENTS テーブルを使用します。

STV_RECENTS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

STV_RECENTS によるトラブルシューティング

STV_RECENTS は、クエリまたはクエリのコレクションが現在実行中であるか完了しているかを判断する場合に特に役立ちます。また、クエリが実行されていた期間も示します。これは、どのクエリの実行時間が長いかを把握するのに役立ちます。

STV_RECENTS を STV_INFLIGHT などの他のシステムビューに結合して、実行中のクエリに関する追加のメタデータを収集できます (この方法を示すサンプルクエリセクションがあります)。このビューから返されたレコードを Amazon Redshift コンソールのモニタリング機能と共に使用して、リアルタイムでトラブルシューティングを行うこともできます。

STV_RECENTS を補完するシステムビューには、SQL コマンドのクエリテキストを取得する STL_QUERYTEXT と、STV_INFLIGHT を STL_QUERYTEXT に結合する SVV_QUERY_INFLIGHT が含まれます。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
status	character(20)	クエリのステータス 有効な値は **Running**、**Done** です。
starttime	timestamp	クエリが開始された時刻。
duration	integer	セッション開始後の経過時間をマイクロ秒で示します。
user_name	character(50)	処理を実行しているユーザーの名前。
db_name	character(50)	データベースの名前
query	character(600)	600 文字以内のクエリテキスト。600 文字を超えるテキストは切り捨てられます。
pid	integer	クエリに関連付けられたセッションのプロセス ID。クエリが完了している場合は -1 です。

サンプルクエリ

データベースに対して現在実行されているクエリを確認するには、次のクエリを実行します。

```
select user_name, db_name, pid, query
from stv_recents
where status = 'Running';
```

以下の出力例には、TICKIT データベース上で実行されているクエリが 1 つ表示されています。

```

user_name | db_name |   pid   | query
----------+---------+---------+-------------
dwuser    | tickit  |  19996  |select venuename, venueseats from
venue where venueseats > 50000 order by venueseats desc;
```

次の例では、実行中のクエリまたはキュー内で実行を待機中のクエリ (存在する場合) のリストを返します。

```
select * from stv_recents where status<>'Done';

status |    starttime          | duration |user_name|db_name| query      | pid
-------+---------------------+----------+---------+-------+-----------+------
Running| 2010-04-21 16:11... | 281566454| dwuser  |tickit | select ...| 23347
```

複数の同時実行クエリを実行中で、そのうちのいくつかがキューにある場合を除き、このクエリは結果を返しません。

以下の例は、前の例を拡張したものです。この場合、真に「フライト中」 (待機中でなく実行中) のクエリは結果から除外されます。

```
select * from stv_recents where status<>'Done'
and pid not in (select pid from stv_inflight);
...
```

クエリパフォーマンスのトラブルシューティングの詳細については、「クエリのトラブルシューティング」を参照してください。

STV_SESSIONS

Amazon Redshift のアクティブなユーザーセッションに関する情報を確認するには、STV_SESSIONS テーブルを使用します。

セッションの履歴を表示したい場合は、STV_SESSIONS ではなく STL_SESSIONS テーブルを使用します。

STV_SESSIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_SESSION_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
starttime	timestamp	セッションが開始された時刻。

列名	データ型	説明
process	integer	セッションのプロセス ID。
user_name	character(50)	セッションに関連付けられたユーザー。
db_name	character(50)	セッションに関連付けられたデータベースの名前。
timeout_sec	int	タイムアウトするまで、セッションが非アクティブまたはアイドル状態を維持する最大秒数。0 の場合は、タイムアウトが設定されていないことを示します。

サンプルクエリ

現在他のユーザーが Amazon Redshift にログインしているかどうかを簡単に調べるには、以下のクエリを入力します。

```
select count(*)
from stv_sessions;
```

結果が 1 より大きい場合は、他に少なくとも 1 人のユーザーが現在データベースにログインしています。

Amazon Redshift のアクティブなセッションをすべて見るには、以下のクエリを入力します。

```
select *
from stv_sessions;
```

次の結果は、現在 Amazon Redshift で実行されている 4 つのアクティブなセッションを示しています。

```

       starttime          | process |user_name                     | db_name
       | timeout_sec
-------------------------+---------+----------------------------
+----------------------------+-------------
  2018-08-06 08:44:07.50 |   13779 | IAMA:aws_admin:admin_grp    | dev
       | 0
  2008-08-06 08:54:20.50 |   19829 | dwuser                      | dev
       | 120
```

```
  2008-08-06 08:56:34.50 |   20279 | dwuser                        | dev
      | 120
  2008-08-06 08:55:00.50 |   19996 | dwuser                        | tickit
      | 0
(3 rows)
```

ユーザー名にプレフィックス IAMA が付いている場合は、ユーザーがフェデレーティッドシングルサインオン を使用してサインオンしたことを示します。詳細については、「IAM 認証を使用したデータベースユーザー認証情報の生成」を参照してください。

STV_SLICES

スライスからノードへの現在のマッピングを見るには、STV_SLICES テーブルを使用します。

STV_SLICES 内の情報は主に、調査目的で使用されます。

STV_SLICES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
node	integer	スライスが存在するクラスターノード。
slice	integer	ノードスライス。
localslice	integer	この情報は、内部使用に限定されています。
type	character (1)	この情報は、内部使用に限定されています。

サンプルクエリ

どのクラスターノードがどのスライスを管理しているかを見るには、以下のクエリを入力します。

```
select node, slice from stv_slices;
```

このクエリは、次のサンプル出力を返します。

```
 node | slice
------+-------
    0 |     2
    0 |     3
    0 |     1
    0 |     0
(4 rows)
```

STV_STARTUP_RECOVERY_STATE

クラスターの再起動オペレーション中に一時的にロックされているテーブルの状態を記録します。Amazon Redshift は、クラスターの再起動後、古いトランザクションを解決するために処理中のテーブルを一時的にロックします。

STV_STARTUP_RECOVERY_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
db_id	integer	データベース ID。
table_id	integer	テーブル ID。
table_name	character(137)	テーブル名。

サンプルクエリ

一時的にロックされているテーブルをモニタリングするには、クラスターの再起動後に次のクエリを実行します。

```
select * from STV_STARTUP_RECOVERY_STATE;

  db_id | tbl_id | table_name
--------+--------+------------
 100044 | 100058 | lineorder
 100044 | 100068 | part
```

```
 100044 | 100072 | customer
 100044 | 100192 | supplier
(4 rows)
```

STV_TBL_PERM

STV_TBL_PERM テーブルには、現在のセッション用にユーザーが作成した一時テーブルを含め、Amazon Redshift の永続テーブルに関する情報が表示されます。STV_TBL_PERM には、すべてのデータベース内のすべてのテーブルに関する情報が含まれます。

このテーブルは、クエリの処理中にシステムが作成する一時的なデータベーステーブルの情報を表示する STV_TBL_TRANS とは異なります。

STV_TBL_PERM はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
slice	integer	テーブルに割り当てられたノードスライス。
id	integer	テーブル ID。
name	character (72)	テーブル名。
rows	bigint	スライス内のデータ行数。
sorted_rows	bigint	ディスク上でソート済みの、スライス内の行数。この数が ROWS の数と異なる場合は、テーブルに vacuum を実行して行をソートし直してください。
temp	integer	テーブルが一時テーブルかそうでないかを示します。0 = false、1 = true。
db_id	integer	テーブルが作成されたデータベースの ID。
insert_pristine	integer	内部使用を目的とします。

列名	データ型	説明
delete_pristine	integer	内部使用を目的とします。
backup	integer	値は、テーブルがスナップショットクラスターに含まれているかどうかを示します。0 = no、1 = yes。詳細については、CREATE TABLE コマンドの BACKUP パラメータを参照してください。
dist_style	integer	スライスが属するテーブルの分散スタイル。値の詳細については、「分散スタイルの表示」を参照してください。分散スタイルの詳細については、「分散スタイル」を参照してください。
block_count	integer	スライスが使用するブロックの数。ブロック数を計算できない場合、値は -1 です。

サンプルクエリ

以下のクエリは、個別のテーブル ID と名前を返します。

```
select distinct id, name
from stv_tbl_perm order by name;

   id   |          name
--------+-------------------------
 100571 | category
 100575 | date
 100580 | event
 100596 | listing
 100003 | padb_config_harvest
 100612 | sales
...
```

テーブル ID は他のシステムテーブルも使用するため、どのテーブル ID がどのテーブルに対応しているかを知っておくと非常に便利です。この例では、SELECT DISTINCT を使用して重複内容が削除されています (テーブルは複数のスライスに配分されています)。

VENUE テーブル内の各列で使用されているブロックの数を調べるには、以下のクエリを使用します:

```
select col, count(*)
```

```
from stv_blocklist, stv_tbl_perm
where stv_blocklist.tbl = stv_tbl_perm.id
and stv_blocklist.slice = stv_tbl_perm.slice
and stv_tbl_perm.name = 'venue'
group by col
order by col;

 col | count
-----+-------
   0 |     8
   1 |     8
   2 |     8
   3 |     8
   4 |     8
   5 |     8
   6 |     8
   7 |     8
(8 rows)
```

使用に関する注意事項

ROW 列には、vacuum されていない (または SORT ONLY オプション付きで vacuum された) 削除済みの行の数が含まれます。このため、STV_TBL_PERM テーブル内の ROW 列の SUM は、特定のテーブルを直接クエリしたときの COUNT(*) 結果と一致しないことがあります。例えば、VENUE から 2 行が削除された場合、COUNT(*) の結果は 200 ですが、SUM (ROWS) の結果は 202 のままです:

```
delete from venue
where venueid in (1,2);

select count(*) from venue;
count
-------
200
(1 row)

select trim(name) tablename, sum(rows)
from stv_tbl_perm where name='venue' group by name;

tablename | sum
-----------+-----
venue     | 202
```

```
(1 row)
```

STV_TBL_PERM のデータを同期するには、VENUE テーブルの完全バキュームを実行します。

```
vacuum venue;

select trim(name) tablename, sum(rows)
from stv_tbl_perm
where name='venue'
group by name;

tablename | sum
-----------+-----
venue     | 200
(1 row)
```

STV_TBL_TRANS

現在メモリ内にある一時データベーステーブルに関する情報を取得するには、STV_TBL_TRANS テーブルを使用します。

一時テーブルとは通常、クエリ実行中に中間結果として使用される一時行セットです。STV_TBL_PERM には恒久的なデータベーステーブルに関する情報が含まれるという点で、STV_TBL_TRANS は STV_TBL_PERM と異なります。

STV_TBL_TRANS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
slice	integer	テーブルに割り当てられたノードスライス。
id	integer	テーブル ID。
rows	bigint	テーブル内のデータ行数。
size	bigint	テーブルに割り当てられたバイト数。
query_id	bigint	クエリ ID。

列名	データ型	説明
ref_cnt	integer	参照の数。
from_suspended	integer	このテーブルが、現在停止されているクエリの実行中に作成されたかどうかを示します。
prep_swap	integer	この一時テーブルが、必要に応じてディスクにスワップできるように準備されているかどうかを示します。(スワップは、メモリが少ない場合のみ発生します。)

サンプルクエリ

クエリ ID が 90 であるクエリの一時テーブル情報を見るには、以下のコマンドを入力します。

```
select slice, id, rows, size, query_id, ref_cnt
from stv_tbl_trans
where query_id = 90;
```

このクエリは、以下の出力例のように、クエリ 90 の一時テーブル情報を返します。

```
slice | id | rows | size | query_ | ref_ | from_      | prep_
      |    |      |      | id     | cnt  | suspended  | swap
------+----+------+------+--------+------+-----------+-------
 1013 | 95 |    0 |    0 |     90 |    4 |          0 |   0
    7 | 96 |    0 |    0 |     90 |    4 |          0 |   0
   10 | 96 |    0 |    0 |     90 |    4 |          0 |   0
   17 | 96 |    0 |    0 |     90 |    4 |          0 |   0
   14 | 96 |    0 |    0 |     90 |    4 |          0 |   0
    3 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 1013 | 99 |    0 |    0 |     90 |    4 |          0 |   0
    9 | 96 |    0 |    0 |     90 |    4 |          0 |   0
    5 | 96 |    0 |    0 |     90 |    4 |          0 |   0
   19 | 96 |    0 |    0 |     90 |    4 |          0 |   0
    2 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 1013 | 98 |    0 |    0 |     90 |    4 |          0 |   0
   13 | 96 |    0 |    0 |     90 |    4 |          0 |   0
    1 | 96 |    0 |    0 |     90 |    4 |          0 |   0
 1013 | 96 |    0 |    0 |     90 |    4 |          0 |   0
    6 | 96 |    0 |    0 |     90 |    4 |          0 |   0
```

```
    11 | 96 |     0 |     0 |      90 |      4 |           0 |   0
    15 | 96 |     0 |     0 |      90 |      4 |           0 |   0
    18 | 96 |     0 |     0 |      90 |      4 |           0 |   0
```

この例では、クエリデータがテーブル 95、96、および 98 に関係していることが分かります。このテーブルにはゼロバイトが割り当てられているため、このクエリはメモリ内で実行可能です。

STV_WLM_CLASSIFICATION_CONFIG

WLM の現在の分類ルールを表示します。

STV_WLM_CLASSIFICATION_CONFIG はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
id	integer	サービスクラス ID。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
condition	character(128)	クエリの状態。
action_seq	integer	将来の利用のために予約されています。
action	character(64)	将来の利用のために予約されています。
action_service_class	integer	アクション実行場所のサービスクラス。

サンプルクエリ

```
select * from STV_WLM_CLASSIFICATION_CONFIG;

id | condition                                   | action_seq | action |
 action_service_class
---+---------------------------------------------+------------+--------
+---------------------
 1 | (system user) and (query group: health)     |          0 | assign |
    1
```

```
  2 | (system user) and (query group: metrics)     |           0 | assign |
      2
  3 | (system user) and (query group: cmstats)     |           0 | assign |
      3
  4 | (system user)                                |           0 | assign |
      4
  5 | (super user) and (query group: superuser)    |           0 | assign |
      5
  6 | (query group: querygroup1)                   |           0 | assign |
      6
  7 | (user group: usergroup1)                     |           0 | assign |
      6
  8 | (user group: usergroup2)                     |           0 | assign |
      7
  9 | (query group: querygroup3)                   |           0 | assign |
      8
 10 | (query group: querygroup4)                   |           0 | assign |
      9
 11 | (user group: usergroup4)                     |           0 | assign |
      9
 12 | (query group: querygroup*)                   |           0 | assign |
     10
 13 | (user group: usergroup*)                     |           0 | assign |
     10
 14 | (querytype: any)                             |           0 | assign |
     11
(4 rows)
```

STV_WLM_QMR_CONFIG

WLM クエリモニタリングルール (QMR) の構成を記録します。詳細については、「WLM クエリモニタリングルール」を参照してください。

STV_WLM_QMR_CONFIG はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
service_class	integer	WLM クエリキュー (サービスクラス) の ID。クエリキューは WLM 設定で定義されます。ルールはユーザー定義の

列名	データ型	説明
		キューに対してのみ定義できます。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
rule_name	character(256)	クエリモニタリングのルールの名前。
action	character(256)	ルールアクション。指定できる値は log、hop、abort、change_query_priority です。
metric_name	character(256)	メトリクスの名前。
metric_operator	character(256)	メトリクス演算子。想定される値は >、=、< です。
metric_value	double	アクションをトリガーする指定されたメトリクスのしきい値。
action_value	character(256)	action が change_query_priority の場合、指定できる値は highest、high、normal、low、lowest です。 action が log、hop、または abort の場合、値は空です。

サンプルクエリ

5 より大きいすべてのサービスクラスについて QMR ルール定義 (ユーザー定義のキューを含む) を表示するには、次のクエリを実行します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

```
Select *
from stv_wlm_qmr_config
where service_class > 5
order by service_class;
```

STV_WLM_QUERY_QUEUE_STATE

サービスクラスのクエリキューの現在の状態を記録します。

STV_WLM_QUERY_QUEUE_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
service_class	integer	サービスクラスの ID。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
position	integer	キュー内でのクエリの位置。**position** 値が最も小さいクエリが次に実行されます。
task	integer	ワークロードマネージャ全体を通じてクエリを追跡するために使用される ID。複数のクエリ ID と関連付けることができます。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
query	integer	クエリ ID。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
slot_count	integer	WLM クエリスロットの数。
start_time	timestamp	クエリがキューに入った時刻
queue_time	bigint	クエリがキューに入ってからの時間をマイクロ秒で示します。

サンプルクエリ

以下のクエリは、サービスクラス 4 以上のキュー内のクエリを表示します。

```
select * from stv_wlm_query_queue_state
where service_class > 4
order by service_class;
```

このクエリは次のサンプル出力を返します。

```
 service_class | position | task | query | slot_count |        start_time          |
 queue_time
---------------+----------+------+-------+------------+----------------------------
+------------
             5 |        0 |  455 |   476 |          5 | 2010-10-06 13:18:24.065838 |
 20937257
             6 |        1 |  456 |   478 |          5 | 2010-10-06 13:18:26.652906 |
 18350191
(2 rows)
```

STV_WLM_QUERY_STATE

WLM に追跡されているクエリの現在の状態を記録します。

STV_WLM_QUERY_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
xid	integer	クエリまたはサブクエリのトランザクション ID。
task	integer	ワークロードマネージャ全体を通じてクエリを追跡するために使用される ID。複数のクエリ ID と関連付けることができます。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。

列名	データ型	説明
query	integer	クエリ ID。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
service_class	integer	サービスクラスの ID。サービスクラスの ID のリストについては、「WLM サービスクラス ID」を参照してください。
slot_count	integer	WLM クエリスロットの数。
wlm_start_time	timestamp	クエリがシステムテーブルキューまたはショートクエリキューに入った時刻。

列名	データ型	説明
state	character(16)	クエリまたはサブクエリの現在の状態。 可能な値には以下のものがあります。 • `Classified` – クエリがサービスクラスに割り当てられています。 • `Completed` – クエリが実行を終了しました。クエリは、正常に実行されたか、キャンセルされたかのいずれかです。最終状態については、STL_QUERY の結果をチェックしてください。 • `Dequeued` – 内部使用限定。 • `Evicted` – 再起動のためにサービスクラスからクエリが削除されました。 • `Evicting` – 再起動のためにサービスクラスからクエリが削除されています。 • `Initialized` – 内部使用限定。 • `Invalid` – 内部使用限定。 • `Queued` – 実行するスロットがないため、クエリがクエリキューに送信されました。 • `QueuedWaiting` – クエリがクエリキューで待機しています。 • `Rejected` – 内部使用限定。 • `Returning` – クエリが結果をクライアントに返しています。 • `Running` – クエリが実行中です。 • `TaskAssigned` – 内部使用限定。
queue_time	bigint	クエリがキューに入ってからの時間 (マイクロ秒)。
exec_time	bigint	クエリが実行されている時間 (マイクロ秒)。

列名	データ型	説明
query_priority	char(20)	クエリの優先度です。有効な値は、n/a、lowest、low、normal、high、および highest です。ここで、n/a は、クエリ優先度がサポートされていないことを意味します。

サンプルクエリ

次のクエリでは、4 より大きいサービスクラスで現在実行中のすべてのクエリを表示します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

```
select xid, query, trim(state) as state, queue_time, exec_time
from stv_wlm_query_state
where service_class > 4;
```

このクエリは、次のサンプル出力を返します。

```
xid    | query | state   | queue_time | exec_time
-------+-------+---------+------------+-----------
100813 | 25942 | Running |          0 |    1369029
100074 | 25775 | Running |          0 | 2221589242
```

STV_WLM_QUERY_TASK_STATE

サービスクラスクエリタスクの現在の状態を表示します。

STV_WLM_QUERY_TASK_STATE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
service_class	integer	サービスクラスの ID。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

列名	データ型	説明
task	integer	ワークロードマネージャ全体を通じてクエリを追跡するために使用される ID。複数のクエリ ID と関連付けることができます。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
query	integer	クエリ ID。クエリを再起動すると、そのクエリには新しいタスク ID ではなく、新しいクエリ ID が割り当てられます。
slot_count	integer	WLM クエリスロットの数。
start_time	timestamp	クエリの実行が開始された時刻。
exec_time	bigint	クエリが実行されている時間をマイクロ秒で示します。

サンプルクエリ

次のクエリでは、4 より大きいサービスクラスのクエリの現在の状態を表示します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

```
select * from stv_wlm_query_task_state
where service_class > 4;
```

このクエリは、次のサンプル出力を返します。

```

service_class | task | query |         start_time         | exec_time
--------------+------+-------+----------------------------+-----------
    5         |  466 |   491 | 2010-10-06 13:29:23.063787 | 357618748
(1 row)
```

STV_WLM_SERVICE_CLASS_CONFIG

WLM のサービスクラス設定を記録します。

STV_WLM_SERVICE_CLASS_CONFIG はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
service_class	integer	サービスクラスの ID。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
queueing_strategy	character(32)	将来の利用のために予約されています。
num_query_tasks	integer	サービスクラスの現在の実際の同時実行レベル。`num_query_tasks` と `target_num_query_tasks` が異なる場合、WLM の動的移行が処理中です。値 `-1` は、自動 WLM が設定されていることを示します。
target_num_query_tasks	integer	最新の WLM の設定変更によって設定された同時実行レベル。
evictable	character(8)	将来の利用のために予約されています。
eviction_threshold	bigint	将来の利用のために予約されています。
query_working_mem	integer	サービスクラスに割り当てられた作業メモリの現在の実際の量 (スロットとノードあたりの MB 単位)。`query_working_mem` と `target_query_working_mem` が異なる場合、WLM の動的移行が処理中です。値 `-1` は、自動 WLM が設定されていることを示します。
target_query_working_mem	integer	作業メモリの量 (スロットとノードあたりの MB 単位) は、最新の WLM の設定変更によって設定されます。
min_step_mem	integer	将来の利用のために予約されています。
name	character(64)	サービスクラスの名前。
max_execution_time	bigint	クエリが終了される前に実行できる時間をミリ秒で示します。
user_group_wild_card	ブール値	TRUE の場合、WLM キューは、WLM 設定のユーザーグループ文字列内のアスタリスク (*) をワイルドカードとして扱います。

列名	データ型	説明
query_group_wild_card	ブール値	TRUE の場合、WLM キューは、WLM 設定のクエリグループ文字列内のアスタリスク (*) をワイルドカードとして扱います。
concurrency_scaling	character(20)	同時実行スケーリングが on か off かを説明します。
query_priority	character(20)	クエリ優先度の値。
user_role_wild_card	ブール値	TRUE の場合、WLM キューは、WLM 設定のユーザー文字列内のアスタリスク (*) をワイルドカードとして扱います。

サンプルクエリ

最初のユーザー定義サービスクラスはサービスクラス 6 であり、サービスクラス #1 という名前です。次のクエリでは、4 より大きいサービスクラスの現在の設定を表示します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

```
select rtrim(name) as name,
num_query_tasks as slots,
query_working_mem as mem,
max_execution_time as max_time,
user_group_wild_card as user_wildcard,
query_group_wild_card as query_wildcard
from stv_wlm_service_class_config
where service_class > 4;

name                           | slots | mem | max_time | user_wildcard | query_wildcard
-----------------------------+--------+-----+----------+---------------+---------------
Service class for super user |     1 | 535 |        0 | false         | false
Queue 1                      |     5 | 125 |        0 | false         | false
Queue 2                      |     5 | 125 |        0 | false         | false
Queue 3                      |     5 | 125 |        0 | false         | false
Queue 4                      |     5 | 627 |        0 | false         | false
Queue 5                      |     5 | 125 |        0 | true          | true
Default queue                |     5 | 125 |        0 | false         | false
```

次のクエリは、WLM の動的移行のステータスを示します。移行が処理されている間、num_query_tasks と target_query_working_mem がターゲット値と同等になるまで更新されます。詳細については、「WLM の動的設定プロパティと静的設定プロパティ」を参照してください。

```
select rtrim(name) as name,
num_query_tasks as slots,
target_num_query_tasks as target_slots,
query_working_mem as memory,
target_query_working_mem as target_memory
from stv_wlm_service_class_config
where num_query_tasks > target_num_query_tasks
or query_working_mem > target_query_working_mem
and service_class > 5;

 name              | slots | target_slots | memory | target_mem
------------------+-------+--------------+--------+------------
 Queue 3          |     5 |           15 |    125 |       375
 Queue 5          |    10 |            5 |    250 |       125
 (2 rows)
```

STV_WLM_SERVICE_CLASS_STATE

サービスクラスの現在の状態を表示します。

STV_WLM_SERVICE_CLASS_STATE はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
service_class	integer	サービスクラスの ID。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
num_queued_queries	integer	現在キュー内にあるクエリの数。
num_executing_queries	integer	現在実行中のクエリの数。

列名	データ型	説明
num_serviced_queries	integer	サービスクラスにこれまで存在したクエリの数。
num_executed_queries	integer	Amazon Redshift が再開されてから実行されたクエリの数です。
num_evicted_queries	integer	Amazon Redshift が再開されてから削除されたクエリの数。クエリが削除された理由には、WLM タイムアウト、QMR のホップアクション、および同時実行スケーリングクラスターでのクエリの失敗があります。
num_concurrency_scaling_queries	integer	Amazon Redshift が再開されてから同時実行スケーリングクラスターで実行されたクエリの数。

サンプルクエリ

次のクエリでは、5 より大きいサービスクラスの状態を表示します。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。

```
select service_class, num_executing_queries,
num_executed_queries
from stv_wlm_service_class_state
where service_class > 5
order by service_class;
```

```
 service_class | num_executing_queries | num_executed_queries
---------------+-----------------------+----------------------
             6 |                     1 |                  222
             7 |                     0 |                  135
             8 |                     1 |                   39
(3 rows)
```

STV_XRESTORE_ALTER_QUEUE_STATE

STV_XRESTORE_ALTER_QUEUE_STATE を使用して、従来のサイズ変更中の各テーブルの移行の進行状況をモニタリングします。これは、ターゲットノードタイプが RA3 の場合、特に当てはまり

ます。RA3 ノードへの従来のサイズ変更について詳しくは、「従来のサイズ変更」を参照してください。

STV_XRESTORE_ALTER_QUEUE_STATE はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_RESTORE_STATE でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	サイズ変更を開始したユーザーの ID。
db_id	integer	データベースの ID。
schema	char(128)	スキーマの名前。
table_name	char(128)	テーブルの名前。
tbl	integer	テーブルの ID。
status	char(64)	テーブル移行の進捗ステータス。指定できる値は以下のとおりです。 • `Waiting`: 再分散の開始を待機中。 • `Applying`: 再分散中。 • `Finished`: 再分散完了。
task_type	integer	テーブルの再分散タイプ。指定できる値は以下のとおりです。 • 1: KEY • 2: EVEN 分散形式の詳細については、「分散スタイル」を参照してください。

サンプルクエリ

次のクエリは、データベース内のサイズ変更待ち、サイズ変更中、サイズ変更完了のステータスを持つテーブルの数を示します。

```
select db_id, status, count(*)
from stv_xrestore_alter_queue_state
group by 1,2 order by 3 desc

db_id  |   status    | count
-------+------------+------
694325 | Waiting    |   323
694325 | Finished   |    60
694325 | Applying   |     1
```

メインおよび同時実行スケーリングクラスターの SVCS ビュー

プレフィックス SVCS のある SVCS システムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

トピック

- SVCS_ALERT_EVENT_LOG
- SVCS_COMPILE
- SVCS_CONCURRENCY_SCALING_USAGE
- SVCS_EXPLAIN
- SVCS_PLAN_INFO
- SVCS_QUERY_SUMMARY
- SVCS_S3LIST
- SVCS_S3LOG
- SVCS_S3PARTITION_SUMMARY
- SVCS_S3QUERY_SUMMARY
- SVCS_STREAM_SEGS
- SVCS_UNLOAD_LOG

SVCS_ALERT_EVENT_LOG

パフォーマンスの問題を示している可能性のある条件がクエリオプティマイザによって特定された場合にアラートを記録します。このビューは、STL_ALERT_EVENT_LOG システムテーブルから派生していますが、同時実行スケーリングクラスターで実行されるクエリのスライスレベルは表示されません。SVCS_ALERT_EVENT_LOG テーブルを使用して、クエリパフォーマンスを向上させる機会を特定します。

複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。詳細については、「クエリ処理」を参照してください。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

SVCS_ALERT_EVENT_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
segment	integer	クエリセグメントを識別する番号。
step	integer	実行されたクエリステップ。
pid	integer	ステートメントとスライスに関連付けられるプロセス ID。複数のスライスで実行される場合、同じクエリに複数の PID がある可能性があります。

列名	データ型	説明
xid	bigint	ステートメントに関連付けられるトランザクション ID。
event	character (1024)	アラートイベントの説明。
solution	character (1024)	推奨される解決策。
event_time	timestamp	UTC で表されたクエリの開始時間。合計時間にはキューイングと実行が含まれます。秒の小数部は 6 桁の精度で表されます。例: **2009-06-12 11:29:19.131358** 。

使用に関する注意事項

SVCS_ALERT_EVENT_LOG を使用してクエリの潜在的な問題を特定し、「クエリパフォーマンスのチューニング」の説明に従ってデータベース設計を最適化して、クエリを再作成できます。SVCS_ALERT_EVENT_LOG は以下のアラートを記録します。

- 見つからない統計

  統計が見つかりません。データロードまたは大規模な更新の後で ANALYZE を実行し、COPY 操作で STATUPDATE を使用します。詳細については、「Amazon Redshift クエリの設計のベストプラクティス」を参照してください。

- ネステッドループ

  通常、ネステッドループは直積集合です。クエリを評価して、関与しているすべてのテーブルが効率的に結合されていることを確認します。

- 非常に選択的なフィルター

  スキャンされた行に対する返された行の比率が 0.05 未満です。スキャンされる行の数は `rows_pre_user_filter` の値であり、返される行の数は STL_SCAN システムテーブルの行の値です。結果セットを決定するために、クエリが著しく大量の行数をスキャンしていることを示します。この問題は、ソートキーが見つからない場合や正しくない場合に起こります。詳細については、「ソートキー」を参照してください。

- 過剰な数の非実体行

削除済みだがバキューム未処理としてマークされた比較的多数の行、または挿入済みだがコミットされていない比較的多数の行がスキャンによってスキップされました。詳細については、「テーブルのバキューム処理」を参照してください。

- サイズの大きな分散

  ハッシュ結合または集計で 100 万を超える行が再分散されました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。
- サイズの大きなブロードキャスト

  ハッシュ結合で 100 万を超える行がブロードキャストされました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。
- 直列実行

  内部テーブル全体が単一ノードに再分散されたために直列実行を強制する、DS_DIST_ALL_INNER 再分散スタイルがクエリプランで指定されました。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。

サンプルクエリ

次のクエリは、4 つのクエリに関するアラートイベントを表示します。

```
SELECT query, substring(event,0,25) as event,
substring(solution,0,25) as solution,
trim(event_time) as event_time from svcs_alert_event_log order by query;

 query |             event              |          solution              |     event_time

-------+-------------------------------+------------------------------
+---------------------
  6567 | Missing query planner statist | Run the ANALYZE command      | 2014-01-03
 18:20:58
  7450 | Scanned a large number of del | Run the VACUUM command to rec| 2014-01-03
 21:19:31
  8406 | Nested Loop Join in the query | Review the join predicates to| 2014-01-04
 00:34:22
 29512 | Very selective query filter:r | Review the choice of sort key| 2014-01-06
 22:00:00

(4 rows)
```

SVCS_COMPILE

スケーリングクラスターで実行されるクエリやメインクラスターで実行されるクエリを含む、クエリの各クエリセグメントのコンパイル時間と場所を記録します。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_COMPILE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SCL_COMPILE の詳細については、「SVL_COMPILE」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントに関連付けられるプロセス ID。
query	integer	クエリ ID。この ID を使用して、他の各種システムテーブルおよびビューを結合できます。
segment	integer	コンパイルするクエリセグメント。
locus	integer	セグメントが実行される場所。コンピューティングノード上にある場合は **1**、リーダーノード上にある場合は **2**。
starttime	timestamp	協定世界時 (UTC) で表されたコンパイルの開始時刻。

列名	データ型	説明
endtime	timestamp	UTC で表されたコンパイルの終了時刻。
compile	integer	コンパイルが再利用された場合は値 **0**、セグメントがコンパイルされた場合は値 **1**。

サンプルクエリ

この例では、クエリ 35878 と 35879 が同じ SQL ステートメントを実行します。クエリ 35878 の compile 列では 4 つのクエリセグメントに対して 1 が表示されており、それらのセグメントがコンパイルされたことを示しています。クエリ 35879 の compile 列ではすべてのセグメントについて `0` が表示されており、セグメントを再びコンパイルする必要がなかったことを示しています。

```
select userid, xid,  pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svcs_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
```

```
    100 | 112782 | 23028 | 35879 |        8 |      1 |         0 |         0
    100 | 112782 | 23028 | 35879 |        9 |      2 |         0 |         0
(20 rows)
```

SVCS_CONCURRENCY_SCALING_USAGE

同時実行スケーリングの使用期間を記録します。各使用期間は、個別の同時実行スケーリングクラスターがアクティブにクエリを処理している連続した期間です。

SVCS_CONCURRENCY_SCALING_USAGE このテーブルはスーパーユーザーに表示されます。データベースのスーパーユーザーは、すべてのユーザーに公開することを選択できます。

テーブルの列

列名	データ型	説明
start_time	タイムゾーンなしのタイムスタンプ	使用期間が開始するとき。
end_time	タイムゾーンなしのタイムスタンプ	使用期間が終了するとき。
クエリ	bigint	この使用期間中に実行されたクエリの数。
usage_in_seconds	numeric(27,0)	この使用期間の合計秒数。

サンプルクエリ

特定の期間の使用期間を秒単位で表示するには、次のクエリを入力します。

```
select * from svcs_concurrency_scaling_usage order by start_time;

start_time | end_time | queries | usage_in_seconds
```

```
-----------------------------+-----------------------------+----------+-------------------
2019-02-14 18:43:53.01063 | 2019-02-14 19:16:49.781649 | 48 | 1977
```

SVCS_EXPLAIN

実行するために送信されたクエリの EXPLAIN プランを表示します。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

SVCS_EXPLAIN はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
nodeid	integer	クエリの実行における 1 つ以上のステップに対応するノードの計画ノード識別子。
parentid	integer	親ノードの計画ノード識別子。親ノードには、いくつかの子ノードがあります。例えば、merge join は、結合されたテーブルに対する複数の scan の親です。
plannode	character(400)	EXPLAIN の出力のノードテキスト。コンピューティングノードでの実行を参照する計画ノードは、EXPLAIN 出力の先頭に **XN** が付けられます。

列名	データ型	説明
info	character(400)	計画ノードの修飾子およびフィルタ情報。例えば、この列には join の条件と WHERE 句の制限が含まれます。

サンプルクエリ

集計 join クエリの次の EXPLAIN 出力について考えます。

```
explain select avg(datediff(day, listtime, saletime)) as avgwait
from sales, listing where sales.listid = listing.listid;
                                  QUERY PLAN

------------------------------------------------------------------------------
 XN Aggregate  (cost=6350.30..6350.31 rows=1 width=16)
  ->  XN Hash Join DS_DIST_NONE  (cost=47.08..6340.89 rows=3766 width=16)
        Hash Cond: ("outer".listid = "inner".listid)
        -> XN Seq Scan on listing  (cost=0.00..1924.97 rows=192497 width=12)
        -> XN Hash  (cost=37.66..37.66 rows=3766 width=12)
              -> XN Seq Scan on sales  (cost=0.00..37.66 rows=3766 width=12)
(6 rows)
```

このクエリを実行し、クエリ ID が 10 の場合は、SVCS_EXPLAIN テーブルを使用して EXPLAIN コマンドが返すものと同じ種類の情報を確認できます。

```
select query,nodeid,parentid,substring(plannode from 1 for 30),
substring(info from 1 for 20) from svcs_explain
where query=10 order by 1,2;

query| nodeid |parentid|           substring            |    substring
-----+--------+--------+--------------------------------+-------------------
10   |      1 |      0 |XN Aggregate  (cost=6717.61..6  |
10   |      2 |      1 |  -> XN Merge Join DS_DIST_NO| Merge Cond:("outer"
10   |      3 |      2 |       -> XN Seq Scan on lis |
10   |      4 |      2 |       -> XN Seq Scan on sal |
(4 rows)
```

次のクエリについて考えます。

```
select event.eventid, sum(pricepaid)
```

```
from event, sales
where event.eventid=sales.eventid
group by event.eventid order by 2 desc;

eventid |   sum
--------+----------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
...
```

このクエリの ID が 15 の場合、以下のシステムテーブルクエリが実行された計画ノードを返します。この場合、ノードの順番は、実際の実行順序を示すために逆順にされます。

```
select query,nodeid,parentid,substring(plannode from 1 for 56)
from svcs_explain where query=15 order by 1, 2 desc;

query|nodeid|parentid|                          substring
-----+------+--------+--------------------------------------------------------
15   |    8 |      7 |                                -> XN Seq Scan on eve
15   |    7 |      5 |                          -> XN Hash(cost=87.98..87.9
15   |    6 |      5 |                          -> XN Seq Scan on sales(cos
15   |    5 |      4 |                    -> XN Hash Join DS_DIST_OUTER(cos
15   |    4 |      3 |              -> XN HashAggregate(cost=862286577.07..
15   |    3 |      2 |        -> XN Sort(cost=1000862287175.47..10008622871
15   |    2 |      1 |  -> XN Network(cost=1000862287175.47..1000862287197.
15   |    1 |      0 |XN Merge(cost=1000862287175.47..1000862287197.46 rows=87
(8 rows)
```

次のクエリは、ウィンドウ関数を含むすべてのクエリプランのクエリ ID を取得します。

```
select query, trim(plannode) from svcs_explain
where plannode like '%Window%';

query|                                     btrim
-----+------------------------------------------------------------------------
26   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
27   | -> XN Window(cost=1000985348268.57..1000985351256.98 rows=170 width=33)
(2 rows)
```

SVCS_PLAN_INFO

SVCS_PLAN_INFO テーブルを使用して、行のセットに関するクエリの EXPLAIN 出力を確認します。これは、クエリプランを確認する代替的な方法となります。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

SVCS_PLAN_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
nodeid	integer	クエリの実行における 1 つ以上のステップに対応するノードの計画ノード識別子。
segment	integer	クエリセグメントを識別する番号。
step	integer	クエリステップを識別する番号。
locus	integer	ステップが実行される場所。コンピューティングノード上にある場合は 0、リーダーノード上にある場合は 1。
plannode	integer	計画ノードの列挙値。計画ノードの列挙値については、次の表を参照してください (SVCS_EXPLAIN の PLANNODE 列には計画ノードのテキストが格納されます)。

列名	データ型	説明
startupcost	double precision	このステップの最初の行を返すのにかかる相対的コストの推定値。
totalcost	double precision	ステップの実行にかかる相対的コストの推定値。
rows	bigint	ステップによって生成される行数の推定値。
バイト	bigint	ステップによって生成されるバイト数の推定値。

サンプルクエリ

次の例は、EXPLAIN コマンドの使用と SVCS_PLAN_INFO テーブルに対するクエリの実行によって返されるシンプルな SELECT クエリのクエリプランを比較します。

```
explain select * from category;
QUERY PLAN
-------------------------------------------------------------
XN Seq Scan on category (cost=0.00..0.11 rows=11 width=49)
(1 row)

select * from category;
catid | catgroup | catname | catdesc
-------+----------+-----------+-----------------------------------------
1 | Sports | MLB | Major League Baseball
3 | Sports | NFL | National Football League
5 | Sports | MLS | Major League Soccer
...

select * from svcs_plan_info where query=256;

query | nodeid | segment | step | locus | plannode | startupcost | totalcost
| rows | bytes
-------+--------+---------+------+-------+----------+-------------+----------+------
+-------
256 | 1 | 0 | 1 | 0 | 104 | 0 | 0.11 | 11 | 539
256 | 1 | 0 | 0 | 0 | 104 | 0 | 0.11 | 11 | 539
(2 rows)
```

この例では、PLANNODE 104 は CATEGORY テーブルのシーケンシャルスキャンを参照します。

```
select distinct eventname from event order by 1;

eventname
------------------------------------------------------------------------
.38 Special
3 Doors Down
70s Soul Jam
A Bronx Tale
...

explain select distinct eventname from event order by 1;

QUERY PLAN
-------------------------------------------------------------------------------------
XN Merge (cost=1000000000136.38..1000000000137.82 rows=576 width=17)
Merge Key: eventname
-> XN Network (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Send to leader
-> XN Sort (cost=1000000000136.38..1000000000137.82 rows=576
width=17)
Sort Key: eventname
-> XN Unique (cost=0.00..109.98 rows=576 width=17)
-> XN Seq Scan on event (cost=0.00..87.98 rows=8798
width=17)
(8 rows)

select * from svcs_plan_info where query=240 order by nodeid desc;

query | nodeid | segment | step | locus | plannode | startupcost |
totalcost | rows | bytes
-------+--------+---------+------+-------+----------+------------------
+------------------+------+--------
240 | 5 | 0 | 0 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 5 | 0 | 1 | 0 | 104 | 0                | 87.98   | 8798 | 149566
240 | 4 | 0 | 2 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 0 | 3 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 0 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 4 | 1 | 1 | 0 | 117 | 0                | 109.975 | 576  | 9792
240 | 3 | 1 | 2 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 3 | 2 | 0 | 0 | 114 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 2 | 2 | 1 | 0 | 123 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
240 | 1 | 3 | 0 | 0 | 122 | 1000000000136.38 | 1000000000137.82 | 576 | 9792
```

```
(10 rows)
```

SVCS_QUERY_SUMMARY

SVCS_QUERY_SUMMARY ビューを使用して、クエリの実行についての全般的な情報を確認します。

SVCS_QUERY_SUMMARY の情報はすべてのノードからの情報の集計であることに注意してください。

> **Note**
>
> SVCS_QUERY_SUMMARY ビューには、Amazon Redshift によって完了されたクエリについての情報のみが含まれます。その他のユーティリティや DDL コマンドの情報は含まれません。Amazon Redshift によって完了されたすべてのステートメントの完全なリストと、DDL およびユーティリティコマンドを含めた情報については、SVL_STATEMENTTEXT ビューをクエリできます。
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_QUERY_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

SVL_QUERY_SUMMARY の詳細については、「SVL_QUERY_SUMMARY」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。

列名	データ型	説明
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
stm	integer	ストリーム。クエリの並行セグメントのセット。クエリには 1 つ以上のストリームがあります。
seg	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。
step	integer	実行されたクエリステップ。
maxtime	bigint	ステップを実行する最大時間 (マイクロ秒)。
avgtime	bigint	ステップを実行する平均時間 (マイクロ秒)。
rows	bigint	クエリステップに含まれるデータ行の数。
bytes	bigint	クエリステップに含まれるデータバイトの数。
rate_row	double precision	行ごとのクエリ実行率。
rate_byte	double precision	バイトごとのクエリ実行率。
label	text	ステッププラベル。これは、クエリステップ名と、該当する場合はテーブル ID とテーブル名 (例: scan tbl=100448 name =user) で構成されます。3 桁のテーブル ID は通常、一時テーブルのスキャンを照会します。 `tbl=0` は、通常、定数値のスキャンを指します。
is_diskbased	character(1)	クエリのこのステップが、クラスター内のいずれかのノードでディスクベースのオペレーションとして実行されたかどうか: true (**t**) または false (**f**)。ハッシュ、ソート、集計といった特定のステップのみがディスクベースで実行できます。ステップの多くのタイプは、常にメモリ内で実行されます。

列名	データ型	説明
workmem	bigint	クエリステップに割り当てられた作業メモリの量 (バイト単位)。
is_rrscan	character(1)	true (**t**) の場合は、ステップで範囲限定スキャンが使用されたことを示します。デフォルトは false (**f**) です。
is_delayed_scan	character(1)	true (**t**) の場合は、ステップで遅延スキャンが使用されたことを示します。デフォルトは false (**f**) です。
rows_pre_filter	bigint	永続テーブルのスキャンの場合、削除対象としてマークされた行をフィルタリングする前に出力された合計行数 (非実体の行)。

サンプルクエリ

クエリステップの処理情報を表示する

次のクエリは、クエリ 87 の各ステップの基本的な処理情報を示します。

```
select query, stm, seg, step, rows, bytes
from svcs_query_summary
where query = 87
order by query, seg, step;
```

このクエリは次のサンプル出力で示されているように、クエリ 87 についての処理情報を取得します。

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
```

```
87      |    3 |    5 |     4 |       0 |        0
(13 rows)
```

クエリステップがディスクに書き出されたかどうかを確認する

次のクエリは、クエリ ID 1025 (SVL_QLOG を参照して、クエリのクエリ ID の取得方法を確認してください) のクエリのステップがディスクに書き出されたかどうか、またはクエリが完全にインメモリで実行されたかどうかを示します。

```
select query, step, rows, workmem, label, is_diskbased
from svcs_query_summary
where query = 1025
order by workmem desc;
```

このクエリは、次のサンプル出力を返します。

```
query| step|  rows   |  workmem   |  label          | is_diskbased
-----+-----+---------+-----------+---------------+--------------
1025 |  0  |16000000|  141557760 |scan tbl=9      | f
1025 |  2  |16000000|  135266304 |hash tbl=142    | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist            | f
(4 rows)
```

IS_DISKBASED の値をスキャンすることで、ディスクに書き出されたクエリステップを確認できます。クエリ 1025 のハッシュステップはディスクで実行されました。ディスクで実行される可能性があるステップは、ハッシュ、集計、ソートステップを含みます。ディスクベースクエリのステップのみを表示するには、前述の例の SQL ステートメントに **`and is_diskbased = 't'`** 句を追加します。

SVCS_S3LIST

SVCS_S3LIST ビューを使用して、セグメントレベルで Amazon Redshift Spectrum クエリの詳細を確認します。1 つのセグメントで外部テーブルスキャンを 1 回実行できます。このビューは、SVL_S3LIST システムビューから派生していますが、同時実行スケーリングクラスターで実行されるクエリのスライスレベルは表示されません。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメイン

クラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_S3LIST はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVL_S3LIST の詳細については、「SVL_S3LIST」を参照してください。

テーブルの列

列名	データ型	説明
query	integer	クエリ ID。
segment	integer	セグメント番号。クエリは、複数のセグメントで構成されます。
node	integer	ノード番号。
eventtime	timestamp	イベントが記録された時刻 (UTC)。
bucket	char(256)	Amazon S3 バケット名。
プレフィックス	char(256)	Amazon S3 バケットがある場所のプレフィックス。
recursive	char(1)	サブフォルダの Recursive Scan の有無。
retrieved_files	integer	表示されたファイルの数。
max_file_size	bigint	表示されたファイルの最大ファイルサイズ。
avg_file_size	double precision	表示されたファイルの平均ファイルサイズ。

列名	データ型	説明
generated_splits	integer	ファイル分割数。
avg_split_length	double precision	ファイル分割の平均サイズ (バイト)。
duration	bigint	ファイルの表示期間 (マイクロ秒)

サンプルクエリ

以下の例は、最後に実行されたクエリについて SVCS_S3LIST をクエリします。

```
select *
from svcs_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3LOG

SVCS_S3LOG ビューを使用して、セグメントレベルで Redshift Spectrum クエリに関するトラブルシューティングの詳細を取得します。1 つのセグメントで外部テーブルスキャンを 1 回実行できます。このビューは、SVL_S3LOG システムテーブルから派生していますが、同時実行スケーリングクラスターで実行されるクエリのスライスレベルは表示されません。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_S3LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVL_S3LOG の詳細については、「SVL_S3LOG」を参照してください。

テーブルの列

列名	データ型	説明
pid	integer	プロセス ID。
query	integer	クエリ ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
step	integer	実行したクエリステップ。
node	integer	ノード番号。
eventtime	timestamp	イベントが記録された時刻 (UTC 時間)。
メッセージ	char(512)	ログエントリのメッセージ。

サンプルクエリ

次の例では、最後に実行されたクエリの SVCS_S3LOG をクエリします。

```
select *
from svcs_s3log
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3PARTITION_SUMMARY

SVCS_S3PARTITION_SUMMARY ビューを使用して、セグメントレベルで Redshift Spectrum クエリのパーティションの概要を取得します。1 つのセグメントで外部テーブルスキャンを 1 回実行できます。

> Note
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメイン

クラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_S3PARTITION_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVL_S3PARTITION の詳細については、「SVL_S3PARTITION」を参照してください。

テーブルの列

列名	データ型	説明
query	integer	クエリ ID。この値を使用して、他の各種システムテーブルおよびビューを結合できます。
segment	integer	セグメント番号。クエリは、複数のセグメントで構成されます。
割り当て	char(1)	ノード間のパーティション割り当てのタイプ。
min_starttime	timestamp	パーティション処理が開始した時刻 (UTC 時間)。
max_endtime	timestamp	パーティション処理が完了した時刻 (UTC 時間)。
min_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの最小処理時間 (マイクロ秒)。
max_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの最大処理時間 (マイクロ秒)。
avg_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの平均処理時間 (マイクロ秒)。
total_partitions	integer	外部テーブルのパーティションの合計数。

列名	データ型	説明
qualified_partitions	integer	適格なパーティションの合計数。
min_assigned_partitions	integer	1 つのノードに割り当てられたパーティションの最小数。
max_assigned_partitions	integer	1 つのノードに割り当てられたパーティションの最大数。
avg_assigned_partitions	bigint	1 つのノードに割り当てられたパーティションの平均数。

サンプルクエリ

以下の例は、最後に実行されたクエリに関するパーティションスキャンの詳細を取得します。

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svcs_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVCS_S3QUERY_SUMMARY

SVCS_S3QUERY_SUMMARY ビューを使用して、システムで実行されたすべての Redshift Spectrum クエリ (S3 クエリ) の概要を取得します。1 つのセグメントで外部テーブルスキャンを 1 回実行できます。

> **Note**
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、SVL ビューがメイン

クラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス SVL を持つビューに似ています。

SVCS_S3QUERY_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVL_S3QUERY の詳細については、「SVL_S3QUERY」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	指定のエントリを生成したユーザーの ID。
query	integer	クエリ ID。この値を使用して、他の各種システムテーブルおよびビューを結合できます。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
step	integer	実行したクエリステップ。
starttime	timestamp	このセグメントで Redshift Spectrum クエリの実行を開始した時刻 (UTC 時間)。1 つのセグメントで外部テーブルスキャンを 1 回行うことができます。
endtime	timestamp	このセグメントで Redshift Spectrum クエリの実行が完了した時刻 (UTC 時間)。1 つのセグメントで外部テーブルスキャンを 1 回行うことができます。
elapsed	integer	このセグメントで Redshift Spectrum クエリの実行にかかった時間 (マイクロ秒)。

列名	データ型	説明
aborted	integer	クエリがシステムによって停止されたかユーザーによってキャンセルされた場合、この列は **1** になります。クエリが最後まで実行された場合、この列は **0** になります。
external_table_name	char(136)	外部テーブルスキャンのテーブルの外部名の内部形式。
file_format	character(16)	外部テーブルデータのファイル形式。
is_partitioned	char(1)	true (**t**) の場合、この列の値は外部テーブルがパーティション化されていることを示します。
is_rrscan	char(1)	true (**t**) の場合、この列の値は範囲限定スキャンが適用されたことを示します。
is_nested	varchar(1)	true (**t**) の場合、この列の値は、ネストされた列のデータ型にアクセスされていることを示します。
s3_scanned_rows	bigint	Amazon S3 からスキャンされ、Redshift Spectrum レイヤーに送信された行数。
s3_scanned_bytes	bigint	Amazon S3 からスキャンされ Redshift Spectrum レイヤーに送信されたバイト数 (圧縮データに基づく)。
s3query_returned_rows	bigint	Redshift Spectrum レイヤーからクラスターに返された行数。
s3query_returned_bytes	bigint	Redshift Spectrum レイヤーからクラスターに返されたバイト数。Amazon Redshift に大量のデータが返されると、システムパフォーマンスに影響が及ぶ可能性があります。
files	integer	この Redshift Spectrum クエリで処理されたファイル数。ファイル数が少ないと、並列処理の利点は制限されます。
files_max	integer	1 つのスライスで処理されるファイルの最大数。

列名	データ型	説明
files_avg	integer	1 つのスライスで処理されるファイルの平均数。
splits	bigint	このセグメントで処理された分割の数。このスライスで処理された分割の数。例えば分割可能なデータファイルの容量が大きい場合 (約 512 MB を超えるデータファイルなど)、Redshift Spectrum はファイルを複数の S3 リクエストに分割し、並列処理を試みます。
splits_max	integer	このスライスで処理された分割の最大数。
splits_avg	bigint	このスライスで処理された分割の平均数。
total_split_size	bigint	処理されたすべての分割の合計サイズ。
max_split_size	bigint	処理された分割の最大サイズ (単位: バイト)。
avg_split_size	bigint	処理された分割の平均サイズ (単位: バイト)。
total_retries	bigint	このセグメントでの Redshift Spectrum クエリの再試行の合計数。
max_retries	integer	処理ファイルごとの再試行の最大数。
max_request_duration	bigint	個別ファイルリクエストの最長時間 (マイクロ秒)。実行時間の長いクエリはボトルネックの可能性があります。
avg_request_duration	bigint	ファイルリクエストの平均時間 (マイクロ秒)。
max_request_parallelism	integer	Redshift Spectrum クエリでの 1 つのスライスの並列リクエストの最大数。

列名	データ型	説明
avg_request_parallelism	double precision	Redshift Spectrum クエリでの 1 つのスライスの並列リクエストの平均数。
total_slowdown_count	bigint	外部テーブルスキャン中に発生したスローダウンエラーを含む Amazon S3 リクエストの合計数。
max_slowdown_count	integer	外部テーブルスキャン中に 1 つのスライスで発生したスローダウンエラーを含む Amazon S3 リクエストの最大数。

サンプルクエリ

次の例では、最後に実行されたクエリのスキャンステップの詳細を取得します。

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svcs_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+-----------------------
+------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                     0
 |                      0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                  8513
 |                 170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                     0
 |                      0 |     0
 4587 |       2 |  216671 |               0 |                0 |                     0
 |                      0 |     0
```

SVCS_STREAM_SEGS

ストリームと並行セグメントの間の関係をリスト表示します。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

SVCS_STREAM_SEGS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
stream	integer	クエリの並行セグメントのセット。
segment	integer	クエリセグメントを識別する番号。

サンプルクエリ

最後に実行したクエリのストリームと並行セグメントの間の関係を表示するには、次のクエリを入力します。

```
select *
from svcs_stream_segs
where query = pg_last_query_id();

 query | stream | segment
-------+--------+---------
    10 |      1 |       2
    10 |      0 |       0
    10 |      2 |       4
    10 |      1 |       3
```

```
    10 |       0 |        1
(5 rows)
```

SVCS_UNLOAD_LOG

SVCS_UNLOAD_LOG を使用して、UNLOAD オペレーションの詳細を取得します。

SVCS_UNLOAD_LOG は、UNLOAD ステートメントによって作成される各ファイルについて 1 行ずつレコードが作成されます。例えば、UNLOAD で 12 個のファイルが作成された場合、SVCS_UNLOAD_LOG にはそれに対応する 12 行が作成されます。このビューは、STL_UNLOAD_LOG システムテーブルから派生していますが、同時実行スケーリングクラスターで実行されるクエリのスライスレベルは表示されません。

> **Note**
>
> プレフィックス SVCS のあるシステムビューは、メインクラスターおよび同時実行スケーリングクラスターの両方のクエリに関する詳細を提供します。ビューは、STL テーブルがメインクラスターで実行されたクエリについてのみ情報を提供することを除いて、プレフィックス STL を持つテーブルに似ています。

SVCS_UNLOAD_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。
pid	integer	クエリステートメントに関連付けられるプロセス ID。
パス	character(1280)	Amazon S3 オブジェクトのファイルへの完全パス。
start_time	timestamp	UNLOAD オペレーションの開始時刻。
end_time	timestamp	UNLOAD オペレーションの終了時刻。

列名	データ型	説明
line_count	bigint	ファイルにアンロードされた行数。
transfer_size	bigint	転送バイト数。
file_format	character(10)	アンロードされたファイルの形式。

サンプルクエリ

UNLOAD コマンドによって Amazon S3 に書き込まれたファイルのリストを取得するには、UNLOAD が完了した後に Amazon S3 のリストオペレーションを呼び出します。しかし、Amazon S3 のリストオペレーションは進行とともに作成されていくため、この呼び出しが早すぎるとリストが不完全になる場合があります。完成した正式のリストをすぐに取得するには、SVCS_UNLOAD_LOG をクエリします。

以下のクエリでは、最後に完了されたクエリの UNLOAD によって作成されたファイルのパス名が返されます。

```
select query, substring(path,0,40) as path
from svcs_unload_log
where query = pg_last_query_id()
order by path;
```

このコマンドは、次のサンプル出力を返します。

```

 query |             path
 ------+---------------------------------------------
  2320 | s3://amzn-s3-demo-bucket/venue0000_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0001_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0002_part_00
  2320 | s3://amzn-s3-demo-bucket/venue0003_part_00
(4 rows)
```

メインクラスターの SVL ビュー

SVL ビューは、STL テーブルへのリファレンスと詳細情報のログを含む Amazon Redshift のシステムビューです。

これらのビューを使用すると、それらのテーブルで検索された一般的なクエリデータにすばやく簡単にアクセスできます。

> **Note**
>
> SVL_QUERY_SUMMARY ビューには、Amazon Redshift によって実行されたクエリに関する情報のみが含まれます。その他のユーティリティや DDL コマンドの情報は含まれません。Amazon Redshift によって実行されたすべてのステートメントの完全なリストと、DDL およびユーティリティコマンドを含めた情報については、SVL_STATEMENTTEXT ビューをクエリできます。

トピック

- SVL_S3PARTITION_SUMMARY
- SVL_S3QUERY
- SVL_S3QUERY_SUMMARY
- SVL_S3RETRIES
- SVL_SPATIAL_SIMPLIFY
- SVL_SPECTRUM_SCAN_ERROR
- SVL_STATEMENTTEXT
- SVL_STORED_PROC_CALL
- SVL_STORED_PROC_MESSAGES
- SVL_TERMINATE
- SVL_UDF_LOG
- SVL_USER_INFO
- SVL_VACUUM_PERCENTAGE

SVL_AUTO_WORKER_ACTION

自動最適化用に定義されたテーブルに Amazon Redshift によって実行された自動アクションを記録します。

SVL_AUTO_WORKER_ACTION はスーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
table_id	integer	テーブル識別子。
type	character (32)	レコメンデーションのタイプ。可能な値は、distkey と sortkey です。
status	character (128)	レコメンデーションの完了ステータス。指定可能な値は、S tart、Complete、Skipped、Abort、Checkpoint、Failed です。
eventtime	timestamp	`status` 列のタイムスタンプ。

列名	データ型	説明
sequence	integer	切り捨てられた `previous_state` 値のシーケンス番号。1 つの `previous_state` に含まれる文字数が 200 を超える場合、その値は追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
previous_state	character(200)	レコメンデーションを適用する前に、テーブルの以前のディストリビューションスタイルとソートキー。この値は 200 文字刻みの増分に切り捨てられます。

`status` 列値の例は次のとおりです。

- スキップ: テーブルが見つかりませんでした。
- スキップ: 推奨値は空です。
- スキップ: ソートキー推奨の適用が無効になっています。
- スキップ: 再試行がテーブルの最大制限を超えています。
- スキップ: テーブルの列が変更されました。
- 中止: このテーブルは AUTO ではありません。
- 中止: このテーブルは最近変換されました。
- 中止: このテーブルはテーブルサイズのしきい値を超えています。
- 中止: このテーブルはすでに推奨されているスタイルです。
- チェックポイント: 進捗状況 *21.9963*%。

サンプルクエリ

次の例では、結果の行に Amazon Redshift が実行したアクションが表示されます。

```
select table_id, type, status, eventtime, sequence, previous_state
from SVL_AUTO_WORKER_ACTION;
```

```
 table_id |   type    |                             status                              |
 eventtime              | sequence | previous_state
```

```
----------+---------+-----------------------------------------------------
+----------------------------+----------+----------------
   118082 | sortkey | Start                                                | 2020-08-22
 19:42:20.727049 | 0          |
   118078 | sortkey | Start                                                | 2020-08-22
 19:43:54.728819 | 0          |
   118082 | sortkey | Start                                                | 2020-08-22
 19:42:52.690264 | 0          |
   118072 | sortkey | Start                                                | 2020-08-22
 19:44:14.793572 | 0          |
   118082 | sortkey | Failed                                               | 2020-08-22
 19:42:20.728917 | 0          |
   118078 | sortkey | Complete                                             | 2020-08-22
 19:43:54.792705 | 0          | SORTKEY: None;
   118086 | sortkey | Complete                                             | 2020-08-22
 19:42:00.72635  | 0          | SORTKEY: None;
   118082 | sortkey | Complete                                             | 2020-08-22
 19:43:34.728144 | 0          | SORTKEY: None;
   118072 | sortkey | Skipped:Retry exceeds the maximum limit for a table. | 2020-08-22
 19:44:46.706155 | 0          |
   118086 | sortkey | Start                                                | 2020-08-22
 19:42:00.685255 | 0          |
   118082 | sortkey | Start                                                | 2020-08-22
 19:43:34.69531  | 0          |
   118072 | sortkey | Start                                                | 2020-08-22
 19:44:46.703331 | 0          |
   118082 | sortkey | Checkpoint: progress 14.755079%                      | 2020-08-22
 19:42:52.692828 | 0          |
   118072 | sortkey | Failed                                               | 2020-08-22
 19:44:14.796071 | 0          |
   116723 | sortkey | Abort:This table is not AUTO.                        | 2020-10-28
 05:12:58.479233 | 0          |
   110203 | distkey | Abort:This table is not AUTO.                        | 2020-10-28
 05:45:54.67259  | 0          |
```

SVL_COMPILE

クエリの各クエリセグメントのコンパイル時間と位置を記録します。

SVL_COMPILE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> SVL_COMPILE には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_QUERY_HISTORY を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

SVCS_COMPILE の詳細については、「SVCS_COMPILE」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントに関連付けられるプロセス ID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
segment	integer	コンパイルするクエリセグメント。
locus	integer	セグメントが実行される場所。コンピューティングノード上にある場合は **1**、リーダーノード上にある場合は **2**。
starttime	timestamp	UTC で表されたコンパイルの開始時間。
endtime	timestamp	UTC で表されたコンパイルの終了時刻。
compile	integer	コンパイルが再利用された場合は `0`、セグメントがコンパイルされた場合は **1**。

サンプルクエリ

この例では、クエリ 35878 と 35879 が同じ SQL ステートメントを実行します。クエリ 35878 の compile 列では 4 つのクエリセグメントに対して 1 が表示されており、それらのセグメントがコンパイルされたことを示しています。クエリ 35879 の compile 列ではすべてのセグメントについて 0 が表示されており、セグメントを再びコンパイルする必要がなかったことを示しています。

```
select userid, xid,  pid, query, segment, locus,
datediff(ms, starttime, endtime) as duration, compile
from svl_compile
where query = 35878 or query = 35879
order by query, segment;

 userid |  xid   |  pid  | query | segment | locus | duration | compile
--------+--------+-------+-------+---------+-------+----------+---------
    100 | 112780 | 23028 | 35878 |       0 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       1 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       2 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       3 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       4 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       5 |     1 |        0 |       0
    100 | 112780 | 23028 | 35878 |       6 |     1 |     1380 |       1
    100 | 112780 | 23028 | 35878 |       7 |     1 |     1085 |       1
    100 | 112780 | 23028 | 35878 |       8 |     1 |     1197 |       1
    100 | 112780 | 23028 | 35878 |       9 |     2 |      905 |       1
    100 | 112782 | 23028 | 35879 |       0 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       1 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       2 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       3 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       4 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       5 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       6 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       7 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       8 |     1 |        0 |       0
    100 | 112782 | 23028 | 35879 |       9 |     2 |        0 |       0
(20 rows)
```

SVL_DATASHARE_CHANGE_LOG

プロデューサークラスターとコンシューマークラスターの両方でデータ共有への変更を追跡するための統合ビューを記録します。

SVL_DATASHARE_CHANGE_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_DATASHARE_CHANGE_LOG でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	アクションを実行するユーザーの ID。
username	varCHAR(28)	アクションを実行するユーザーの名前。
pid	integer	プロセスの ID。
xid	bigint	トランザクションの ID。
share_id	integer	影響を受けるデータ共有の ID。
share_name	varCHAR(28)	データ共有の名前。
source_database_id	integer	データ共有が属するデータベースの ID。
source_database_name	varCHAR(28)	データ共有が属するデータベースの名前。
consumer_database_id	integer	データ共有からインポートされたデータベースの ID。

列名	データ型	説明
consumer_database_name	varCHAR(28)	データ共有からインポートされたデータベースの名前。
arn	varchar(192)	インポートされたデータベースをバッキングするリソースの ARN。
recordtime	timestamp	アクションのタイムスタンプ。
action	varCHAR(28)	実行されているアクション。可能な値は、CREATE DATASHARE、DROP DATASHARE、GRANT ALTER、REVOKE ALTER、GRANT SHARE、REVOKE SHARE、ALTER ADD、ALTER REMOVE、ALTER SET、GRANT USAGE、REVOKE USAGE、CREATE DATABASE、共有データベースでの GRANT または REVOAKE USAGE、DROP SHARED DATABASE、ALTER SHARED DATABASE です。
status	integer	アクションのステータス。可能な値は、SUCCESS コードと ERROR-ERROR CODE です。
share_object_type	varCHAR(64)	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトのタイプ。使用可能な値は、スキーマ、テーブル、列、関数、およびビューです。こちらは、プロデューサクラスターのフィールドです。
share_object_id	integer	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトの ID。こちらは、プロデューサクラスターのフィールドです。
share_object_name	varCHAR(28)	データ共有に追加またはデータ共有から削除されたデータベースオブジェクトの名前。こちらは、プロデューサクラスターのフィールドです。
target_user_type	varCHAR(6)	権限が付与されたユーザーまたはグループのタイプ。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。

列名	データ型	説明
target_userid	integer	権限が付与されたユーザーまたはグループの ID。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。
target_username	varCHAR(128)	権限が付与されたユーザーまたはグループの名前。こちらは、プロデューサークラスターとコンシューマークラスターの両方のフィールドです。
consumer_account	varCHAR(16)	データコンシューマーのアカウント ID。こちらは、プロデューサクラスターのフィールドです。
consumer_namespace	varCHAR(64)	データコンシューマーアカウントの名前空間。こちらは、プロデューサクラスターのフィールドです。
producer_account	varCHAR(16)	データ共有が属するプロデューサアカウントのアカウント ID。こちらは、コンシューマークラスタークラスターのフィールドです。
producer_namespace	varCHAR(64)	データ共有が属する製品アカウントの名前空間。こちらは、コンシューマークラスタークラスターのフィールドです。
attribute_name	varCHAR(64)	データ共有データベースまたは共有データベースの属性名。
attribute_value	varCHAR(128)	データ共有データベースまたは共有データベースの属性値。
メッセージ	varCHAR(512)	アクションが失敗したときのエラーメッセージ。

サンプルクエリ

次の例は、SVL_DATACHARE_CHANGE_LOG ビューを示しています。

```
SELECT DISTINCT action
FROM svl_datashare_change_log
WHERE share_object_name LIKE 'tickit%';
```

```
        action
-----------------------
 "ALTER DATASHARE ADD"
```

SVL_DATASHARE_CROSS_REGION_USAGE

SVL_DATASHARE_CROSS_REGION_USAGE ビューを使用して、クロスリージョンデータ共有クエリによって発生したクロスリージョンデータ転送使用量の概要を取得します。SVL_DATASHARE_CROSS_REGION_USAGE はセグメントレベルで詳細を集計します。

SVL_DATASHARE_CROSS_REGION_USAGE はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_DATASHARE_CROSS_REGION_USAGE でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
query	integer	クエリの ID。この値を使用して他のシステムテーブルおよびビューを結合します。
segment	bigint	セグメントの番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
start_time	時系	データ転送が開始される UTC の時刻。
end_time	時系	データ転送が終了した UTC の時刻。
transferred_data	bigint	プロデューサーリージョンからコンシューマーリージョンに転送されたデータのバイト数。
source_region	char(25)	クエリのデータ転送元であるプロデューサーリージョン。

列名	データ型	説明
recordtime	timestamp	アクションが記録される時刻。

サンプルクエリ

次の例は、SVL_DATASHARE_CROSS_REGION_USAGE ビューを示しています。

```
SELECT query, segment, transferred_data, source_region
from svl_datashare_cross_region_usage
where query = pg_last_query_id()
order by query,segment;

  query | segment | transferred_data | source_region
--------+---------+------------------+---------------
 200048 |       2 |          4194304 |    us-west-1
 200048 |       2 |          4194304 |    us-east-2
```

SVL_DATASHARE_USAGE_CONSUMER

データ共有のアクティビティと使用状況を記録します。このビューは、コンシューマークラスターにのみ関連しています。

SVL_DATASHARE_USAGE_CONSUMER はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_DATASHARE_USAGE_CONSUMER でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	リクエストを発行しているユーザーの ID。

列名	データ型	説明
pid	integer	クエリを実行しているリーダープロセスの ID。
xid	bigint	現在のトランザクションのコンテキスト。
request_id	varCHAR(50)	リクエストされた API コールの一意の ID。
request_type	varCHAR(25)	プロデューサクラスターに対して行われたリクエストのタイプ。
transaction_uid	varCHAR(50)	トランザクションの一意の ID。
recordtime	timestamp	アクションが記録される時刻。
status	integer	リクエストされた API コールのステータス。
error	varCHAR(512)	エラーのメッセージ。

サンプルクエリ

次の例は、SVL_DATASHARE_USAGE_CONSUMER ビューを示しています。

```
SELECT request_type, status, trim(error) AS error
FROM svl_datashare_usage_consumer

  request_type   | status | error
----------------+--------+--------
 "GET RELATION" |   0    |
```

SVL_DATASHARE_USAGE_PRODUCER

データ共有のアクティビティと使用状況を記録します。このビューは、コンシューマークラスターにのみ関連しています。

SVL_DATASHARE_USAGE_PRODUCER はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_DATASHARE_USAGE_PRODUCER でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
share_id	integer	データ共有のオブジェクト ID (OID)。
share_name	varCHAR(28)	データ共有の名前。
request_id	varCHAR(0)	リクエストされた API コールの一意の ID。
request_type	varCHAR(5)	プロデューサクラスターに対して行われたリクエストのタイプ。
object_type	varCHAR(4)	データ共有から共有されているオブジェクトのタイプ。使用可能な値は、スキーマ、テーブル、列、関数、およびビューです。
object_oid	integer	データ共有から共有されているオブジェクトの ID。
object_name	varCHAR(28)	データ共有から共有されているオブジェクトの名前。
consumer_account	varCHAR(6)	データ共有が共有されるコンシューマーアカウントのアカウント。
consumer_namespace	varCHAR(4)	データ共有が共有されるコンシューマーアカウントの名前空間。

列名	データ型	説明
consumer_transaction_uid	varCHAR(50)	コンシューマークラスター上のステートメントの一意のトランザクション ID。
recordtime	timestamp	アクションが記録される時刻。
status	integer	データ共有のステータス。
error	varCHAR(512)	エラーのメッセージ。
consumer_region	char(64)	コンシューマークラスターが属するリージョン。

サンプルクエリ

次の例は、SVL_DATACHARE_USAGE_PRODUCER ビューを示しています。

```
SELECT DISTINCT request_type
FROM svl_datashare_usage_producer
WHERE object_name LIKE 'tickit%';

   request_type
 ------------------
   "GET RELATION"
```

SVL_FEDERATED_QUERY

SVL_FEDERATED_QUERY ビューは、フェデレーテッドクエリの呼び出しに関する情報を表示するために使用します。

SVL_FEDERATED_QUERY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使い

やすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	クエリを実行しているユーザーの ID。
xid	bigint	トランザクション ID。
pid	integer	クエリを実行しているリーダープロセスの ID。
query	integer	フェデレーテッド呼び出しのクエリ ID。
sourcetype	character (32)	フェデレーションコールのソースタイプ (「PG」など)。
recordtime	timestamp	クエリがフェデレーションのために送信される時刻。UTC が使用されます。
querytext	character (4000)	実行のためにリモートの PostgreSQL エンジンに送信されるクエリ文字列。
num_rows	bigint	フェデレーションクエリによって返された行の数。
num_bytes	bigint	フェデレーションクエリによって返されたバイトの数。
duration	bigint	カーソル呼び出しから行の取得にかかった時間 (マイクロ秒単位)。フェデレーションク

列名	データ型	説明
		エリの実行と結果の取得にかかった時間。

サンプルクエリ

フェデレーテッドクエリの呼び出しに関する情報を表示するには、次のようなクエリを実行します。

```
select query, trim(sourcetype) as type, recordtime, trim(querytext) as "PG Subquery"
 from svl_federated_query where query = 4292;

 query | type |         recordtime         |                         pg subquery
-------+------+----------------------------
+---------------------------------------------------------------
  4292 | PG   | 2020-03-27 04:29:58.485126 | SELECT "level" FROM functional.employees
 WHERE ("level" >= 6)
(1 row)
```

SVL_MULTI_STATEMENT_VIOLATIONS

SVL_MULTI_STATEMENT_VIOLATIONS ビューを使用して、トランザクションブロックの制限に違反するシステムで実行されたすべての SQL コマンドの完全なレコードを取得します。

Amazon Redshift がトランザクションブロックまたはマルチステートメントのリクエスト内で制限する次の SQL コマンドのいずれかを実行すると、違反が発生します。

- CREATE DATABASE
- DROP DATABASE
- ALTER TABLE APPEND
- CREATE EXTERNAL TABLE
- DROP EXTERNAL TABLE
- RENAME EXTERNAL TABLE
- ALTER EXTERNAL TABLE
- CREATE TABLESPACE
- DROP TABLESPACE
- ライブラリを作成する

- DROP LIBRARY
- REBUILDCAT
- INDEXCAT
- REINDEX DATABASE
- VACUUM
- GRANT
- COPY

> **Note**
> このビューにエントリがある場合は、対応するアプリケーションおよび SQL スクリプトを変更します。アプリケーションコードを変更して、これらの制限された SQL コマンドの使用をトランザクションブロックの外に移動することをお勧めします。さらにサポートが必要な場合は、AWS サポートにお問い合わせください。

SVL_MULTI_STATEMENT_VIOLATIONS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	違反の原因となったユーザーの ID。
database	character(32)	ユーザーが接続されたデータベースの名前。
cmdname	character(20)	トランザクションブロックまたはマルチステートメントのリクエスト内で実行できないコマンドの名前。例えば、CREATE DATABASE、DROP DATABASE、ALTER TABLE APPEND、CREATE EXTERNAL TABLE、DROP EXTERNAL TABLE、RENAME

列名	データ型	説明
		EXTERNAL TABLE、ALTER EXTERNAL TABLE、CREATE LIBRARY、DROP LIBRARY、REBUILDCAT、INDEXCAT、REINDEX DATABASE、VACUUM、外部リソースでの GRANT、CLUSTER、COPY、CREATE TABLESPACE、および DROP TABLESPACE があります。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントのプロセス ID。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドは空になります。
starttime	timestamp	ステートメントの実行が開始された正確な時刻。秒の小数部の精度 (6 桁) を使用します (例: **2009-06-12 11:29:19.131358**)。
endtime	timestamp	ステートメントの実行が終了した正確な時刻。秒の小数部の精度 (6 桁) を使用します (例: **2009-06-12 11:29:19.193640**)。
sequence	integer	1 つのステートメントに含まれる文字数が 200 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
type	varchar(10)	SQL ステートメントのタイプ。**QUERY**、**DDL**、または **UTILITY** です。
text	character(200)	200 文字単位の SQL テキスト。このフィールドには、バックスラッシュ (\\) や改行 (\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

次のクエリは、違反がある複数のステートメントを返します。

```
select * from svl_multi_statement_violations order by starttime asc;

userid | database | cmdname | xid | pid | label | starttime | endtime | sequence | type
 | text
==============================================================================================================================
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ********* | ******* | 0 | DDL |
 create table c(b int);
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ********* | ******* | 0 | UTILITY |
 create database b;
1 | dev | CREATE DATABASE | 1034 | 5729 |label1 | ********* | ******* | 0 | UTILITY |
 COMMIT
...
```

SVL_MV_REFRESH_STATUS

SVL_MV_REFRESH_STATUS ビューには、マテリアライズドビューの更新作業用の行が含まれています。

マテリアライズドビューの詳細については、「Amazon Redshift でのマテリアライズドビュー」を参照してください。

SVL_MV_REFRESH_STATUS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_MV_REFRESH_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
db_name	char(128)	マテリアライズドビューを含むデータベース。
userid	bigint	更新を実行したユーザーの ID。

列名	データ型	説明
schema_name	char(128)	マテリアライズドビューのスキーマ。
mv_name	char(128)	マテリアライズドビューの名前。
xid	bigint	更新トランザクションの ID。
starttime	timestamp	更新の開始時間。
endtime	timestamp	更新の終了時間。

列名	データ型	説明
status	text	更新のステータス。値の例は次のとおりです。 • MV の段階的な更新を実施しました ストリーミング用のマテリアライズドビューの場合、メッセージには、レコード数に関する追加の限定詞が含まれている場合があります。これには以下が含まれます。 • ストリームは新しいデータを返しませんでした — レコードは取得されませんでした。 • ストリームから受信したすべてのレコードがスキップされました — レコードは取得されましたが、エラーによりすべてスキップされました。 • 一部のストリームレコードがスキップされました — レコードは取得されましたが、エラーにより一部のレコードがスキップされました。 限定詞がない場合は、少なくとも 1 つのレコードが取得され、すべてのレコードがマテリアライズドビューに表示されます。以下は使われる可能性のあるもう 1 つの限定詞です。 • ストリームにはさらに多くのデータが含まれている可能性があります — Amazon Redshift がこれ以上使用するレコードがないと判断する前に更新が終了しました。ストリームは最新のものである可能性がありますが、Amazon Redshift によって確認されていません。 • MV の再計算を正常に実施しました • 有効なトランザクションまで、一部更新された MV を段階的に更新しました

列名	データ型	説明
		• MV は既に更新されています • 更新に失敗しました。ベーステーブルの列の名前が変更されました • 更新に失敗しました。ベーステーブルの列のタイプが変更されました • 更新に失敗しました。ベーステーブル名が変更されました • 内部エラーのため、更新できませんでした • 更新に失敗しました。ベーステーブルの列が削除されました • 更新に失敗しました。MV のスキーマ名が変更されました • 更新に失敗しました。MV が見つかりませんでした • ユーザーのワークロードが過大なため、自動更新が中止されました • 更新に失敗しました。直列化可能分離違反
refresh_type	char(32)	更新タイプの定義。値の例には、[Manual] (手動) および [Auto] (自動) が含まれます。

サンプルクエリ

マテリアライズドビューの更新ステータスを表示するには、次のクエリを実行します。

```
select * from svl_mv_refresh_status;
```

このクエリは、次のサンプル出力を返します。

```
db_name | userid |   schema    |  name    |  xid  |            starttime            |
      endtime              |                          status                          |
refresh_type
```

```
---------+--------+-----------+---------+-------+----------------------------
+---------------------------+--------------------------------------------------------
+-----------------
 dev     |    169 | mv_schema | mv_test |  6640 | 2020-02-14 02:26:53.497935 |
 2020-02-14 02:26:53.556156 | Refresh successfully recomputed MV from scratch     |
 Manual
 dev     |    166 | mv_schema | mv_test |  6517 | 2020-02-14 02:26:39.287438 |
 2020-02-14 02:26:39.349539 | Refresh successfully updated MV incrementally       |
 Auto
 dev     |    162 | mv_schema | mv_test |  6388 | 2020-02-14 02:26:27.863426 |
 2020-02-14 02:26:27.918307 | Refresh successfully recomputed MV from scratch     |
 Manual
 dev     |    161 | mv_schema | mv_test |  6323 | 2020-02-14 02:26:20.020717 |
 2020-02-14 02:26:20.080002 | Refresh successfully updated MV incrementally       |
 Auto
 dev     |    161 | mv_schema | mv_test |  6301 | 2020-02-14 02:26:05.796146 |
 2020-02-14 02:26:07.853986 | Refresh successfully recomputed MV from scratch     |
 Manual
 dev     |    153 | mv_schema | mv_test |  6024 | 2020-02-14 02:25:18.762335 |
 2020-02-14 02:25:20.043462 | MV was already updated                              |
 Manual
 dev     |    143 | mv_schema | mv_test |  5557 | 2020-02-14 02:24:23.100601 |
 2020-02-14 02:24:23.100633 | MV was already updated                              |
 Manual
 dev     |    141 | mv_schema | mv_test |  5447 | 2020-02-14 02:23:54.102837 |
 2020-02-14 02:24:00.310166 | Refresh successfully updated MV incrementally       |
 Auto
 dev     |      1 | mv_schema | mv_test |  5329 | 2020-02-14 02:22:26.328481 |
 2020-02-14 02:22:28.369217 | Refresh successfully recomputed MV from scratch     |
 Auto
 dev     |    138 | mv_schema | mv_test |  5290 | 2020-02-14 02:21:56.885093 |
 2020-02-14 02:21:56.885098 | Refresh failed. MV was not found                    |
 Manual
```

SVL_QERROR

SVL_QERROR ビューは廃止されました。

SVL_QLOG

SVL_QLOG ビューにはデータベースに対して実行されたすべてのクエリのログが含まれます。

Amazon Redshift は STL_QUERY テーブルから取得した情報の読み込み可能なサブセットとして SVL_QLOG ビューを作成します。このテーブルを使用して最近実行されたクエリのクエリ ID を検索したり、クエリが完了するまでにかかった時間を確認します。

SVL_QLOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。この ID を使用して、他のさまざまなシステムテーブルやビューを結合できます。
xid	bigint	トランザクション ID。
pid	integer	クエリに関連付けられたプロセス ID。
starttime	timestamp	ステートメントの実行が開始された正確な時刻。秒の小数部の精度 (6 桁) を使用します (例: **2009-06-12 11:29:19.131358**)。
endtime	timestamp	ステートメントの実行が終了した正確な時刻。秒の小数部の精度 (6 桁) を使用します (例: **2009-06-12 11:29:19.193640**)。
elapsed	bigint	クエリの実行にかかった時間の長さ (マイクロ秒)。
aborted	integer	クエリがシステムによって停止されたかユーザーによってキャンセルされた場合、この列は **1** になります。クエリが最後まで実行された場合、この列は **0** になります。ワークロードを管理する目的でキャンセルされ、その後再開されたクエリでも、この列の値が **1** になります。

列名	データ型	説明
ラベル	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドの値は `default` になります。
substring	character(60)	切り捨てられたクエリのテキスト。
source_query	integer	クエリで結果のキャッシュが使用された場合、キャッシュされた結果のソースとなったクエリのクエリ ID。結果のキャッシュが使用されていない場合、このフィールドの値は NULL です。
concurrency_scaling_status_txt	text	クエリがメインクラスター、または同時実行スケーリングクラスターのどちらで実行されたかについての説明。
from_sp_call	integer	クエリがストアドプロシージャから呼び出された場合は、プロシージャ呼び出しのクエリ ID。クエリがストアドプロシージャの一部として実行されなかった場合、このフィールドは NULL です。

サンプルクエリ

以下の例では、`userid = 100` のユーザーによって実行された最近 5 つのデータベースクエリのクエリ ID、実行時間、および切り捨てられたクエリテキストが返されます。

```
select query, pid, elapsed, substring from svl_qlog
where userid = 100
order by starttime desc
limit 5;

 query  |  pid  | elapsed  |           substring
--------+-------+----------+-----------------------------------------------
 187752 | 18921 | 18465685 | select query, elapsed, substring from svl_...
 204168 |  5117 |    59603 | insert into testtable values (100);
```

```
 187561 | 17046 |  1003052 | select * from pg_table_def where tablename...
 187549 | 17046 |  1108584 | select * from STV_WLM_SERVICE_CLASS_CONFIG
 187468 | 17046 |  5670661 | select * from pg_table_def where schemaname...
(5 rows)
```

次の例では、キャンセルされたクエリ (**aborted=1**) の SQL スクリプト名 (LABEL 列) と経過時間を返します。

```
select query, elapsed, trim(label) querylabel
from svl_qlog where aborted=1;

 query | elapsed  |       querylabel
-------+----------+-------------------------
    16 |  6935292 | alltickittablesjoin.sql
(1 row)
```

SVL_QUERY_METRICS

SVL_QUERY_METRICS は、完了したクエリのメトリクスを表示します。このビューは STL_QUERY_METRICS システムテーブルから取得されます。このビューの値は、クエリのモニタリングルールを定義するしきい値を決定する際に役立ちます。詳細については、「WLM クエリモニタリングルール」を参照してください。

SVL_QUERY_METRICS はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したクエリを実行したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。

列名	データ型	説明
service_class	integer	WLM クエリキュー (サービスクラス) の ID。クエリキューは WLM 設定で定義されます。メトリクスはユーザー定義のキューについてのみレポートされます。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
dimension	varCHAR(24)	メトリクスが報告されたディメンション。想定される値は、query、segment、step です。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。セグメント値が 0 である場合、メトリクスセグメントの値はクエリレベルまでロールアップされます。
step	integer	実行されたステップのタイプの ID。ステップタイプの説明は、`step_label` 列に表示されます。
step_label	varCHAR(30)	実行されたステップのタイプ。
query_cpu_time	bigint	クエリに使用される CPU 時間 (秒)。CPU 時間は、クエリの実行時間とは異なります。
query_blocks_read	bigint	クエリによって読み取られた 1 MB ブロックの数。
query_execution_time	bigint	経過したクエリ実行時間 (秒)。キューでの待機時間は実行時間に含まれません。キューに追加される時間については、「`query_queue_time` 」を参照してください。
query_cpu_usage_percent	bigint	クエリが使用する CPU 容量の割合。
query_temp_blocks_to_disk	bigint	クエリが中間結果の書き込みに使用するディスク容量 (MB)。

列名	データ型	説明
segment_e xecution_time	bigint	単一セグメントで経過した実行時間 (秒)。
cpu_skew	numeric(3 8,2)	任意のスライスの最大 CPU 使用量とすべてのスライスの平均 CPU 使用量の比率。このメトリクスはセグメントレベルで定義されます。
io_skew	numeric(3 8,2)	任意のスライスの最大ブロック読み取り (I/O) とすべてのスライスの平均ブロック読み取りの比率。
scan_row_count	bigint	スキャンステップの行数。行数は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前およびユーザー定義のクエリフィルタが適用される前に出力された合計行数を表します。
join_row_count	bigint	結合ステップで処理された行数。
nested_lo op_join_r ow_count	bigint	ネストしたループ結合の行数。
return_ro w_count	bigint	クエリによって返された行の数。
spectrum_ scan_row_count	bigint	Amazon S3 で Amazon Redshift Spectrum によってスキャンされた行数。
spectrum_ scan_size_mb	bigint	Amazon S3 の Amazon Redshift Spectrum によってスキャンされたデータ量 (MB)。
query_que ue_time	bigint	クエリがキューに追加される時間（秒単位）。

SVL_QUERY_METRICS_SUMMARY

SVL_QUERY_METRICS_SUMMARY ビューは、完了したクエリのメトリクスの最大値を示します。このビューは STL_QUERY_METRICS システムテーブルから取得されます。このビューの値は、ク

エリのモニタリングルールを定義するしきい値を決定する際に役立ちます。Amazon Redshift のクエリモニタリングのルールとメトリクスの詳細については、「WLM クエリモニタリングルール」を参照してください。

SVL_QUERY_METRICS_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したクエリを実行したユーザーの ID。
query	integer	クエリ ID。クエリ列は、他の各種システムテーブルおよびビューを結合するために使用できます。
service_class	integer	WLM クエリキュー (サービスクラス) の ID。クエリキューは WLM 設定で定義されます。メトリクスはユーザー定義のキューについてのみレポートされます。サービスクラス ID のリストについては、「WLM サービスクラス ID」を参照してください。
query_cpu_time	bigint	クエリに使用される CPU 時間 (秒)。CPU 時間は、クエリの実行時間とは異なります。
query_blocks_read	bigint	クエリによって読み取られた 1 MB ブロックの数。
query_execution_time	bigint	経過したクエリ実行時間 (秒)。キューでの待機時間は実行時間に含まれません。
query_cpu_usage_percent	numeric(38,2)	クエリが使用する CPU 容量の割合。

列名	データ型	説明
query_temp_blocks_to_disk	bigint	クエリが中間結果の書き込みに使用するディスク容量 (MB)。
segment_execution_time	bigint	単一セグメントで経過した実行時間 (秒)。
cpu_skew	numeric(38,2)	任意のスライスの最大 CPU 使用量とすべてのスライスの平均 CPU 使用量の比率。このメトリクスはセグメントレベルで定義されます。
io_skew	numeric(38,2)	任意のスライスの最大ブロック読み取り (I/O) とすべてのスライスの平均ブロック読み取りの比率。
scan_row_count	bigint	スキャンステップの行数。行数は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前およびユーザー定義のクエリフィルタが適用される前に出力された合計行数を表します。
join_row_count	bigint	結合ステップで処理された行数。
nested_loop_join_row_count	bigint	ネストしたループ結合の行数。
return_row_count	bigint	クエリによって返された行の数。
spectrum_scan_row_count	bigint	Amazon S3 で Amazon Redshift Spectrum によってスキャンされた行数。
spectrum_scan_size_mb	bigint	Amazon S3 の Amazon Redshift Spectrum によってスキャンされたデータ量 (MB)。
query_queue_time	bigint	クエリがキューに追加される時間（秒単位）。

SVL_QUERY_QUEUE_INFO

ワークロード管理 (WLM) クエリキューまたはコミットキューで時間がかかったクエリの詳細が示されます。

SVL_QUERY_QUEUE_INFO ビューは、システムによって実行されたクエリをフィルタリングし、ユーザーによって実行されたクエリのみを表示します。

SVL_QUERY_QUEUE_INFO ビューでは、STL_QUERY、STL_WLM_QUERY、および STL_COMMIT_STATS システムテーブルからの情報の概要が表示されます。

SVL_QUERY_QUEUE_INFO はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
データベース	text	クエリが発行されたときにユーザーが接続されたデータベースの名前。
query	integer	クエリ ID。
xid	bigint	トランザクション ID。
userid	integer	クエリを生成したユーザーの ID。
querytxt	text	クエリテキストの最初の 100 文字。
queue_start_time	timestamp	UTC で表された、クエリが WLM キューに入った時間。
exec_start_time	timestamp	UTC で表されたクエリ実行の開始時間。
service_class	integer	サービスクラスの ID。サービスクラスは、WLM 設定ファイル内で定義されます。
slots	integer	WLM クエリスロットの数。

列名	データ型	説明
queue_elapsed	bigint	WLM キューでのクエリの待機時間 (秒単位)。
exec_elapsed	bigint	クエリの実行にかかった時間 (秒単位)。
wlm_total_elapsed	bigint	WLM キューでクエリにかかった時間 (queue_elapsed) と、クエリの実行にかかった時間 (exec_elapsed)。
commit_queue_elapsed	bigint	コミットキューでのクエリの待機時間 (秒単位)。
commit_exec_time	bigint	コミット操作でのクエリの待機時間 (秒単位)。
service_class_name	character(64)	サービスクラスの名前。

サンプルクエリ

次の例は、WLM キューでクエリにかかった時間を示しています。

```
select query, service_class, queue_elapsed, exec_elapsed, wlm_total_elapsed
from svl_query_queue_info
where wlm_total_elapsed > 0;

  query   | service_class | queue_elapsed | exec_elapsed | wlm_total_elapsed
---------+---------------+---------------+--------------+-------------------
 2742669 |             6 |             2 |          916 |                918
 2742668 |             6 |             4 |          197 |                201
(2 rows)
```

SVL_QUERY_REPORT

Amazon Redshift は、Amazon Redshift の STL システムテーブルのいくつかを UNION (結合) したものから SVL_QUERY_REPORT ビューを作成し、完了済みのクエリステップに関する情報を提供します。

このビューは、完了済みのクエリに関する情報を、スライスごと、およびステップごとに分類します。これは、Amazon Redshift クラスターでのノードやスライスの問題のトラブルシューティングに役立ちます。

SVL_QUERY_REPORT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
slice	integer	ステップが実行されたデータスライス。
segment	integer	セグメント番号。 複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。
step	integer	完了されたクエリステップ。
start_time	timestamp	セグメントの実行が開始された正確な時間 (UTC)。小数秒は 6 桁まで示されます。例: **`2012-12-12 11:29:19.131358`**
end_time	timestamp	セグメントの実行が終了した正確な時間 (UTC)。小数秒は 6 桁まで示されます。例: **`2012-12-12 11:29:19.131467`**
elapsed_time	bigint	セグメントの実行にかかった時間 (マイクロ秒)。

列名	データ型	説明
rows	bigint	ステップが生成した行の数 (スライスあたり)。この数値はステップを実行した結果生成されたスライスの行数を表すもので、ステップが受け取った、または処理した行の数ではありません。すなわち、この数値はステップの実行後、次のステップに渡された行の数です。
bytes	bigint	ステップが生成したバイトの数 (スライスあたり)。
label	char(256)	クエリステップ名および該当する場合はテーブル ID とテーブル名 (`scan tbl=100448 name =user` など) からなるステップラベル。3 桁のテーブル ID は通常、一時テーブルのスキャンを照会します。`tbl=0` は、通常、定数値のスキャンを指します。
is_diskbased	character (1)	クエリのこのステップがディスクベースのオペレーションとして実行されたかどうか: true (**t**) または false (**f**)。ハッシュ、ソート、集計といった特定のステップのみがディスクベースで実行できます。ステップの多くのタイプは、常にメモリ内で実行されます。
workmem	bigint	クエリステップに割り当てられた作業メモリの量 (バイト単位)。この値は実行中に使用するために割り当てられた query_working_mem しきい値です。実際に使用されたメモリの量ではありません。
is_rrscan	character (1)	true (**t**) の場合は、ステップで範囲限定スキャンが使用されたことを示します。
is_delayed_scan	character (1)	true (**t**) の場合は、ステップで遅延スキャンが使用されたことを示します。
rows_pre_filter	bigint	永続テーブルのスキャンの場合は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前でユーザー定義のクエリフィルタが適用される前に出力された合計行数。

サンプルクエリ

次のクエリは、クエリ ID 279 のクエリを実行したときに返される行のデータの偏りを示します。このクエリを使用して、データベースのデータがデータウェアハウスクラスターのスライス全体に均等に分散されているかどうかを確認します。

```
select query, segment, step, max(rows), min(rows),
case when sum(rows) > 0
then ((cast(max(rows) -min(rows) as float)*count(rows))/sum(rows))
else 0 end
from svl_query_report
where query = 279
group by query, segment, step
order by segment, step;
```

このクエリにより返されるデータは、以下のサンプルの出力のようになります。

```
query | segment | step |   max    |   min    |         case
------+---------+------+----------+----------+----------------------
279 |       0 |    0 | 19721687 | 19721687 |                     0
279 |       0 |    1 | 19721687 | 19721687 |                     0
279 |       1 |    0 |   986085 |   986084 | 1.01411202804304e-06
279 |       1 |    1 |   986085 |   986084 | 1.01411202804304e-06
279 |       1 |    4 |   986085 |   986084 | 1.01411202804304e-06
279 |       2 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       2 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    2 |  1775517 |   788460 |     1.00098637606408
279 |       3 |    3 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    0 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    1 |  1775517 |   788460 |     1.00098637606408
279 |       4 |    2 |        1 |        1 |                     0
279 |       5 |    0 |        1 |        1 |                     0
279 |       5 |    1 |        1 |        1 |                     0
279 |       6 |    0 |       20 |       20 |                     0
279 |       6 |    1 |        1 |        1 |                     0
279 |       7 |    0 |        1 |        1 |                     0
279 |       7 |    1 |        0 |        0 |                     0
(19 rows)
```

SVL_QUERY_SUMMARY

SVL_QUERY_SUMMARY ビューを使用して、クエリの実行についての全般的な情報を確認します。

SVL_QUERY_SUMMARY ビューには SVL_QUERY_REPORT ビューからのデータのサブセットが含まれます。SVL_QUERY_SUMMARY の情報はすべてのノードからの情報の集計であることに注意してください。

> **Note**
>
> SVL_QUERY_SUMMARY ビューには、Amazon Redshift によって実行されたクエリに関する情報のみが含まれます。その他のユーティリティや DDL コマンドの情報は含まれません。Amazon Redshift によって実行されたすべてのステートメントの完全なリストと、DDL およびユーティリティコマンドを含めた情報については、SVL_STATEMENTTEXT ビューをクエリできます。

SVL_QUERY_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

SVCS_QUERY_SUMMARY の詳細については、「SVCS_QUERY_SUMMARY」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
query	integer	クエリ ID。他の各種システムテーブルおよびビューを結合するために使用できます。
stm	integer	ストリーム。クエリの並行セグメントのセット。クエリには 1 つ以上のストリームがあります。
seg	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。
step	integer	実行されたクエリステップ。

列名	データ型	説明
maxtime	bigint	ステップを実行する最大時間 (マイクロ秒)。
avgtime	bigint	ステップを実行する平均時間 (マイクロ秒)。
rows	bigint	クエリステップに含まれるデータ行の数。
バイト	bigint	クエリステップに含まれるデータバイトの数。
rate_row	double precision	行ごとのクエリ実行率。
rate_byte	double precision	バイトごとのクエリ実行率。
label	text	ステップラベル。これは、クエリステップ名と、該当する場合はテーブル ID とテーブル名 (例: scan tbl=100448 name =user) で構成されます。3 桁のテーブル ID は通常、一時テーブルのスキャンを照会します。 `tbl=0` は、通常、定数値のスキャンを指します。
is_diskbased	character(1)	クエリのこのステップが、クラスター内のいずれかのノードでディスクベースのオペレーションとして実行されたかどうか: true (**t**) または false (**f**)。ハッシュ、ソート、集計といった特定のステップのみがディスクベースで実行できます。ステップの多くのタイプは、常にメモリ内で実行されます。
workmem	bigint	クエリステップに割り当てられた作業メモリの量 (バイト単位)。
is_rrscan	character(1)	true (**t**) の場合は、ステップで範囲限定スキャンが使用されたことを示します。デフォルトは false (**f**) です。
is_delayed_scan	character(1)	true (**t**) の場合は、ステップで遅延スキャンが使用されたことを示します。デフォルトは false (**f**) です。
rows_pre_filter	bigint	永続テーブルのスキャンの場合、削除対象としてマークされた行をフィルタリングする前に出力された合計行数 (非実体の行)。

サンプルクエリ

クエリステップの処理情報を表示する

次のクエリは、クエリ 87 の各ステップの基本的な処理情報を示します。

```
select query, stm, seg, step, rows, bytes
from svl_query_summary
where query = 87
order by query, seg, step;
```

このクエリは次のサンプル出力で示されているように、クエリ 87 についての処理情報を取得します。

```
 query | stm | seg | step |  rows  |  bytes
-------+-----+-----+------+--------+---------
87     |   0 |   0 |    0 |     90 |    1890
87     |   0 |   0 |    2 |     90 |     360
87     |   0 |   1 |    0 |     90 |     360
87     |   0 |   1 |    2 |     90 |    1440
87     |   1 |   2 |    0 | 210494 | 4209880
87     |   1 |   2 |    3 |  89500 |       0
87     |   1 |   2 |    6 |      4 |      96
87     |   2 |   3 |    0 |      4 |      96
87     |   2 |   3 |    1 |      4 |      96
87     |   2 |   4 |    0 |      4 |      96
87     |   2 |   4 |    1 |      1 |      24
87     |   3 |   5 |    0 |      1 |      24
87     |   3 |   5 |    4 |      0 |       0
(13 rows)
```

クエリステップがディスクに書き出されたかどうかを確認する

次のクエリは、クエリ ID 1025 (SVL_QLOG を参照して、クエリのクエリ ID の取得方法を確認してください) のクエリのステップがディスクに書き出されたかどうか、またはクエリが完全にインメモリで実行されたかどうかを示します。

```
select query, step, rows, workmem, label, is_diskbased
from svl_query_summary
where query = 1025
order by workmem desc;
```

このクエリは、次のサンプル出力を返します。

```
query| step|  rows   |  workmem    |  label         | is_diskbased
-----+-----+--------+-----------+---------------+--------------
1025 |  0  |16000000|  141557760 |scan tbl=9     | f
1025 |  2  |16000000|  135266304 |hash tbl=142   | t
1025 |  0  |16000000|  128974848 |scan tbl=116536| f
1025 |  2  |16000000|  122683392 |dist           | f
(4 rows)
```

IS_DISKBASED の値をスキャンすることで、ディスクに書き出されたクエリステップを確認できます。クエリ 1025 のハッシュステップはディスクで実行されました。ディスクで実行される可能性があるステップは、ハッシュ、集計、ソートステップを含みます。ディスクベースクエリのステップのみを表示するには、前述の例の SQL ステートメントに **and is_diskbased = 't'** 句を追加します。

SVL_RESTORE_ALTER_TABLE_PROGRESS

SVL_RESTORE_ALTER_TABLE_PROGRESS を使用して、RA3 ノードへの従来のサイズ変更中のクラスター内の各テーブルについて、移行の進行状況をモニタリングします。サイズ変更操作中のデータ移行の履歴スループットをキャプチャします。RA3 ノードへの従来のサイズ変更について詳しくは、「従来のサイズ変更」を参照してください。

SVL_RESTORE_ALTER_TABLE_PROGRESS はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_RESTORE_LOG でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

> **Note**
>
> 進行状況が `100.00%` または `ABORTED` の行は 7 日後に削除されます。従来のサイズ変更中またはサイズ変更後に削除されたテーブルの行は、引き続き SVL_RESTORE_ALTER_TABLE_PROGRESS に表示されることがあります。

テーブルの列

列名	データ型	説明
tbl	integer	テーブルの ID。
progress	char(32)	テーブルの再分散の進捗状況。表示される値は 0.00% から 100.00% のパーセント値および ABORTED メッセージです。ABORTED は、message 列に表示される理由により、再分散が完了しなかったことを示します。
message	char(256)	テーブルの再分散の進捗状況に関連するメッセージ。

サンプルクエリ

次のクエリは、実行中とキューに登録されたクエリを返します。

```
select * from svl_restore_alter_table_progress;

tbl      | progress |                          message
--------+----------+------------------------------------------------------
105614  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105610  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105594  | 0.00%    | Table waiting for alter diststyle conversion.
105602  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105606  | ABORTED  | Abort:Table no longer contains the prior dist key column.
105598  | 100.00%  | Restored to distkey successfully.
```

SVL_S3LIST

SVL_S3LIST ビューを使用して、セグメントレベルで Amazon Redshift Spectrum クエリの詳細を確認します。

SVL_S3LIST はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> SVL_S3LIST には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
query	integer	クエリ ID。
segment	integer	セグメント番号。クエリは、複数のセグメントで構成されます。
node	integer	ノード番号。
slice	integer	特定のセグメントが実行されたデータスライス。
eventtime	timestamp	イベントが記録された時刻 (UTC)。
bucket	text	Amazon S3 バケット名。
プレフィックス	text	Amazon S3 バケットがある場所のプレフィックス。
recursive	char(1)	サブフォルダの Recursive Scan の有無。
retrieved_files	integer	表示されたファイルの数。
max_file_size	bigint	表示されたファイルの最大ファイルサイズ。
avg_file_size	double precision	表示されたファイルの平均ファイルサイズ。

列名	データ型	説明
generated_splits	integer	ファイル分割数。
avg_split_length	double precision	ファイル分割の平均サイズ (バイト)。
duration	bigint	ファイルの表示期間 (マイクロ秒)

サンプルクエリ

以下の例は、最後に実行されるクエリについて SVL_S3LIST をクエリします。

```
select *
from svl_s3list
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3LOG

SVL_S3LOG ビューを使用して、セグメントおよびノードスライスレベルで Amazon Redshift Spectrum クエリの詳細を確認します。

SVL_S3LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> SVL_S3LOG には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
pid	integer	プロセス ID。
query	integer	クエリ ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
step	integer	実行したクエリステップ。
node	integer	ノード番号。
slice	integer	特定のセグメントが実行されたデータスライス。
eventtime	timestamp	ステップの実行が開始された時間 (UTC)。
メッセージ	text	ログエントリのメッセージ。

サンプルクエリ

以下の例は、最後に実行されたクエリについて SVL_S3LOG をクエリします。

```
select *
from svl_s3log
where query = pg_last_query_id()
order by query,segment,slice;
```

SVL_S3PARTITION

SVL_S3PARTITION ビューを使用して、セグメントおよびノードスライスレベルで Amazon Redshift Spectrum パーティションの詳細を確認します。

SVL_S3PARTITION はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
>
> SVL_S3PARTITION には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
query	integer	クエリ ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
node	integer	ノード番号。
slice	integer	特定のセグメントが実行されたデータスライス。
starttime	タイムゾーンなしのタイムスタンプ	パーティション削除が開始した時刻 (UTC 時間)。
endtime	タイムゾーンなしのタイムスタンプ	パーティション削除が完了した時刻 (UTC 時間)。
duration	bigint	経過時間 (マイクロ秒)。
total_partitions	integer	合計パーティションの数。
qualified_partitions	integer	適格なパーティションの数。
assigned_partitions	integer	スライス上の割り当てられたパーティションの数。

列名	データ型	説明
割り当て	文字	割り当てのタイプ。

サンプルクエリ

以下の例は、最後に完了されたクエリに関するパーティションの詳細を取得します。

```
SELECT query, segment,
       MIN(starttime) AS starttime,
       MAX(endtime) AS endtime,
       datediff(ms,MIN(starttime),MAX(endtime)) AS dur_ms,
       MAX(total_partitions) AS total_partitions,
       MAX(qualified_partitions) AS qualified_partitions,
       MAX(assignment) as assignment_type
FROM svl_s3partition
WHERE query=pg_last_query_id()
GROUP BY query, segment
```

```

query | segment |           starttime           |           endtime           | dur_ms|
 total_partitions | qualified_partitions | assignment_type
------+---------+-------------------------------+-----------------------------+-------
+------------------+----------------------+----------------
99232 |       0 | 2018-04-17 22:43:50.201515    | 2018-04-17 22:43:54.674595  |   4473 |
       2526       |         334          | p
```

SVL_S3PARTITION_SUMMARY

SVL_S3PARTITION_SUMMARY ビューを使用して、セグメントレベルで Redshift Spectrum クエリパーティションの概要を取得します。

SVL_S3PARTITION_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

SVCS_S3PARTITION の詳細については、「SVCS_S3PARTITION_SUMMARY」を参照してください。

テーブルの列

列名	データ型	説明
query	integer	クエリ ID。この値を使用して、他の各種システムテーブルおよびビューを結合できます。
segment	integer	セグメント番号。クエリは、複数のセグメントで構成されます。
割り当て	char(1)	ノード間のパーティション割り当てのタイプ。
min_starttime	timestamp	パーティション処理が開始した時刻 (UTC 時間)。
max_endtime	timestamp	パーティション処理が完了した時刻 (UTC 時間)。
min_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの最小処理時間 (マイクロ秒)。
max_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの最大処理時間 (マイクロ秒)。
avg_duration	bigint	ノードでこのクエリを実行するのにかかるパーティションの平均処理時間 (マイクロ秒)。
total_partitions	integer	外部テーブルのパーティションの合計数。
qualified_partitions	integer	適格なパーティションの合計数。
min_assigned_partitions	integer	1 つのノードに割り当てられたパーティションの最小数。
max_assigned_partitions	integer	1 つのノードに割り当てられたパーティションの最大数。

列名	データ型	説明
avg_assigned_partitions	bigint	1 つのノードに割り当てられたパーティションの平均数。

サンプルクエリ

以下の例は、最後に完了されたクエリに関するパーティションスキャンの詳細を取得します。

```
select query, segment, assignment, min_starttime, max_endtime, min_duration,
 avg_duration
from svl_s3partition_summary
where query = pg_last_query_id()
order by query,segment;
```

SVL_S3QUERY

SVL_S3QUERY ビューを使用して、セグメントおよびノードスライスレベルで Amazon Redshift Spectrum クエリの詳細を確認します。

SVL_S3QUERY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

> **Note**
> SVL_S3QUERY には、メインクラスターで実行されるクエリのみが含まれます。同時実行スケーリングクラスターで実行されるクエリは含まれていません。メインクラスターおよび同時実行スケーリングクラスターの両方で実行されるクエリにアクセスするには、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL を使用することをお勧めします。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。

テーブルの列

列名	データ型	説明
userid	integer	指定のエントリを生成したユーザーの ID。
query	integer	クエリ ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
step	integer	実行したクエリステップ。
node	integer	ノード番号。
slice	integer	特定のセグメントが実行されたデータスライス。
starttime	timestamp	クエリの実行が開始された時間 (UTC)。
endtime	timestamp	クエリの実行が完了した時間 (UTC)。
elapsed	integer	経過時間 (マイクロ秒)。
external_table_name	char(136)	s3 スキャンステップの外部テーブル名の内部形式。
is_partitioned	char(1)	true (**t**) の場合、この列の値は外部テーブルがパーティション化されていることを示します。
is_rrscan	char(1)	true (**t**) の場合、この列の値は範囲限定スキャンが適用されたことを示します。
s3_scanned_rows	bigint	Amazon S3 からスキャンされ、Redshift Spectrum レイヤーに送信された行数。
s3_scanned_bytes	bigint	Amazon S3 からスキャンされ、Redshift Spectrum レイヤーに送信されたバイト数。

列名	データ型	説明
s3query_returned_rows	bigint	Redshift Spectrum レイヤーからクラスターに返された行数。
s3query_returned_bytes	bigint	Redshift Spectrum レイヤーからクラスターに返されたバイト数。
files	integer	このスライスのこの S3 で処理されたファイルの数。
splits	int	このスライスで処理された分割の数。例えば分割可能なデータファイルの容量が大きい場合 (約 512 MB を超えるデータファイルなど)、Redshift Spectrum はファイルを複数の S3 リクエストに分割し、並列処理を試みます。
total_split_size	bigint	このスライスで処理された分割の合計サイズ (単位: バイト)。
max_split_size	bigint	このスライスで処理された分割の最大サイズ (単位: バイト)。
total_retries	integer	処理されたファイルの再試行の総数。
max_retries	integer	個別の処理ファイルの再試行の最大数。
max_request_duration	integer	Redshift Spectrum の個別リクエストの最長時間 (マイクロ秒)。
avg_request_duration	double precision	Redshift Spectrum リクエストの平均時間 (マイクロ秒)。
max_request_parallelism	integer	この S3 スキャンステップのこのスライス上で未処理の Redshift Spectrum の最大数。

列名	データ型	説明
avg_request_parallelism	double precision	この S3 スキャンステップのこのスライス上の並列 Redshift Spectrum リクエストの平均数。

サンプルクエリ

以下の例は、最後に完了されたクエリに関するスキャンステップの詳細を取得します。

```
select query, segment, slice, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query
where query = pg_last_query_id()
order by query,segment,slice;
```

```
query | segment | slice | elapsed | s3_scanned_rows | s3_scanned_bytes |
 s3query_returned_rows | s3query_returned_bytes | files
------+---------+-------+---------+-----------------+------------------
+-----------------------+------------------------+------
 4587 |       2 |     0 |   67811 |               0 |                0 |
      0 |                       0 |     0
 4587 |       2 |     1 |  591568 |          172462 |         11260097 |
   8513 |                  170260 |     1
 4587 |       2 |     2 |  216849 |               0 |                0 |
      0 |                       0 |     0
 4587 |       2 |     3 |  216671 |               0 |                0 |
      0 |                       0 |     0
```

SVL_S3QUERY_SUMMARY

SVL_S3QUERY_SUMMARY ビューを使用して、システムで実行されたすべての Amazon Redshift Spectrum クエリ (S3 クエリ) 概要を取得します。SVL_S3QUERY_SUMMARY は、SVL_S3QUERY の詳細をセグメントレベルで集計します。

SVL_S3QUERY_SUMMARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_EXTERNAL_QUERY_DETAIL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

SVCS_S3QUERY_SUMMARY については、「SVCS_S3QUERY_SUMMARY」を参照してください。

テーブルの列

列名	データ型	説明
userid	integer	指定のエントリを生成したユーザーの ID。
query	integer	クエリ ID。この値を使用して、他の各種システムテーブルおよびビューを結合できます。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。
segment	integer	セグメント番号。複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。
step	integer	実行したクエリステップ。
starttime	timestamp	クエリの実行が開始された時間 (UTC)。
endtime	timestamp	クエリの完了した時間 (UTC)。
elapsed	integer	クエリの実行にかかった時間の長さ (マイクロ秒)。
aborted	integer	クエリがシステムによって停止されたかユーザーによってキャンセルされた場合、この列は **1** になります。クエリが最後まで実行された場合、この列は **0** になります。
external_table_name	char(136)	外部テーブルスキャンのテーブルの外部名の内部形式。

列名	データ型	説明
file_format	character(16)	外部テーブルデータのファイル形式。
is_partitioned	char(1)	true (**t**) の場合、この列の値は外部テーブルがパーティション化されていることを示します。
is_rrscan	char(1)	true (**t**) の場合、この列の値は範囲限定スキャンが適用されたことを示します。
is_nested	char(1)	true (**t**) の場合、この列の値は、ネストされた列のデータ型にアクセスされていることを示します。
s3_scanned_rows	bigint	Amazon S3 からスキャンされ、Redshift Spectrum レイヤーに送信された行数。
s3_scanned_bytes	bigint	Amazon S3 からスキャンされ Redshift Spectrum レイヤーに送信されたバイト数 (圧縮データに基づく)。
s3query_returned_rows	bigint	Redshift Spectrum レイヤーからクラスターに返された行数。
s3query_returned_bytes	bigint	Redshift Spectrum レイヤーからクラスターに返されたバイト数。Amazon Redshift に大量のデータが返されると、システムパフォーマンスに影響が及ぶ可能性があります。
files	integer	この Redshift Spectrum クエリで処理されたファイル数。ファイル数が少ないと、並列処理の利点は制限されます。
files_max	integer	1 つのスライスで処理されるファイルの最大数。
files_avg	integer	1 つのスライスで処理されるファイルの平均数。
splits	int	このセグメントで処理された分割の数。このスライスで処理された分割の数。例えば分割可能なデータファイルの容量が大きい場合 (約 512 MB を超えるデータファイルなど)、Redshift Spectrum はファイルを複数の S3 リクエストに分割し、並列処理を試みます。

列名	データ型	説明
splits_max	int	このスライスで処理された分割の最大数。
splits_avg	int	このスライスで処理された分割の平均数。
total_split_size	bigint	処理されたすべての分割の合計サイズ。
max_split_size	bigint	処理された分割の最大サイズ (単位: バイト)。
avg_split_size	bigint	処理された分割の平均サイズ (単位: バイト)。
total_retries	integer	個別の処理ファイルでの再試行の総数。
max_retries	integer	処理ファイルでの再試行の最大数。
max_request_duration	integer	個別ファイルリクエストの最長時間 (マイクロ秒)。実行時間の長いクエリはボトルネックの可能性があります。
avg_request_duration	double precision	ファイルリクエストの平均時間 (マイクロ秒)。
max_request_parallelism	integer	Redshift Spectrum クエリでの 1 つのスライスの並列リクエストの最大数。
avg_request_parallelism	double precision	Redshift Spectrum クエリでの 1 つのスライスの並列リクエストの平均数。
total_slowdown_count	bigint	外部テーブルスキャン中に発生したスローダウンエラーを含む Amazon S3 リクエストの合計数。

列名	データ型	説明
max_slowdown_count	integer	外部テーブルスキャン中に 1 つのスライスで発生したスローダウンエラーを含む Amazon S3 リクエストの最大数。

サンプルクエリ

以下の例は、最後に完了されたクエリに関するスキャンステップの詳細を取得します。

```
select query, segment, elapsed, s3_scanned_rows, s3_scanned_bytes,
 s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query_summary
where query = pg_last_query_id()
order by query,segment;
```

```
query | segment | elapsed | s3_scanned_rows | s3_scanned_bytes | s3query_returned_rows
 | s3query_returned_bytes | files
------+---------+---------+-----------------+------------------+-----------------------
+------------------------+------
 4587 |       2 |   67811 |               0 |                0 |                      0
 |                      0 |     0
 4587 |       2 |  591568 |          172462 |         11260097 |                   8513
 |                 170260 |     1
 4587 |       2 |  216849 |               0 |                0 |                      0
 |                      0 |     0
 4587 |       2 |  216671 |               0 |                0 |                      0
 |                      0 |     0
```

SVL_S3RETRIES

SVL_S3RETRIES ビューを使用して、Amazon S3 に基づく Amazon Redshift Spectrum クエリが失敗した理由に関する情報を入手します。

SVL_S3RETRIES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明		
query	integer	クエリ ID。		
segment	integer	セグメント番号。 複数のセグメントから構成された 1 つのクエリ。各セグメントは 1 つ以上のステップから構成されます。複数のクエリセグメントを同時に実行できます。各セグメントは 1 つのプロセスで実行されます。		
node	integer	ノード番号。		
slice	integer	特定のセグメントが実行されたデータスライス。		
eventtime	タイムゾーンなしのタイムスタンプ	ステップの実行が開始された時間 (UTC)。		
retries	integer	クエリの再試行回数。		
successful_fetches	integer	データが返された回数。		
file_size	bigint	このファイルのサイズ (バイト単位)。		
location	text	テーブルの場所。		
message	text	エラーメッセージ。		

サンプルクエリ

次の例では、失敗した S3 クエリに関するデータを取得します。

```
SELECT svl_s3retries.query, svl_s3retries.segment, svl_s3retries.node,
 svl_s3retries.slice, svl_s3retries.eventtime, svl_s3retries.retries,
svl_s3retries.successful_fetches, svl_s3retries.file_size,
 btrim((svl_s3retries."location")::text) AS "location",
 btrim((svl_s3retries.message)::text)
AS message FROM svl_s3retries;
```

SVL_SPATIAL_SIMPLIFY

システムビュー SVL_SPATIAL_SIMPLIFY をクエリして、COPY コマンドを使用して、簡略化された空間ジオメトリオブジェクトに関する情報を取得できます。シェープファイルで COPY を使用する場合、SIMPLIFY `tolerance`、SIMPLIFY AUTO、および SIMPLIFY AUTO `max_tolerance` の取り込みオプションを指定できます。簡略化の結果は SVL_SPATIAL_SIMPLIFY システムビューに要約されます。

SIMPLIFY AUTO `max_tolerance` が設定されている場合、このビューには、最大サイズを超えた各ジオメトリの行が含まれます。SIMPLIFY `tolerance` が設定されている場合、COPY オペレーション全体の 1 行が保存されます。この行は、COPY クエリ ID と指定された簡略化の許容値を参照します。

SVL_SPATIAL_SIMPLIFY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_SPATIAL_SIMPLIFY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
query	integer	この行を生成したクエリの ID (COPY コマンド)。

列名	データ型	説明
line_number	integer	COPY SIMPLIFY AUTO オプションが指定されている場合、この値はシェープファイル内の簡略化されたレコードのレコード番号です。
maximum_tolerance	double	COPY コマンドで指定された距離許容値。これは、SIMPLIFY AUTO オプションを使用した最大許容値、または SIMPLIFY オプションを使用した固定許容値のいずれかです。
initial_size	integer	簡略化前の GEOMETRY データ値のサイズ (バイト単位)。
簡略化済み	char(1)	COPY SIMPLIFY AUTO オプションが指定されている場合、ジオメトリが正常に簡略化されている場合は t、それ以外の場合は f です。指定した最大許容値で簡略化した後でも、そのサイズが最大ジオメトリのサイズよりも大きい場合、ジオメトリが正常に簡略化されないことがあります。
final_size	integer	COPY SIMPLIFY AUTO オプションが指定されている場合、これは簡略化後のジオメトリのサイズ (バイト単位) です。
final_tolerance	double	

サンプルクエリ

次のクエリは、COPY で簡略化されたレコードのリストを返します。

```
SELECT * FROM svl_spatial_simplify WHERE query = pg_last_copy_id();
 query | line_number | maximum_tolerance | initial_size | simplified | final_size |
 final_tolerance
-------+-------------+-------------------+--------------+------------+------------
+----------------------
    20 |     1184704 |                -1 |      1513736 | t          |    1008808 |
 1.276386653895e-05
```

```
    20 |     1664115 |                   -1 |       1233456 | t          |    1023584 |
 6.11707814796635e-06
```

SVL_SPECTRUM_SCAN_ERROR

システムビュー SVL_SPECTRUM_SCAN_ERROR をクエリして Redshift Spectrum スキャンエラーに関する情報を取得することができます。

SVL_SPECTRUM_SCAN_ERROR はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_EXTERNAL_QUERY_ERROR でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

ログに記録されたエラーのサンプルを表示します。デフォルトは、クエリあたり 10 エントリです。

列名	データ型	説明
userid	integer	この行を生成したユーザーの ID。
query	integer	この行を生成したクエリの ID。
ロケーション	character(128)	クエリされているデータの場所。
rowid	character(128)	ファイル内のエラー場所。`rowid` パーツは : (コロン) で区切られ、追加のパーツを今後追加することができます。 `row_offset :row_group :row_id` row_offset は、ファイル内の行のオフセット (バイト単位) で、サポートされていないファイルフォーマットの場合は -1 に設定されます。テーブルは row_groups に分割され、各グループには異なる row_id の行があります。

列名	データ型	説明
colname	character(128)	クエリによって返された列の名前。
original_value	character(128)	クエリされた元の値。
modified_value	character(128)	クエリで指定されているデータ処理設定オプションに基づいて返された変更済みの値。
トリガー	character(128)	クエリで指定されているデータ処理オプション。
アクション	character(128)	クエリで指定されているデータ処理オプションに関連付けられたアクション。
action_value	character(128)	クエリで指定されているデータ処理オプションに関連付けられたアクションパラメータの値。
error_code	integer	クエリで指定されているデータ処理オプションの結果コード。

サンプルクエリ

次のクエリは、データ処理操作が実行された行のリストを返します。

```
SELECT * FROM svl_spectrum_scan_error;
```

クエリによって以下のような結果が返されます。

```
   userid  query     location                                      rowid    colname
          original_value              modified_value       trigger           action
                  action_valueerror_code
     100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_name
          Barclays Premier League     Barclays Premier Lea UNSPECIFIED       TRUNCATE
                                       156
     100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:0     league_nspi
          34595                        32767                 UNSPECIFIED
 OVERFLOW_VALUE                                    199
```

```
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:1     league_nspi
          34151                        32767                     UNSPECIFIED
OVERFLOW_VALUE                                      199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_name
         Barclays Premier League     Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                       156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:2     league_nspi
          33223                        32767                     UNSPECIFIED
OVERFLOW_VALUE                                      199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_name
         Barclays Premier League     Barclays Premier Lea UNSPECIFIED      TRUNCATE
                                       156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:3     league_nspi
          32808                        32767                     UNSPECIFIED
OVERFLOW_VALUE                                      199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:4     league_nspi
          32790                        32767                     UNSPECIFIED
OVERFLOW_VALUE                                      199
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:5     league_name
         Spanish Primera Division    Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                       156
    100   1574007   s3://spectrum-uddh/league/spi_global_rankings.0:6     league_name
         Spanish Primera Division    Spanish Primera Divi UNSPECIFIED      TRUNCATE
                                       156
```

SVL_STATEMENTTEXT

SVL_STATEMENTTEXT ビューを使用して、システムで実行されたすべての SQL コマンドの完全な記録を取得します。

SVL_STATEMENTTEXT ビューにはSTL_DDLTEXT、STL_QUERYTEXT、STL_UTILITYTEXT テーブルのすべての行の統合が含まれます。また、このビューには STL_QUERY テーブルの結合も含まれます。

SVL_STATEMENTTEXT はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	エントリを生成したユーザーの ID。
xid	bigint	ステートメントに関連付けられるトランザクション ID。
pid	integer	ステートメントのプロセス ID。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドは空になります。
starttime	timestamp	ステートメントの実行が開始された正確な時間。秒の小数部の精度 (6 桁) を使用します。例: **2009-06-12 11:29:19.131358**
endtime	timestamp	ステートメントの実行が終了した正確な時間。秒の小数部の精度 (6 桁) を使用します。例: **2009-06-12 11:29:19.193640**
sequence	integer	1 つのステートメントに含まれる文字数が 200 を超える場合、そのステートメントは追加の行に記録されます。シーケンス 0 が最初の行、1 が 2 番目の行、という順番です。
type	varchar(10)	SQL ステートメントのタイプ。**QUERY**、**DDL**、または **UTILITY** です。
text	character(200)	200 文字単位の SQL テキスト。このフィールドには、バックスラッシュ (\\) や改行 (\n) などの特殊文字が含まれる場合があります。

サンプルクエリ

次のクエリでは、2009 年 6 月 16 日に実行された DDL ステートメントを返します。

```
select starttime, type, rtrim(text) from svl_statementtext
where starttime like '2009-06-16%' and type='DDL' order by starttime asc;

starttime                  | type |              rtrim
---------------------------|------|--------------------------------
2009-06-16 10:36:50.625097 | DDL  | create table ddltest(c1 int);
2009-06-16 15:02:16.006341 | DDL  | drop view alltickitjoin;
2009-06-16 15:02:23.65285  | DDL  | drop table sales;
2009-06-16 15:02:24.548928 | DDL  | drop table listing;
2009-06-16 15:02:25.536655 | DDL  | drop table event;
...
```

ストアド SQL の再構築

SVL_STATEMENTTEXT の text 列に保存されている SQL を再構築するには、SELECT ステートメントを実行して、text 列の 1 つ以上の部分から SQL を作成します。再構築された SQL を実行する前に、特殊文字 (\n) がある場合は、改行に置き換えます。次の SELECT ステートメントの結果は、query_statementフィールドに再構築された SQL の行です。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS query_statement
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

例えば、次のクエリでは 3 つの列を選択します。クエリ自体は 200 文字より長く、SVL_STATEMENTTEXT の部分に保存されます。

```
select
1 AS a0123456789012345678901234567890123456789012345678901234567890,
2 AS b0123456789012345678901234567890123456789012345678901234567890,
3 AS b012345678901234567890123456789012345678901234
FROM stl_querytext;
```

この例では、クエリは SVL_STATEMENTTEXT の text 列の 2 つの部分 (行) に保存されます。

```
select sequence, text from SVL_STATEMENTTEXT where pid = pg_backend_pid() order by
 starttime, sequence;
```

```

 sequence |
                text
```

```
----------
+----------------------------------------------------------------------------------------
        0 | select\n1 AS
 a012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n2 AS
 b012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234
        1 | \nFROM stl_querytext;
```

STL_STATEMENTTEXT に保存された SQL を再構築するには、次の SQL を実行します。

```
select LISTAGG(CASE WHEN LEN(RTRIM(text)) = 0 THEN text ELSE RTRIM(text) END, '')
 within group (order by sequence) AS text
from SVL_STATEMENTTEXT where pid=pg_backend_pid();
```

再構築された SQL をクライアントで使用するには、特殊文字 (\n) を改行に置き換えます。

```
                       text

-----------------------------------------------------------------------------------------
 select\n1 AS a012345678901234567890123456789012345678901234567890123456789012345678901234567890,
\n2 AS b012345678901234567890123456789012345678901234567890123456789012345678901234567890,\n3 AS
 b012345678901234567890123456789012345678901234\nFROM stl_querytext;
```

SVL_STORED_PROC_CALL

システムビュー SVL_STORED_PROC_CALL をクエリして、ストアドプロシージャコールに関する開始時刻、終了時刻、およびコールがキャンセルされたかどうかなどの情報を取得できます。ストアドプロシージャ呼び出しごとにクエリ ID を受け取ります。

SVL_STORED_PROC_CALL はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_PROCEDURE_CALL でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	ステートメントを実行するために使用された権限を持つユーザーの ID。この呼び出しが SECURITY DEFINER ストアドプロシージャ内にネストされている場合、これはそのストアドプロシージャの所有者のユーザー ID です。
session_userid	integer	セッションの作成元であり、最上位ストアドプロシージャ呼び出しの呼び出し元であるユーザーの ID。
query	integer	プロシージャ呼び出しのクエリ ID。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドの値はデフォルト になります。
xid	bigint	トランザクション ID。
pid	integer	プロセス ID。通常、セッション内のすべてのクエリは同一プロセスで実行されるため、一連のクエリを同一セッションで実行した場合、通常、この値は一定です。特定の内部イベントに続いて、Amazon Redshift はアクティブなセッションを再起動し、新しい PID 値を割り当てる場合があります。詳細については、「STL_RESTARTED_SESSIONS」を参照してください。
database	character(32)	クエリが発行されたときにユーザーが接続されたデータベースの名前。
querytxt	character(4000)	プロシージャ呼び出しのクエリの実際のテキスト。
starttime	timestamp	UTC で表されたクエリの実行開始時刻。秒の小数部の精度 (6 桁) を使用します。例: **2009-06-12 11:29:19.131358.**

列名	データ型	説明
endtime	timestamp	UTC で表されたクエリの実行終了時刻。秒の小数部の精度 (6 桁) を使用します。例: **2009-06-12 11:29:19.131358.**
aborted	integer	ストアドプロシージャがシステムによって停止されたか、ユーザーによってキャンセルされた場合、この列は 1 になります。呼び出しが最後まで実行された場合、この列は 0 になります。
from_sp_call	integer	プロシージャ呼び出しが別のプロシージャ呼び出しによって呼び出された場合、この列は外部呼び出しのクエリ ID になります。それ以外の場合、フィールドは NULL です。

サンプルクエリ

次のクエリは、降順の経過時間と、過去 1 日のストアドプロシージャ呼び出しの完了ステータスを返します。

```
select query, datediff(seconds, starttime, endtime) as elapsed_time, aborted,
 trim(querytxt) as call from svl_stored_proc_call where starttime >= getdate() -
 interval '1 day' order by 2 desc;

  query | elapsed_time | aborted |                                       call
--------+--------------+---------
+-------------------------------------------------------------------------------
   4166 |            7 |       0 | call search_batch_status(35,'succeeded');
   2433 |            3 |       0 | call test_batch (123456)
   1810 |            1 |       0 | call prod_benchmark (123456)
   1836 |            1 |       0 | call prod_testing (123456)
   1808 |            1 |       0 | call prod_portfolio ('N', 123456)
   1816 |            1 |       1 | call prod_portfolio ('Y', 123456)
```

SVL_STORED_PROC_MESSAGES

システムビュー SVL_STORED_PROC_MESSAGES を照会して、ストアドプロシージャメッセージに関する情報を取得できます。ストアドプロシージャコールがキャンセルされた場合でも、発生したメッセージがログに記録されます。ストアドプロシージャ呼び出しごとにクエリ

ID を受け取ります。ログ記録されるメッセージの最小レベルを設定する方法の詳細については、stored_proc_log_min_messages を参照してください。

SVL_STORED_PROC_MESSAGES はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_PROCEDURE_MESSAGES でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
userid	integer	ステートメントを実行するために使用された権限を持つユーザーの ID。この呼び出しが SECURITY DEFINER ストアドプロシージャ内にネストされている場合、これはそのストアドプロシージャの所有者のユーザー ID です。
session_userid	integer	セッションの作成元であり、最上位ストアドプロシージャ呼び出しの呼び出し元であるユーザーの ID。
pid	integer	プロセス ID。
xid	bigint	プロシージャ呼び出しのクエリのトランザクション ID。
query	integer	プロシージャ呼び出しのクエリ ID。
recordtime	timestamp	メッセージが発生した時刻 (UTC)。
loglevel	integer	発生したメッセージのログレベルの数値。使用できる値: 20 – LOG の場合 30 – INFO の場合 40 – NOTICE の場合 50 – WARNING の場合 60 – EXCEPTION の場合
loglevel_text	character(10)	loglevel の数値に対応するログレベル。可能な値: LOG、INFO、NOTICE、WARNING、EXCEPTION。
メッセージ	character(1024)	発行されたメッセージのテキスト。

列名	データ型	説明
linenum	integer	発行されたステートメントの行番号。
querytext	character(500)	プロシージャ呼び出しのクエリの実際のテキスト。
label	character(320)	クエリを実行するために使用される名前、または SET QUERY_GROUP コマンドによって定義されるラベル。クエリがファイルベースでないか、QUERY_GROUP パラメータが設定されていない場合、このフィールドの値はデフォルト になります。
aborted	integer	ストアドプロシージャがシステムによって停止されたか、ユーザーによってキャンセルされた場合、この列は 1 になります。呼び出しが最後まで実行された場合、この列は 0 になります。
message_xid	bigint	発行されたメッセージのトランザクション ID。

サンプルクエリ

次の SQL ステートメントは、SVL_STORED_PROC_MESSAGES を使用して発行されたメッセージを確認する方法を示しています。

```
-- Create and run a stored procedure
CREATE OR REPLACE PROCEDURE test_proc1(f1 int) AS
$$
BEGIN
    RAISE INFO 'Log Level: Input f1 is %',f1;
    RAISE NOTICE 'Notice Level: Input f1 is %',f1;
    EXECUTE 'select invalid';
    RAISE NOTICE 'Should not print this';

EXCEPTION WHEN OTHERS THEN
     raise exception 'EXCEPTION level: Exception Handling';
END;
$$ LANGUAGE plpgsql;

-- Call this stored procedure
```

```
CALL test_proc1(2);

-- Show raised messages with level higher than INFO
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel > 30 AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
             | aborted
-------+----------------------------+----------+---------------
+-------------------------------------+---------
   193 | 2020-03-17 23:57:18.277196 |       40 | NOTICE        | Notice Level: Input f1
 is 2          |       1
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
(2 rows)

-- Show raised messages at EXCEPTION level
SELECT query, recordtime, loglevel, loglevel_text, trim(message) as message, aborted
 FROM svl_stored_proc_messages
  WHERE loglevel_text = 'EXCEPTION' AND query = 193 ORDER BY recordtime;

 query |         recordtime         | loglevel | loglevel_text |               message
             | aborted
-------+----------------------------+----------+---------------
+-------------------------------------+---------
   193 | 2020-03-17 23:57:18.277987 |       60 | EXCEPTION     | EXCEPTION level:
 Exception Handling |       1
```

次の SQL ステートメントは、ストアドプロシージャの作成時に、SET オプションと SVL_STORED_PROC_MESSAGES を使用して発行されたメッセージを表示する方法を示します。test_proc() には最低ログレベルの NOTICE があるため、SVL_STORED_PROC_MESSAGES には、NOTICE、WARNING、EXCEPTION レベルのメッセージのみが記録されます。

```
-- Create a stored procedure with minimum log level of NOTICE
CREATE OR REPLACE PROCEDURE test_proc() AS
$$
BEGIN
    RAISE LOG 'Raise LOG messages';
    RAISE INFO 'Raise INFO messages';
    RAISE NOTICE 'Raise NOTICE messages';
    RAISE WARNING 'Raise WARNING messages';
    RAISE EXCEPTION 'Raise EXCEPTION messages';
```

```
    RAISE WARNING 'Raise WARNING messages again'; -- not reachable
END;
$$ LANGUAGE plpgsql SET stored_proc_log_min_messages = NOTICE;

-- Call this stored procedure
CALL test_proc();

-- Show the raised messages
SELECT query, recordtime, loglevel_text, trim(message) as message, aborted FROM
 svl_stored_proc_messages
  WHERE query = 149 ORDER BY recordtime;

 query |         recordtime         | loglevel_text |          message          |
 aborted
-------+----------------------------+---------------+-------------------------
+---------
   149 | 2020-03-16 21:51:54.847627 | NOTICE        | Raise NOTICE messages    |
 1
   149 | 2020-03-16 21:51:54.84766  | WARNING       | Raise WARNING messages   |
 1
   149 | 2020-03-16 21:51:54.847668 | EXCEPTION     | Raise EXCEPTION messages |
 1
(3 rows)
```

SVL_TERMINATE

ユーザーがプロセスをキャンセルまたは終了した時刻を記録します。

SELECT PG_TERMINATE_BACKEND(pid)、SELECT PG_CANCEL_BACKEND(pid)、および CANCEL pid は、SVL_TERMINATE にログのエントリを作成します。

SVL_TERMINATE は、スーパーユーザーにのみ表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_QUERY_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
pid	integer	キャンセルまたは終了されたプロセスのプロセス ID。
eventtime	timestamp	プロセスがキャンセルまたは終了された時刻。
userid	integer	コマンドを実行しているユーザーのユーザー ID。
type	string	終了のタイプ。CANCEL または TERMINATE です。

次のコマンドでは、キャンセルされた最新のクエリが表示されます。

```
select * from svl_terminate order by eventtime desc limit 1;
 pid  |         eventtime          | userid |  type
------+----------------------------+--------+--------
 8324 | 2020-03-24 09:42:07.298937 |      1 | CANCEL
(1 row)
```

SVL_UDF_LOG

システムで定義された、ユーザー定義関数 (UDF) の実行時に生成するエラーと警告メッセージを記録します。

SVL_UDF_LOG はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_UDF_LOG でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
query	bigint	クエリ ID。この ID を使用して、他の各種システムテーブ

列名	データ型	説明
		ルおよびビューを結合できます。
メッセージ	char(4096)	関数によって生成されたメッセージ。
作成済み	timestamp	ログが作成された時間。
トレースバック	char(4096)	使用可能な場合、この値は UDF 用のスタックのトレースバックを提供します。詳細については、Python の標準ライブラリの traceback を参照してください。
funcname	character(256)	実行中の UDF の名前。
node	integer	メッセージが生成されたノード。
slice	integer	メッセージが生成されたスライス。
seq	integer	スライス上のメッセージのシーケンス。

サンプルクエリ

次の例では UDF がシステム定義されたエラーをどのように処理するかを示します。最初のブロックは、引数の逆を返す UDF 関数の定義を示します。2 番目のブロックに示すように、関数を実行して引数を 0 に指定すると、関数はエラーを返します。3 番目のステートメントは、SVL_UDF_LOG とログされたエラーメッセージを読み取ります

```
-- Create a function to find the inverse of a number

CREATE OR REPLACE FUNCTION  f_udf_inv(a int)
  RETURNS float IMMUTABLE
AS $$
```

```
    return 1/a
$$ LANGUAGE plpythonu;

-- Run the function with a 0 argument to create an error
Select f_udf_inv(0) from sales;

-- Query SVL_UDF_LOG to view the message

Select query, created, message::varchar
from svl_udf_log;

 query |          created           | message
-------+----------------------------
+---------------------------------------------------------
  2211 | 2015-08-22 00:11:12.04819  | ZeroDivisionError: long division or modulo by
 zero\nNone
```

次の例では、UDF にログされたメッセージと警告メッセージを追加することで、ゼロで分割された操作がエラーメッセージで停止する代わりに警告メッセージとなります。

```
-- Create a function to find the inverse of a number and log a warning

CREATE OR REPLACE FUNCTION f_udf_inv_log(a int)
  RETURNS float IMMUTABLE
 AS $$
  import logging
  logger = logging.getLogger() #get root logger
  if a==0:
    logger.warning('You attempted to divide by zero.\nReturning zero instead of error.
\n')
    return 0
  else:
     return 1/a
$$ LANGUAGE plpythonu;
```

次の例は、関数を実行し、その後 SVL_UDF_LOG をクエリしていうメッセージを表示します。

```
-- Run the function with a 0 argument to trigger the warning
Select f_udf_inv_log(0) from sales;

-- Query SVL_UDF_LOG to view the message
```

```
Select query, created, message::varchar
from svl_udf_log;

query |          created           | message
------+----------------------------+---------------------------------
    0 | 2015-08-22 00:11:12.04819  | You attempted to divide by zero.
                                     Returning zero instead of error.
```

SVL_USER_INFO

SVL_USER_INFO ビューを使用して、Amazon Redshift データベースユーザーに関するデータを取得できます。

SVL_USER_INFO はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
usename	text	このロールのユーザー名。
usesysid	integer	このユーザーのユーザー ID。
usecreatedb	boolean	ユーザーがデータベースを作成する権限を持っているかどうかを示す値。
usesuper	boolean	ユーザーがスーパーユーザーであるかどうかを示す値。
usecatupd	boolean	ユーザーがシステムカタログを更新できるかどうかを示す値。
useconnlimit	text	ユーザーが開くことができる接続数。
syslogaccess	text	ユーザーがシステムログにアクセスできるかどうかを示す値。指定できる値は `RESTRICTED` と `UNRESTRICTED` の 2 つです。`RESTRICTED` は、スーパーユーザーではないユーザーが自分のレコードを表示できることを意味します。`UNRESTRICTED` は、スーパーユーザーではないユーザーが、`SELECT` 権限を持っているシステムビューおよびテーブル内のすべてのレコードを表示できることを意味します。

列名	データ型	説明
last_ddl_ts	timestamp	ユーザーが実行した最後のデータ定義言語 (DDL) create ステートメントのタイムスタンプ。
sessiontimeout	integer	タイムアウトするまで、セッションが非アクティブまたはアイドル状態を維持する最大秒数。0 の場合は、タイムアウトが設定されていないことを示します。クラスターのアイドルまたは非アクティブ状態のタイムアウト設定については、「Amazon Redshift 管理ガイド」の「Amazon Redshift でのクォータと制限」を参照してください。
external_id	text	サードパーティーにおける ID プロバイダー内のユーザーの一意の識別子。

サンプルクエリ

次のコマンドは SVL_USER_INFO からユーザー情報を取得します。

```
SELECT * FROM SVL_USER_INFO;
```

SVL_VACUUM_PERCENTAGE

SVL_VACUUM_PERCENTAGE ビューはバキューム処理の実行後にテーブルに割り当てられるデータブロックのパーセントを示します。このパーセント数は再利用されたディスクスペースの量を示します。バキューム処理ユーティリティの詳細については、VACUUM コマンドを参照してください。

SVL_VACUUM_PERCENTAGE はスーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

このテーブルの一部またはすべてのデータは、SYS モニタリングビュー SYS_VACUUM_HISTORY でも確認できます。SYS モニタリングビューのデータは、使いやすく理解しやすいようにフォーマットされます。クエリには、SYS モニタリングビューを使用することをお勧めします。

テーブルの列

列名	データ型	説明
xid	bigint	vacuum ステートメントのトランザクション ID。

列名	データ型	説明
table_id	integer	バキューム処理されたテーブルのテーブル ID。
割合(%)	bigint	バキューム処理後のデータブロックのパーセント (バキューム処理が実行される前のテーブルのブロック数との相対値)。

サンプルクエリ

次のクエリは、テーブル 100238 での具体的な処理のパーセントを示します。

```
select * from svl_vacuum_percentage
where table_id=100238 and xid=2200;

xid  | table_id | percentage
-----+----------+------------
1337 |   100238 |         60
(1 row)
```

バキューム処理後、テーブルには元のブロックの 60% が含まれます。

システムカタログテーブル

トピック

- PG_ATTRIBUTE_INFO
- PG_CLASS_INFO
- PG_DATABASE_INFO
- PG_DEFAULT_ACL
- PG_EXTERNAL_SCHEMA
- PG_LIBRARY
- PG_PROC_INFO
- PG_STATISTIC_INDICATOR
- PG_TABLE_DEF
- PG_USER_INFO
- カタログテーブルへのクエリの実行

システムカタログは、テーブルと列に関する情報などのスキーマメタデータを格納します。システムカタログテーブルには PG プレフィックスが付けられています。

Amazon Redshift ユーザーは、標準の PostgreSQL カタログテーブルにアクセスできます。PostgreSQL システムカタログの詳細については、「PostgreSQL System Tables」を参照してください。

PG_ATTRIBUTE_INFO

PG_ATTRIBUTE_INFO は、PostgreSQL カタログテーブル PG_ATTRIBUTE と内部カタログテーブル PG_ATTRIBUTE_ACL 上に構築された Amazon Redshift システムビューです。PG_ATTRIBUTE_INFO には、テーブルまたはある場合は、列アクセス制御リストを含むビューの列に関する詳細が含まれます。

テーブルの列

PG_ATTRIBUTE_INFO には、PG_ATTRIBUTE の列に加えて、次の列が表示されます。

列名	データ型	説明
attacl	aclitem[]	この列に対して特別に付与されている列レベルのアクセス権限 (存在する場合)。

PG_CLASS_INFO

PG_CLASS_INFO は、PostgreSQL のカタログテーブル PG_CLASS と PG_CLASS_EXTENDED に基づいて構築された Amazon Redshift システムビューです。PG_CLASS_INFO には、テーブル作成時間と現在の分散スタイルに関する詳細が含まれます。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください

PG_CLASS_INFO はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

PG_CLASS_INFO は、PG_CLASS の列に加えて、以下の列を示します。PG_CLASS_INFO テーブルでは、PG_CLASS の `oid` 列は `relid` と呼ばれます。

列名	データ型	説明
relcreationtime	timestamp	テーブルが作成された時間 (UTC)。
releffectivediststyle	integer	テーブルのディストリビューションスタイル、またはテーブルが自動ディストリビューションを使用している場合は Amazon Redshift によって割り当てられた現在のディストリビューションスタイル。

テーブルの現在の分散スタイルは、PG_CLASS_INFO の RELEFFECTIVEDISTSTYLE 列に示されます。テーブルが自動分散を使用する場合、RELEFFECTIVEDISTSTYLE は 10、11、または 12 です。これは、効率的な分散スタイルが AUTO (ALL)、AUTO (EVEN)、または AUTO (KEY) のどれであるかを示します。テーブルが自動分散を使用する場合、分散スタイルは当初 AUTO (ALL) と表示され、その後テーブルが大きくなると AUTO (EVEN) に変わり、列が分散キーとして有効と見なされると AUTO (KEY) と表示されます。

次の表は、RELEFFECTIVEDISTSTYLE 列に含まれる各値の分散スタイルを示しています。

RELEFFECTIVEDISTSTYLE	現在の分散スタイル
0	EVEN
1	KEY
8	ALL
10	AUTO (ALL)
11	AUTO (EVEN)
12	AUTO (KEY)

例

次のクエリでは、カタログ内のテーブルの現在の分散スタイルが返ります。

```
select relid as tableid,trim(nspname) as schemaname,trim(relname) as
 tablename,reldiststyle,releffectivediststyle,
CASE WHEN "reldiststyle" = 0 THEN 'EVEN'::text
     WHEN "reldiststyle" = 1 THEN 'KEY'::text
     WHEN "reldiststyle" = 8 THEN 'ALL'::text
     WHEN "releffectivediststyle" = 10 THEN 'AUTO(ALL)'::text
     WHEN "releffectivediststyle" = 11 THEN 'AUTO(EVEN)'::text
     WHEN "releffectivediststyle" = 12 THEN 'AUTO(KEY)'::text ELSE '<<UNKNOWN>>'::text
 END as diststyle,relcreationtime
from pg_class_info a left join pg_namespace b on a.relnamespace=b.oid;
```

```
 tableid | schemaname | tablename | reldiststyle | releffectivediststyle | diststyle  |
      relcreationtime
---------+------------+-----------+--------------+-----------------------+------------
+----------------------------
 3638033 | public     | customer  |            0 |                     0 | EVEN       |
 2019-06-13 15:02:50.666718
 3638037 | public     | sales     |            1 |                     1 | KEY        |
 2019-06-13 15:03:29.595007
 3638035 | public     | lineitem  |            8 |                     8 | ALL        |
 2019-06-13 15:03:01.378538
 3638039 | public     | product   |            9 |                    10 | AUTO(ALL)  |
 2019-06-13 15:03:42.691611
 3638041 | public     | shipping  |            9 |                    11 | AUTO(EVEN) |
 2019-06-13 15:03:53.69192
 3638043 | public     | support   |            9 |                    12 | AUTO(KEY)  |
 2019-06-13 15:03:59.120695
(6 rows)
```

PG_DATABASE_INFO

PG_DATABASE_INFO は、PostgreSQL カタログテーブル PG_DATABASE を拡張する Amazon Redshift システムビューです。

PG_DATABASE_INFO はすべてのユーザーに表示されます。

テーブルの列

PG_DATABASE_INFO には、PG_DATABASE の列に加えて、次の列が含まれます。PG_DATABASE_INFO テーブルでは、PG_DATABASEの `oid` 列は `datid` と呼ばれます。詳細については、[PostgreSQL ドキュメント](https://www.postgresql.org/docs/8.0/static/catalog-pg-database.html)を参照してください。

列名	データ型	説明
datid	oid	システムテーブルによって内部的に使用されるオブジェクト識別子 (OID)。
datconnlimit	text	このデータベースに対して作成できる同時接続の最大数。値を -1 にすると、無制限に設定されます。

PG_DEFAULT_ACL

デフォルトのアクセス権限に関する情報を格納します。デフォルトのアクセス権限の詳細については、「ALTER DEFAULT PRIVILEGES」参照してください。

PG_DEFAULT_ACL はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
defacluser	integer	リストされている権限が適用されるユーザーの ID。
defaclnamespace	oid	デフォルト権限が適用されるスキーマのオブジェクト ID。スキーマが指定されていない場合、デフォルト値は 0 です。
defaclobjtype	character	デフォルト権限が適用されるオブジェクトの型。有効な値は次のとおりです。 • r – リレーション (テーブルまたはビュー) • f–function • p – ストアドプロシージャ
defaclacl	aclitem[]	指定されたユーザーまたはユーザーグループとオブジェクト型のデフォルト権限を定義する文字列。

列名	データ型	説明
		権限がユーザーに付与される場合、文字列は次のようになります。 *{ username=privilegestring/grantor }* username 権限が付与されるユーザーの名前。username を省略すると、権限は PUBLIC に付与されます。 権限がユーザーグループに付与される場合、文字列は次のようになります。 *{ "group groupname=privilegestring/grantor" }* privilegestring どの権限を付与するかを指定する文字列。 有効な値は次のとおりです。 • r–SELECT (読み込み) • a – INSERT (付加) • w–UPDATE (書き込み) • d–DELETE • x – 外部キー制限 (リファレンス) を作成する権限を付与します。 • X–EXECUTE • * 権限を付与されるユーザーが、他のユーザーにも同じ権限を付与できる (WITH GRANT OPTION) ことを示します。 grantor 権限を付与したユーザーの名前。

列名	データ型	説明
		次の例は、ユーザー admin が WITH GRANT OPTION を含むすべての権限をユーザー dbuser に付与したことを示します。 `dbuser=r*a*w*d*x*X*/admin`

例

次のクエリはデータベースに対して定義されているすべてのデフォルト権限を返します。

```
select pg_get_userbyid(d.defacluser) as user,
n.nspname as schema,
case d.defaclobjtype when 'r' then 'tables' when 'f' then 'functions' end
as object_type,
array_to_string(d.defaclacl, ' + ')  as default_privileges
from pg_catalog.pg_default_acl d
left join pg_catalog.pg_namespace n on n.oid = d.defaclnamespace;

 user  | schema | object_type |              default_privileges
-------+--------+-------------+-------------------------------------------------------
 admin | tickit | tables      | user1=r/admin + "group group1=a/admin" + user2=w/admin
```

前の例の結果は、ユーザー admin が tickit スキーマで作成するすべての新しいテーブルにおいて、adminが user1 には SELECT 権限を、group1には INSERT 権限を、user2には UPDATE 権限を付与することを示します。

PG_EXTERNAL_SCHEMA

外部スキーマの情報を保存します。

PG_EXTERNAL_SCHEMA はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できます。通常のユーザーはアクセスできるメタデータのみを表示できます。詳細については、「CREATE EXTERNAL SCHEMA」を参照してください。

テーブルの列

列名	データ型	説明
esoid	oid	外部スキーマ ID。
eskind	integer	外部スキーマのタイプ。
esdbname	text	外部データベース名。
esoptions	text	外部スキーマのオプション。

例

以下の例に示しているのは、外部スキーマの詳細です。

```
select esoid, nspname as schemaname, nspowner, esdbname as external_db, esoptions
from pg_namespace a,pg_external_schema b where a.oid=b.esoid;

esoid  | schemaname      | nspowner | external_db | esoptions

-------+-----------------+----------+-------------
+-------------------------------------------------------------
100134 | spectrum_schema |      100 | spectrum_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
100135 | spectrum        |      100 | spectrumdb  |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
100149 | external        |      100 | external_db |
 {"IAM_ROLE":"arn:aws:iam::123456789012:role/mySpectrumRole"}
```

PG_LIBRARY

ユーザー定義のライブラリに関する情報を格納します。

PG_LIBRARY はすべてのユーザーに表示されます。スーパーユーザーはすべての行を表示できますが、通常のユーザーは自分のデータのみを表示できます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください。

テーブルの列

列名	データ型	説明
name	name	ライブラリ名です。
language_oid	oid	将来の利用のために予約されています。
file_store_id	integer	将来の利用のために予約されています。
owner	integer	ライブラリ所有者のユーザー ID です。

例

次の例では、ユーザーがインストールしたライブラリの情報を返します。

```
select * from pg_library;

name        | language_oid | file_store_id | owner
-----------+--------------+---------------+------
f_urlparse |       108254 |          2000 |   100
```

PG_PROC_INFO

PG_PROC_INFO は、PostgreSQL カタログテーブル PG_PROC と内部カタログテーブル PG_PROC_EXTENDED に構築された Amazon Redshift システムビューです。PG_PROC_INFO には、出力引数に関する情報 (ある場合) など、ストアドプロシージャおよび関数に関する詳細が含まれます。

テーブルの列

PG_PROC_INFO は、PG_PROC の列に加えて、以下の列を示します。PG_PROC_INFO テーブルでは、PG_PROC の `oid` 列は `prooid` と呼ばれます。

列名	データ型	説明
prooid	oid	関数またはストアドプロシージャのオブジェクト ID。

列名	データ型	説明
prokind	"char"	関数またはストアドプロシージャの型を示す値。正規関数は「f」、ストアドプロシージャは「p」、集計関数は「a」が値になります。
proargmodes	"char"[ ]	プロシージャ引数のモードを含む配列。IN 引数は「i」、OUT 引数は「o」、INOUT 引数は「b」としてエンコードされます。すべての引数が IN 引数である場合、このフィールドは NULL になります。サブスクリプトは、proallargtypes 配列の位置に対応します。
proallargtypes	oid[ ]	プロシージャ引数のデータ型を含む配列。この配列にはすべての型の引数 (OUT 引数や INOUT 引数など) が含まれます。ただし、すべての引数が IN 引数である場合、このフィールドは NULL になります。サブスクリプトは 1 から始まります。一方、proargtypes のサブスクリプトは 0 から始まります。

PG_PROC_INFO のフィールド proargnames には、すべての型の引数 (OUT や INOUT など) の名前が含まれます。

PG_STATISTIC_INDICATOR

最後の ANALYZE 以降に挿入または削除された行数に関する情報を保存します。PG_STATISTIC_INDICATOR テーブルは DML オペレーションの後で頻繁に更新されるので、統計は概算です。

PG_STATISTIC_INDICATOR は、スーパーユーザーのみに表示されます。詳細については、「システムテーブルとビューのデータの可視性」を参照してください

テーブルの列

列名	データ型	説明
stairelid	oid	テーブル ID
stairows	float	テーブル内の合計行数。

列名	データ型	説明
staiins	float	最後の ANALYZE 以降に挿入された行数。
staidels	float	最後の ANALYZE 以降に削除または更新された行数。

例

次の例では、最後の ANALYZE 以降に変更されたテーブルの情報を返します。

```
select * from pg_statistic_indicator;

stairelid | stairows | staiins | staidels
----------+----------+---------+---------
   108271 |       11 |       0 |        0
   108275 |      365 |       0 |        0
   108278 |     8798 |       0 |        0
   108280 |    91865 |       0 |   100632
   108267 |    89981 |   49990 |     9999
   108269 |      808 |     606 |      374
   108282 |   152220 |   76110 |   248566
```

PG_TABLE_DEF

テーブルの列に関する情報を格納します。

PG_TABLE_DEF は、テーブルに関してユーザーに表示される情報のみを返します。もし PG_TABLE_DEF が予期した結果を返さない場合、該当のスキーマを含むように search_path パラメータが正しく設定されていることを確認します。

SVV_TABLE_INFO を使用すると、データ分散スキュー、キー分散スキュー、テーブルサイズ、統計情報など、テーブルに関するより包括的な情報を表示することができます。

テーブルの列

列名	データ型	説明
schemaname	name	スキーマ名。

列名	データ型	説明
tablename	name	テーブル名。
column	name	列名。
type	text	列のデータ型。
encoding	character(32)	列のエンコード。
distkey	boolean	この列がテーブルの分散キーである場合は True。
sortkey	integer	ソートキーの列の順序。テーブルが複合ソートキーを使用する場合、ソートキーに含まれるすべての列は、ソートキー内の列の位置を示す正の値を持ちます。テーブルがインターリーブソートキーを使用する場合、ソートキーに含まれる各列は正または負の値を交互に持ち、この値の絶対値がソートキー内の列の位置を示します。0 の場合、列はソートキーに含まれません。
notnull	boolean	列が NOT NULL 制約を持つ場合は True。

例

次に、LINEORDER_COMPOUND テーブルの複合ソートキーの列の例を示します。

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_compound'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |       1 | true
lo_custkey   | integer | none     | false   |       2 | true
lo_partkey   | integer | none     | true    |       3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
lo_orderdate | integer | delta    | false   |       5 | true
(5 rows)
```

次に、LINEORDER_INTERLEAVED テーブルのインターリーブソートキーの列の例を示します。

```
select "column", type, encoding, distkey, sortkey, "notnull"
from pg_table_def
where tablename = 'lineorder_interleaved'
and sortkey <> 0;

column       | type    | encoding | distkey | sortkey | notnull
-------------+---------+----------+---------+---------+--------
lo_orderkey  | integer | delta32k | false   |      -1 | true
lo_custkey   | integer | none     | false   |       2 | true
lo_partkey   | integer | none     | true    |      -3 | true
lo_suppkey   | integer | delta32k | false   |       4 | true
lo_orderdate | integer | delta    | false   |      -5 | true
(5 rows)
```

PG_TABLE_DEF は、検索パスに含まれているスキーマのテーブルの情報のみを返します。詳細については、「search_path」を参照してください

例えば、新しいスキーマと新しいテーブルを作成し、PG_TABLE_DEF にクエリを実行するとします。

```
create schema demo;
create table demo.demotable (one int);
select * from pg_table_def where tablename = 'demotable';

schemaname|tablename|column| type | encoding | distkey | sortkey | notnull
----------+---------+------+------+----------+---------+---------+--------
```

このクエリは、新しいテーブルの列を返しません。`search_path` の設定を検証します。

```
show search_path;

  search_path
---------------
 $user, public
(1 row)
```

demo スキーマを検索パスに追加し、クエリを再度実行します。

```
set search_path to '$user', 'public', 'demo';
```

```
select * from pg_table_def where tablename = 'demotable';

schemaname| tablename |column|  type   | encoding |distkey|sortkey| notnull
----------+-----------+------+---------+----------+-------+-------+--------
demo      | demotable | one  | integer | none     | f     |     0 | f
(1 row)
```

PG_USER_INFO

PG_USER_INFO は、ユーザー ID やパスワードの有効期限などのユーザー情報が表示される Amazon Redshift システムビューです。

PG_USER_INFO は、スーパーユーザーだけが閲覧できます。

テーブルの列

PG_USER_INFO には、次の列が含まれています。詳細については、PostgreSQL ドキュメントを参照してください。

列名	データ型	説明
usename	name	ユーザー名。
usesysid	integer	ユーザー ID。
usecreatedb	ブール値	ユーザーがデータベースを作成できる場合は true。
usesuper	ブール値	ユーザーがスーパーユーザーである場合は true。
usecatupd	ブール値	ユーザーがシステムカタログを更新できる場合は true。
passwd	text	パスワード。
valuntil	abstime	パスワードの有効期限の日時。
useconfig	text[]	ランタイム変数のセッションのデフォルト設定。
useconnlimit	text	ユーザーが開くことができる接続数。

カタログテーブルへのクエリの実行

トピック

- カタログクエリの例

通常、カタログテーブルとビュー (名前が **PG_** で始まる関係) を Amazon Redshift テーブルとビューに結合できます。

カタログテーブルは、Amazon Redshift がサポートしない多数のデータ型を使用します。クエリでカタログテーブルを Amazon Redshift テーブルに結合するときは、次のデータ型がサポートされます。

- ブール
- "char"
- float4
- int2
- int4
- int8
- name
- oid
- text
- varchar

サポートされないデータ型を持つ列を明示的または暗黙的に参照する結合クエリを記述すると、クエリはエラーを返します。カタログテーブルの一部で使用される SQL 関数も、PG_SETTINGS および PG_LOCKS テーブルで使用されるものを除いてサポートされません。

例えば、PG_STATS テーブルに対し、Amazon Redshift テーブルとの結合でクエリを実行することはできません。これは、関数がサポートされていないためです。

次のカタログテーブルとビューは、Amazon Redshift テーブルの情報に結合できる有益な情報を提供します。これらのテーブルの一部では、データ型と関数の制約のため、部分的なアクセスのみが許可されます。部分的にアクセス可能なテーブルにクエリを実行するときは、列を慎重に選択または参照してください。

次のテーブルは完全にアクセス可能で、サポートされない型または関数は含まれません。

- pg_attribute
- pg_cast
- pg_depend
- pg_description
- pg_locks
- pg_opclass

次のテーブルは部分的にアクセス可能で、サポートされない型、関数、および切り捨てられたテキスト列をいくつか含みます。テキスト列の値は varchar(256) 値に切り捨てられます。

- pg_class
- pg_constraint
- pg_database
- pg_group
- pg_language
- pg_namespace
- pg_operator
- pg_proc
- pg_settings
- pg_statistic
- pg_tables
- pg_type
- pg_user
- pg_views

ここにリストされていないカタログテーブルはアクセス不可能であるか、Amazon Redshift 管理者にとって有益でない可能性があります。ただし、Amazon Redshift テーブルへの結合がクエリに含まれていない場合は、任意のカタログテーブルまたはビューに対してオープンにクエリを実行できます。

Postgres カタログテーブルの OID 列を結合列として使用できます。例えば、結合条件 `pg_database.oid = stv_tbl_perm.db_id` は、STV_TBL_PERM テーブルで表示される DB_ID 列を持つ各 PG_DATABASE 行の内部データベースオブジェクト ID に一致します。OID 列は、テー

ブルからの選択時に表示されない内部プライマリキーです。カタログビューには OID 列はありません。

一部の Amazon Redshift 関数は、コンピューティングノードのみで実行される必要があります。ユーザーが作成したテーブルをクエリが参照すると、コンピューティングノードで SQL が実行されます。

カタログテーブルのみを参照するクエリ (PG_TABLE_DEF など、PG プレフィックスを持つテーブル) またはテーブルを参照しないクエリは、リーダーノードのみで実行されます。

コンピューティングノード関数を使用するクエリがユーザー定義のテーブルまたは Amazon Redshift システムテーブルを参照しない場合、次のエラーが返されます。

```
[Amazon](500310) Invalid operation: One or more of the used functions must be applied
 on at least one user created table.
```

カタログクエリの例

次のクエリは、カタログテーブルのクエリを実行して Amazon Redshift データベースに関する有益な情報を取得できるいくつかの方法を示しています。

テーブル ID、データベース名、スキーマ名、テーブル名の参照

次のビュー定義は、STV_TBL_PERM システムテーブルを PG_CLASS、PG_NAMESPACE、および PG_DATABASE システムカタログテーブルと統合し、テーブル ID、データベース名、スキーマ名、テーブル名を返します。

```
create view tables_vw as
select distinct(stv_tbl_perm.id) table_id
,trim(pg_database.datname)   db_name
,trim(pg_namespace.nspname)   schema_name
,trim(pg_class.relname)   table_name
from stv_tbl_perm
join pg_class on pg_class.oid = stv_tbl_perm.id
join pg_namespace on pg_namespace.oid = pg_class.relnamespace
join pg_database on pg_database.oid = stv_tbl_perm.db_id;
```

次の例は、テーブル ID 117855 の情報を返します。

```
select * from tables_vw where table_id = 117855;
```

```
table_id | db_name   | schema_name | table_name
---------+-----------+-------------+-----------
  117855 |       dev | public      | customer
```

Amazon Redshift テーブルごとの列数のリスト

次のクエリは、カタログテーブルを結合して、各 Amazon Redshift テーブルに含まれる列の数を調べます。Amazon Redshift テーブル名は、PG_TABLES と STV_TBL_PERM の両方に保存されます。可能であれば、PG_TABLES を使用して Amazon Redshift テーブル名を返します。

このクエリには Amazon Redshift テーブルは関連しません。

```
select nspname, relname, max(attnum) as num_cols
from pg_attribute a, pg_namespace n, pg_class c
where n.oid = c.relnamespace and  a.attrelid = c.oid
and c.relname not like '%pkey'
and n.nspname not like 'pg%'
and n.nspname not like 'information%'
group by 1, 2
order by 1, 2;

nspname | relname  | num_cols
--------+----------+----------
public  | category |        4
public  | date     |        8
public  | event    |        6
public  | listing  |        8
public  | sales    |       10
public  | users    |       18
public  | venue    |        5
(7 rows)
```

データベースのスキーマとテーブルのリスト

次のクエリは STV_TBL_PERM をいくつかの PG テーブルに結合し、TICKIT データベースのテーブルのリストとそのスキーマ名 (NSPNAME 列) を返します。このクエリは、各テーブルの行の合計数も返します。(このクエリは、システムの複数のスキーマに同じテーブル名がある場合に役立ちます。)

```
select datname, nspname, relname, sum(rows) as rows
```

```
from pg_class, pg_namespace, pg_database, stv_tbl_perm
where pg_namespace.oid = relnamespace
and pg_class.oid = stv_tbl_perm.id
and pg_database.oid = stv_tbl_perm.db_id
and datname ='tickit'
group by datname, nspname, relname
order by datname, nspname, relname;

datname | nspname | relname  |  rows
--------+---------+----------+--------
tickit  | public  | category |     11
tickit  | public  | date     |    365
tickit  | public  | event    |   8798
tickit  | public  | listing  | 192497
tickit  | public  | sales    | 172456
tickit  | public  | users    |  49990
tickit  | public  | venue    |    202
(7 rows)
```

テーブル ID、データ型、列名、およびテーブル名のリスト

次のクエリは、各ユーザーテーブルとその列に関するいくつかの情報 (各列のテーブル ID、テーブル名、列名、およびデータ型) をリストします。

```
select distinct attrelid, rtrim(name), attname, typname
from pg_attribute a, pg_type t, stv_tbl_perm p
where t.oid=a.atttypid and a.attrelid=p.id
and a.attrelid between 100100 and 110000
and typname not in('oid','xid','tid','cid')
order by a.attrelid asc, typname, attname;

attrelid |  rtrim   |    attname     |  typname
---------+----------+----------------+-----------
  100133 | users    | likebroadway   | bool
  100133 | users    | likeclassical  | bool
  100133 | users    | likeconcerts   | bool
...
  100137 | venue    | venuestate     | bpchar
  100137 | venue    | venueid        | int2
  100137 | venue    | venueseats     | int4
  100137 | venue    | venuecity      | varchar
...
```

テーブルの各列のデータブロック数のカウント

次のクエリは、STV_BLOCKLIST テーブルを PG_CLASS に結合し、SALES テーブルの列のストレージ情報を返します。

```
select col, count(*)
from stv_blocklist s, pg_class p
where s.tbl=p.oid and relname='sales'
group by col
order by col;

col | count
----+-------
  0 |     4
  1 |     4
  2 |     4
  3 |     4
  4 |     4
  5 |     4
  6 |     4
  7 |     4
  8 |     4
  9 |     8
 10 |     4
 12 |     4
 13 |     8
(13 rows)
```

設定リファレンス

設定ファイルを使用して、環境をカスタマイズできます。Amazon Redshift では、さまざまなパラメータと設定によって、データウェアハウス環境をカスタマイズし最適化することができます。設定リファレンスには、使用可能なクラスタープロパティ、データベースパラメータ、ワークロード管理 (WLM) 設定オプションの概要が記載されています。このリファレンスを参照して、特定の要件に基づいてパフォーマンス、セキュリティ、リソース割り当てを調整できます。以下のリファレンスでは、必要なデータウェアハウス構成を実現するために、これらの設定を変更するための詳細なガイダンスを提供します。

トピック

- max_cursor_result_set_size
- mv_enable_aqmv_for_session
- navigate_super_null_on_error
- parse_super_null_on_error
- pg_federation_repeatable_read
- query_group
- search_path
- spectrum_enable_pseudo_columns
- enable_spectrum_oid
- spectrum_query_maxerror
- statement_timeout
- stored_proc_log_min_messages
- timezone
- use_fips_ssl
- wlm_query_slot_count

サーバー設定の変更

以下の方法でサーバー設定を変更できます。

- SET コマンドを使用して、現在のセッションの期間だけ設定をオーバーライドします。

  次に例を示します。

  ```
  set extra_float_digits to 2;
  ```
- クラスターのパラメータグループ設定を変更します。パラメータグループ設定には、設定可能な追加のパラメータが含まれています。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift パラメータグループ」を参照してください。
- ALTER USER コマンドを使用して、指定したユーザーによって実行されるすべてのセッションに対して、設定パラメータを新しいデフォルト値に設定します。

  ```
  ALTER USER username SET parameter { TO | = } { value | DEFAULT }
  ```

現在のパラメータ設定を見るには、SHOW コマンドを使用します。SET コマンドを使用して設定できるすべての設定値を表示するには、SHOW ALL を使用します。

```
SHOW ALL;

name                         | setting
-----------------------------+--------------
analyze_threshold_percent | 10
datestyle                    | ISO, MDY
extra_float_digits           | 2
query_group                  | default
search_path                  | $user, public
statement_timeout            | 0
timezone                     | UTC
wlm_query_slot_count         | 1
```

> **Note**
>
> 設定パラメータは、データウエアハウス内の接続先のデータベースに適用されます。

analyze_threshold_percent

値 (デフォルトは太字)

10, 0~100.0

説明

テーブルの分析用に、変更された行の割合のしきい値を設定します。処理時間を短縮し、システムの全体的なパフォーマンスを向上させるために、Amazon Redshift は、analyze_threshold_percentの指定よりも変更された行の割合が低いテーブルの ANALYZE を省略します。例えば、テーブルに 100,000,000 行が含まれていて、最後の ANALYZE 以降に 9,000,000 行が変更された場合、変更されたのは行の 10 パーセント未満であるため、デフォルトではこのテーブルはスキップされます。数行のみを変更した場合にテーブルを分析するには、analyze_threshold_percentに任意の小さい数字を設定します。例えば、analyze_threshold_percentを 0.01 に設定すると、少なくとも 10,000 行が変更された場合は 100,000,000 行のテーブルをスキップしません。行が変更されていない場合でもすべてのテーブルを分析するには、analyze_threshold_percentを 0 に設定します。

現在のセッションの `analyze_threshold_percent` パラメータは、SET コマンドを使用することによってのみ、変更できます。パラメータグループでパラメータを変更することはできません。

例

```
set analyze_threshold_percent to 15;
set analyze_threshold_percent to 0.01;
set analyze_threshold_percent to 0;
```

cast_super_null_on_error

値 (デフォルトは太字)

on、off

説明

配列のオブジェクトまたは要素に存在しないメンバーにアクセスしようとすると、クエリがデフォルトの lax モードで実行された場合、Amazon Redshift が NULL 値を返すように指定します。

datashare_break_glass_session_var

値 (デフォルトは太字)

デフォルト値はありません。この値には、以下で説明するように、推奨されないオペレーションが発生した際に Amazon Redshift によって生成される任意の文字列を指定できます。

説明

AWS Data Exchange のデータ共有では、一般的に推奨されていない特定のオペレーションに関するアクセス許可が適用されます。

通常の場合は、DROP DATASHARE あるいは ALTER DATASHARE SET PUBLICACCESSIBLE ステートメントを使用して、AWS Data Exchangeデータ共有の削除または変更を行わないことをお勧めします。パブリックアクセス可能な設定を無効化するために、AWS Data Exchangeデータ共有の削除または変更を許可する場合は、`datashare_break_glass_session_var`変数を 1 回限り有効

な値に設定します。この 1 回限りの値は、Amazon Redshift によって生成され、対象オペレーションでの最初の試行で発生した、エラーメッセージの中に出力されます。

変数を 1 回限り生成される値として設定した後、DROP DATASHARE または ALTER DATASHARE ステートメントを再度実行します。

詳細については、「ALTER DATASHARE の使用に関する注意事項」または「DROP DATASHARE の使用に関する注意事項」を参照してください。

例

```
set datashare_break_glass_session_var to '620c871f890c49';
```

datestyle

値 (デフォルトは太字)

形式の指定 (ISO、Postgres、SQL、German) および年/月/日の順序 (DMY、MDY、YMD)。

- ISO – YYYY-MM-DD HH:MM:SS の datestyle を使用します。
- Postgres — MM-DD HH: MM: SS
- SQL — MM-DD-YYYY HH: MM: SS の datestyle を使用します。
- German — DD-MM-YYYY HH: MM: SS の datestyle を使用します。

説明

日付と時刻の値の表示形式、およびあいまいな日付入力値を解釈するためのルールを設定します。文字列には、個別または一緒に変更できる 2 つのパラメータが含まれています。

例

```
show datestyle;
DateStyle
-----------
ISO, MDY
(1 row)
```

```
set datestyle to 'SQL,DMY';
```

default_geometry_encoding

値 (デフォルトは太字)

1、2

説明

このセッション中に作成された空間ジオメトリを、境界ボックスでエンコードするかどうかを指定するセッション構成です。default_geometry_encoding が 1 に設定されている場合、ジオメトリは境界ボックスでエンコードされません。default_geometry_encoding が 2 に設定されている場合、ジオメトリは境界ボックスでエンコードされます。境界ボックスのサポートの詳細については、「境界ボックス」を参照してください。

describe_field_name_in_uppercase

値 (デフォルトは太字)

オフ (false)、オン (true)

説明

SELECT ステートメントによって返される列名を大文字か小文字かで指定します。このパラメータがオンになっている場合、列名は大文字で返されます。このパラメータがオフの場合、列名は小文字で返されます。Amazon Redshift は、describe_field_name_in_uppercase の設定に関係なく、列名を小文字で保存します。

例

```
set describe_field_name_in_uppercase to on;

show describe_field_name_in_uppercase;

DESCRIBE_FIELD_NAME_IN_UPPERCASE
--------------------------------
```

```
on
```

downcase_delimited_identifier

値 (デフォルトは太字)

on、off

説明

この設定は、廃止されています。代わりに enable_case_sensitive_identifier を使用します。

スーパーパーサーが大文字または大文字と小文字が混在する JSON フィールドを読み読み込むことができます。また、データベース、スキーマ、テーブル、および列で大文字と小文字が混在する名前を持つ、サポート対象の PostgreSQL データベースに対する横串検索のサポートを有効にします。大文字と小文字を区別する識別子を使用するには、このパラメータをオフに設定します。

使用に関する注意事項

- 行レベルのセキュリティ機能や動的データマスキング機能を使用している場合、クラスターまたはワークグループのパラメータグループに downcase_delimited_identifier 値を設定することをお勧めします。これにより、ポリシーを作成してアタッチし、ポリシーが適用されたリレーションをクエリするまでの間、downcase_delimited_identifier の一貫性が保たれます。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。動的データマスキングの詳細については、「動的データマスキング」を参照してください。
- downcase_delimited_identifier をオフに設定してテーブルを作成すると、大文字と小文字を区別する列名を設定できます。downcase_delimited_identifier をオンに設定してテーブルをクエリすると、列名が小文字化されます。これにより、downcase_delimited_identifier がオフに設定されている場合とは異なるクエリ結果が生成される可能性があります。次の例を考えます。

```
SET downcase_delimited_identifier TO off;
--Amazon Redshift preserves case for column names and other identifiers.

--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);
```

```
SELECT * FROM t;

 c | C
---+---
 1 | 2
(1 row)

SET enable_downcase_delimited_identifier TO on;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 動的データマスキングまたは行レベルのセキュリティポリシーが適用されたテーブルをクエリする標準ユーザーは、デフォルトの downcase_delimited_identifier 設定を使用することをお勧めします。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

enable_case_sensitive_identifier

値 (デフォルトは太字)

true、false

説明

データベース、テーブル、および列の名前識別子が大文字と小文字を区別するかどうかを決定する構成値。二重引用符で囲むことで、名前識別子の大文字と小文字を保持できます。`enable_case_sensitive_identifier` を `true` に設定すると、名前識別子の大文字と小文字が区別されます。`enable_case_sensitive_identifier` を `false` に設定すると、名前識別子の大文字と小文字は区別されません。

二重引用符で囲まれたユーザーネームの大文字と小文字は、`enable_case_sensitive_identifier`設定オプションの設定にかかわらず、常に保持されます。

例

次の例は、テーブル名と列名で大文字と小文字を区別する識別子を作成して使用する方法を示しています。

```
-- To create and use case sensitive identifiers
SET enable_case_sensitive_identifier TO true;

-- Create tables and columns with case sensitive identifiers
CREATE TABLE "MixedCasedTable" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable (MixedCasedColumn int);

-- Now query with case sensitive identifiers
SELECT "MixedCasedColumn" FROM "MixedCasedTable";

MixedCasedColumn
------------------
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable;

mixedcasedcolumn
------------------
(0 rows)
```

次の例では、識別子の大文字と小文字が保持されない場合を示しています。

```
-- To not use case sensitive identifiers
RESET enable_case_sensitive_identifier;

-- Mixed case identifiers are lowercased
CREATE TABLE "MixedCasedTable2" ("MixedCasedColumn" int);

CREATE TABLE MixedCasedTable2 (MixedCasedColumn int);

ERROR:  Relation "mixedcasedtable2" already exists


SELECT "MixedCasedColumn" FROM "MixedCasedTable2";

 mixedcasedcolumn
```

```
------------------
(0 rows)


SELECT MixedCasedColumn FROM MixedCasedTable2;

 mixedcasedcolumn
------------------
(0 rows)
```

使用に関する注意事項

- マテリアライズドビューに自動更新を使用している場合は、クラスターまたはワークグループのパラメータグループで enable_case_sensitive_identifier 値を設定することをお勧めします。これにより、マテリアライズドビューが更新されても、enable_case_sensitive_identifier は一定に保たれます。マテリアライズドビューの自動更新については、「マテリアライズドビューの更新」を参照してください。パラメータグループの構成値の設定については、Amazon Redshift 管理ガイドの「Amazon Redshift パラメータグループ」を参照してください。
- 行レベルのセキュリティ機能や動的データマスキング機能を使用している場合、クラスターまたはワークグループのパラメータグループに enable_case_sensitive_identifier 値を設定することをお勧めします。これにより、ポリシーを作成してアタッチし、ポリシーが適用されたリレーションをクエリするまでの間、enable_case_sensitive_identifier の一貫性が保たれます。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。動的データマスキングの詳細については、「動的データマスキング」を参照してください。
- enable_case_sensitive_identifier をオンに設定してテーブルを作成すると、大文字と小文字を区別する列名を設定できます。enable_case_sensitive_identifier をオフに設定してテーブルをクエリすると、列名が小文字化されます。これにより、enable_case_sensitive_identifier がオンに設定されている場合とは異なるクエリ結果が生成される可能性があります。次の例を考えます。

```
SET enable_case_sensitive_identifier TO on;
--Amazon Redshift preserves case for column names and other identifiers.

--Create a table with two columns that are identical except for the case.
CREATE TABLE t ("c" int, "C" int);

INSERT INTO t VALUES (1, 2);
```

```
SELECT * FROM t;

 c | C
---+---
 1 | 2
(1 row)

SET enable_case_sensitive_identifier TO off;
--Amazon Redshift no longer preserves case for column names and other identifiers.

SELECT * FROM t;

 c | c
---+---
 1 | 1
(1 row)
```

- 動的データマスキングまたは行レベルのセキュリティポリシーがアタッチされたテーブルをクエリする標準ユーザーには、デフォルトの enable_case_sensitive_identifier 設定を使用することをお勧めします。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。動的データマスキングの詳細については、「動的データマスキング」を参照してください。

enable_case_sensitive_super_attribute

値 (デフォルトは太字)

true、false

説明

属性名が区切られていない SUPER データ型構造のナビゲートで大文字と小文字を区別するかどうかを決定する設定値。`enable_case_sensitive_super_attribute` を `true` に設定すると、属性名が区切られていない SUPER 型構造のナビゲートでは大文字と小文字が区別されます。値を `false` に設定すると、属性名が区切られていない SUPER 型構造のナビゲートでは大文字と小文字が区別されません。

属性名を二重引用符で囲んで、`enable_case_sensitive_identifier` を `true` に設定すると、`enable_case_sensitive_super_attribute` 構成オプションの設定にかかわらず、大文字と小文字は常に保持されます。

enable_case_sensitive_super_attribute は、SUPER データ型の列にのみ適用されます。他のすべての列については、代わりに enable_case_sensitive_identifier を使用することを検討してください。

SUPER データ型の詳細については、「SUPER タイプ」と「Amazon Redshift の半構造化データ」を参照してください。

例

次の例は、enable_case_sensitive_super_attribute が有効な場合と無効な場合に、SUPER 値を選択した結果を示しています。

```
--Create a table with a SUPER column.
CREATE TABLE tbl (col SUPER);

--Insert values.
INSERT INTO tbl VALUES (json_parse('{
 "A": "A", "a": "a"
}'));

SET enable_case_sensitive_super_attribute TO ON;

SELECT col.A FROM tbl;
  a
-----
 "A"
(1 row)

SELECT col.a FROM tbl;
  a
-----
 "a"
(1 row)

SET enable_case_sensitive_super_attribute TO OFF;

SELECT col.A FROM tbl;
  a
-----
 "a"
(1 row)

SELECT col.a FROM tbl;
```

```
  a
-----
 "a"
(1 row)
```

使用に関する注意事項

- ビューとマテリアライズドビューは、作成時の enable_case_sensitive_super_attribute の値に従います。レイトバインディングビュー、ストアドプロシージャ、およびユーザー定義関数は、クエリ時の enable_case_sensitive_super_attribute の値に従います。
- マテリアライズドビューに自動更新を使用している場合は、クラスターまたはワークグループのパラメータグループで enable_case_sensitive_identifier value を設定することをお勧めします。これにより、マテリアライズドビューが更新されても、enable_case_sensitive_identifier は一定に保たれます。マテリアライズドビューの自動更新については、「マテリアライズドビューの更新」を参照してください。パラメータグループの構成値の設定については、Amazon Redshift 管理ガイドの「Amazon Redshift パラメータグループ」を参照してください。
- ステートメント結果内の列名は、enable_case_sensitive_super_attribute の値に関係なく、常に小文字です。列名でも大文字と小文字を区別するには、enable_case_sensitive_identifier を有効にします。
- 行レベルのセキュリティポリシーがアタッチされたテーブルをクエリする標準ユーザーには、デフォルトの enable_case_sensitive_identifier 設定を使用することをお勧めします。行レベルのセキュリティの詳細については、「行レベルのセキュリティ」を参照してください。

enable_numeric_rounding

値 (デフォルトは太字)

on (true)、off (false)

説明

数値の四捨五入を使用するかどうかを指定します。enable_numeric_rounding が on である場合、Amazon Redshift は、数値を整数や 10 進数などの他の数値型にキャストするとき、四捨五入します。enable_numeric_rounding が off である場合、Amazon Redshift は、数値を他の数値型にキャストするとき、数値を切り捨てます。数値型の詳細については、「数値型」を参照してください。

例

```
--Create a table and insert the numeric value 1.5 into it.
CREATE TABLE t (a numeric(10, 2));

INSERT INTO t VALUES (1.5);

SET enable_numeric_rounding to ON;
--Amazon Redshift now rounds NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 2
(1 row)


SELECT a::decimal(10, 0) FROM t;

 a
---
 2
(1 row)


 SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
(1 row)


SET enable_numeric_rounding to OFF;
--Amazon Redshift now truncates NUMERIC values when casting to other numeric types.

SELECT a::int FROM t;

 a
---
 1
(1 row)
```

```
SELECT a::decimal(10, 0) FROM t;

 a
---
 1
(1 row)


SELECT a::decimal(10, 5) FROM t;

    a
---------
 1.50000
(1 row)
```

enable_result_cache_for_session

値 (デフォルトは太字)

on (true)、off (false)

説明

クエリ結果のキャッシュを使用するかどうかを指定します。`enable_result_cache_for_session` が on の場合、Amazon Redshift は、クエリ結果の有効なキャッシュ済みコピーを、クエリが送信された際に確認します。結果のキャッシュに一致するものが見つかった場合、Amazon Redshift はキャッシュ済みの結果を使用し、クエリは実行しません。`enable_result_cache_for_session` が `off` の場合、Amazon Redshift は結果のキャッシュを無視し、クエリの送信時にそれらのすべてを実行します。

例

```
SET enable_result_cache_for_session TO off;
--Amazon Redshift now ignores the results cache
```

enable_vacuum_boost

値 (デフォルトは太字)

False、True

説明

セッションで実行するすべての VACUUM コマンドに対して vacuum boost オプションを有効にするかどうかを指定します。`enable_vacuum_boost` が `true` の場合、Amazon Redshift は、セッションのすべての VACUUM コマンドを BOOST オプションを使って実行します。`enable_vacuum_boost` が `false` の場合、Amazon Redshift は、デフォルトで BOOST オプションを使用して実行しません。BOOST オプションの詳細については、「VACUUM」を参照してください。

error_on_nondeterministic_update

値 (デフォルトは太字)

False、True

説明

行ごとに複数の一致を持つ UPDATE クエリがエラーを返すかどうかを指定します。

例

```
SET error_on_nondeterministic_update TO true;

CREATE TABLE t1(x1 int, y1 int);

CREATE TABLE t2(x2 int, y2 int);

INSERT INTO t1 VALUES (1,10), (2,20), (3,30);

INSERT INTO t2 VALUES (2,40), (2,50);

UPDATE t1 SET y1=y2 FROM t2 WHERE x1=x2;

ERROR: Found multiple matches to update the same tuple.
```

extra_float_digits

値 (デフォルトは太字)

0、-15～2

説明

浮動小数点値 (float4 と float8 を含みます) の表示桁数を設定します。値は標準の桁数 (FLT_DIG または DBL_DIG) に追加されます。設定できる最高の値は 2 で、有効桁の一部を含みます。これは、正確に復元する必要がある浮動小数点データを出力する場合に特に便利です。または、負の値を設定して、不要な桁を抑制できます。

例

次の例では、`extra_float_digits` を `-2` に設定します。まず、現在のパラメータ設定を表示します。

```
show all;
      name                      | setting
--------------------------+----------------
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
extra_float_digits        | 2
query_group               | default
search_path               | $user, public
statement_timeout         | 0
timezone                  | UTC
wlm_query_slot_count      | 1
```

次に、新しい値を `-2` に設定します。

```
set extra_float_digits to -2;
```

最後に、更新されたパラメータ設定を表示します。

```
show all;
      name                      | setting
--------------------------+----------------
analyze_threshold_percent | 10
datestyle                 | ISO, MDY
```

```
extra_float_digits          | -2
query_group                 | default
search_path                 | $user, public
statement_timeout           | 0
timezone                    | UTC
wlm_query_slot_count        | 1
```

interval_forbid_composite_literals

値 (デフォルトは太字)

false、true

説明

YEAR TO MONTH と DAY TO SECOND の両方のパートを含む間隔の値を変更するセッション設定。

`interval_forbid_composite_literals` が `true` の場合、YEAR TO MONTH と DAY TO SECOND の両方のパートを持つ間隔が検出されると、エラーが返されます。例えば、次の SQL の場合、INTERVAL DAY TO SECOND には YEAR TO MONTH と DAY TO SECOND の両方のパートが含まれています。

```
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;
ERROR:  Interval Day To Second literal cannot contain year-month parts. Disable the GUC
 interval_forbid_composite_literals to suppress this error and silently discard the
 year-month part.
```

`interval_forbid_composite_literals` が `false` の場合、Amazon Redshift はエラーを抑制し、YEAR TO MONTH パートを INTERVAL DAY TO SECOND 値から切り捨てます。例えば、次の SQL の場合、INTERVAL DAY TO SECOND には YEAR TO MONTH と DAY TO SECOND の両方のパートが含まれています。

```
SET interval_forbid_composite_literals to "false";
SELECT INTERVAL '1 year 1 day' DAY TO SECOND;


 intervald2s
------------------------------
1 days 0 hours 0 mins 0.0 secs
```

json_serialization_enable

値 (デフォルトは太字)

False、True

説明

ORC、JSON、Ion、および Parquet 形式のデータの JSON シリアライゼーション動作を変更するセッション構成。`json_serialization_enable` が `true` の場合、すべての最上位コレクションは自動的に JSON にシリアル化され、VARCHAR (65535) として返されます。複合型以外の列は影響を受けず、シリアル化されません。コレクション列は VARCHAR (65535) としてシリアル化されるため、ネストされたサブフィールドにクエリ構文の一部として (フィルター句で) 直接アクセスすることができなくなりました。`json_serialization_enable` が `false` の場合、トップレベルのコレクションは JSON にシリアル化されません。ネストされた JSON シリアル化の詳細については、「複雑なネストされた JSON のシリアル化」を参照してください。

json_serialization_parse_nested_strings

値 (デフォルトは太字)

False、True

説明

ORC、JSON、Ion、および Parquet 形式のデータの JSON シリアライゼーション動作を変更するセッション構成。`json_serialization_parse_nested_strings` と `json_serialization_enable` の両方が true の場合、複合型 (マップ、構造体、配列など) に保存されている文字列値が解析され、それが有効な JSON である場合は、インラインで直接的に結果に対し書き込まれます。`json_serialization_parse_nested_strings` が false の場合、ネストされた複合型内の文字列は、エスケープされた JSON 文字列としてシリアル化されます。詳細については、「JSON 文字列を含む複合型のシリアル化」を参照してください。

max_concurrency_scaling_clusters

値 (デフォルトは太字)

1、0~10

説明

同時実行スケーリングが有効になっている場合に許可される同時実行スケーリングクラスターの最大数を設定します。同時実行スケーリングがさらに必要な場合は、この値を増やします。この値を小さくすると、同時実行スケーリングクラスターの使用量とそれに伴う課金が減少します。

同時実行スケーリングクラスターの最大数は、調整可能なクォータです。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift クォータ」を参照してください。

max_cursor_result_set_size

値 (デフォルトは太字)

0 (デフォルトは最大値) ~ 14,400,000 MB

説明

`max_cursor_result_set_size` パラメータは使用できなくなりました。カーソル結果セットのサイズの詳細については、カーソルの制約を参照してください。

mv_enable_aqmv_for_session

値 (デフォルトは太字)

true、false

説明

Amazon Redshift がセッションレベルでマテリアライズドビューの自動クエリ書き換えを実行できるかどうかを指定します。

navigate_super_null_on_error

値 (デフォルトは太字)

on、off

説明

配列のオブジェクトまたは要素に存在しないメンバーをナビゲートしようとすると、クエリがデフォルトの lax モードで実行された場合、Amazon Redshift が NULL 値を返すように指定します。

parse_super_null_on_error

値 (デフォルトは太字)

オフ、オン

説明

Amazon Redshift が配列のオブジェクトまたは要素に存在しないメンバーを解析しようとすると、クエリが厳密モードで実行された場合、Amazon Redshift が NULL 値を返すように指定します。

pg_federation_repeatable_read

値 (デフォルトは太字)

true、false

説明

PostgreSQL データベースからの結果のフェデレーティッドクエリトランザクション分離レベルを指定します。

- `pg_federation_repeatable_read` が true の場合、フェデレーティッドトランザクションは REPEATABLE READ 分離レベルのセマンティクスで処理されます。これがデフォルトです。
- `pg_federation_repeatable_read` が false の場合、フェデレーティッドトランザクションは READ COMMITTED 分離レベルのセマンティクスで処理されます。

詳細については、次を参照してください:

- Amazon Redshift でフェデレーティッドデータにアクセスする際の考慮事項.
- 同時書き込み操作を管理する.

例

次のコマンドは、セッションの `pg_federation_repeatable_read` を on に設定します。show コマンドは、設定値の値を表示します。

```
set pg_federation_repeatable_read to on;
```

```
show pg_federation_repeatable_read;

pg_federation_repeatable_read
-----------------------------
on
```

query_group

値 (デフォルトは太字)

デフォルトはありません。任意の文字列を指定できます。

説明

1 つのセッションの間に実行されるクエリのグループに対してユーザー定義のラベルを適用します。このラベルはクエリログにキャプチャされます。STL_QUERY テーブル、STV_INFLIGHT テーブル、および SVL_QLOG ビューの結果を制限するために使用できます。例えば、実行するクエリごとに異なるラベルを適用すると、ID を見なくてもクエリを一意に識別できます。

このパラメータはサーバー設定ファイルには存在せず、実行時に SET コマンドで設定する必要があります。長い文字列をラベルとして使用できますが、STL_QUERY テーブルおよび SVL_QLOG ビューの LABEL 列では 30 文字に切り捨てられます (STV_INFLIGHT では 15 文字)。

次の例では、query_group は **`Monday`** に設定されており、そのラベルで複数のクエリが実行されています。

```
set query_group to 'Monday';
SET
select * from category limit 1;
...
...
select query, pid, substring, elapsed, label
```

```
from svl_qlog where label ='Monday'
order by query;

query | pid  |        substring                     |  elapsed  | label
------+------+--------------------------------------+-----------+--------
789   | 6084 | select * from category limit 1;      | 65468     | Monday
790   | 6084 | select query, trim(label) from ... | 1260327   | Monday
791   | 6084 | select * from svl_qlog where ..      | 2293547   | Monday
792   | 6084 | select count(*) from bigsales;       | 108235617 | Monday
...
```

search_path

値 (デフォルトは太字)

'$user', public, schema_names

既存のスキーマ名のカンマ区切りリスト。'$user' が存在する場合は、SESSION_USERと同じ名前を持つスキーマが置き換えられ、それ以外の場合は無視されます。

説明

オブジェクト (テーブルや関数など) がスキーマコンポーネントなしの簡潔な名前で参照されたときにスキーマを検索する順序を指定します。

- 検索パスは外部スキーマと外部テーブルでサポートされていません。外部テーブルは、外部スキーマによって明示的に修飾される必要があります。
- 特定のターゲットスキーマなしで作成されたオブジェクトは、検索パスにリストされている最初のスキーマに配置されます。検索パスが空の場合、システムはエラーを返します。
- 異なるスキーマに同じ名前のオブジェクトが存在する場合は、検索パスで最初に見つかったものが使用されます。
- 検索パスのどのスキーマにも存在しないオブジェクトは、それを含むスキーマを修飾 (ドット区切り) 名で指定することによってのみ参照できます。
- システムカタログスキーマ pg_catalog は常に検索されます。パスで記述されている場合は、指定されている順序で検索されます。記述されていない場合は、すべてのパス項目の前に検索されます。
- 現在のセッションの一時テーブルスキーマ pg_temp_nnn は、存在する場合は常に検索されます。エイリアス pg_temp を使用して、パスで明示的に指定できます。パスで指定されていない場合

は、最初に検索されます (pg_catalog よりさらに前)。ただし、一時スキーマでは関係名 (テーブル、ビュー) だけが検索されます。関数名は検索されません。

例

次の例は、スキーマ ENTERPRISE を作成し、search_path を新しいスキーマに設定します。

```
create schema enterprise;
set search_path to enterprise;
show search_path;

 search_path
-------------
 enterprise
(1 row)
```

次の例は、スキーマ ENTERPRISE をデフォルトの search_path に追加します。

```
set search_path to '$user', public, enterprise;
show search_path;

         search_path
-----------------------------
 "$user", public, enterprise
(1 row)
```

次の例では、テーブル FRONTIER をスキーマ ENTERPRISE に追加します。

```
create table enterprise.frontier (c1 int);
```

テーブル PUBLIC.FRONTIER が同じデータベース内で作成され、ユーザーがクエリでスキーマ名を指定しない場合は、PUBLIC.FRONTIER が ENTERPRISE.FRONTIER より優先されます。

```
create table public.frontier(c1 int);
insert into enterprise.frontier values(1);
select * from frontier;

frontier
----
(0 rows)
```

```
select * from enterprise.frontier;

c1
----
1
(1 row)
```

spectrum_enable_pseudo_columns

値 (デフォルトは太字)

true、false

説明

セッションの疑似列の作成を無効にするには、`spectrum_enable_pseudo_columns`設定パラメータを `false` に設定します。

例

次のコマンドはセッションの疑似列の作成を無効にします。

```
set spectrum_enable_pseudo_columns to false;
```

enable_spectrum_oid

値 (デフォルトは太字)

true、false

説明

`enable_spectrum_oid` 設定パラメータを `false` に設定して、`$spectrum_oid` 擬似列のみを無効にすることもできます。

例

次のコマンドは `enable_spectrum_oid` 設定パラメータを `false` に設定して、`$spectrum_oid` 擬似列を無効にします。

```
set enable_spectrum_oid to false;
```

spectrum_query_maxerror

値 (デフォルトは太字)

-1、整数

説明

整数を指定して、クエリをキャンセルする前に許容するエラーの最大数を示すことができます。負の値を指定すると、エラーデータ処理の最大数がオフになります。結果は SVL_SPECTRUM_SCAN_ERROR にログされます。

例

以下の例では、余剰文字と無効な文字が含まれる ORC データを想定しています。`my_string` の列定義は 3 文字の長さを指定します。以下はこの例のサンプルデータです。

```
my_string
---------------
abcdef
gh�
ab
```

以下のコマンドは、エラーの最大数を 1 に設定し、クエリを実行します。

```
set spectrum_query_maxerror to 1;
SELECT my_string FROM orc_data;
```

クエリが停止し、結果が SVL_SPECTRUM_SCAN_ERROR にログされます。

statement_timeout

値 (デフォルトは太字)

0 (制限オフ) 、x ミリ秒

説明

指定されたミリ秒数以上かかっているステートメントを停止します。

`statement_timeout` 値は、Amazon Redshift によって終了されるまでにクエリを実行できる最大時間です。この時間には、計画、ワークロード管理 (WLM) でのキューイング、および実行時間が含まれます。この時間を、WLM のタイムアウト (max_execution_time) および実行時間のみが含まれる QMR (query_execution_time) と比較します。

WLM タイムアウト (max_execution_time) も WLM 設定の一部として指定されている場合、statement_timeout および max_execution_time の低い方が使用されます。詳細については、「WLM タイムアウト」を参照してください。

例

以下のクエリは 1 ミリ秒より長くかかっているので、タイムアウトしてキャンセルされます。

```
set statement_timeout = 1;

select * from listing where listid>5000;
ERROR:  Query (150) canceled on user's request
```

stored_proc_log_min_messages

値 (デフォルトは太字)

LOG、INFO、NOTICE、WARNING、EXCEPTION

説明

発生したストアドプロシージャメッセージの最小ログ記録レベルを指定します。指定したレベル以上のメッセージがログに記録されます。デフォルトは LOG です (すべてのメッセージがログに記録されます)。ログレベルの順序は、以下のように最大から最小になります。

1. EXCEPTION
2. WARNING
3. NOTICE
4. INFO
5. LOG

例えば、「NOTICE」の値を指定すると、メッセージは NOTICE、WARNING、EXCEPTION についてのみログに記録されます。

timezone

値 (デフォルトは太字)

UTC、タイムゾーン

構文

```
SET timezone { TO | = } [ time_zone | DEFAULT ]

SET time zone [ time_zone | DEFAULT ]
```

説明

現在のセッションのタイムゾーンを設定します。タイムゾーンは、協定世界時 (UTC) またはタイムゾーン名からのオフセットにすることができます。

> **Note**
>
> クラスターパラメータグループを使用して `timezone` 設定パラメータを設定することはできません。タイムゾーンは、SET コマンドを使用して現在のセッションにのみ設定できます。指定のデータベースユーザーが実行するすべてのセッションのタイムゾーンを設定するには、ALTER USERコマンドを使用します。ALTER USER … SET TIMEZONE は、現在のセッションではなく後のセッションのタイムゾーンを変更します。

`SET timezone` または `TO` を含む = (1 語) コマンドを使用してタイムゾーンを設定する場合、以下に示すように、time_zone をタイムゾーン名、POSIX スタイル形式のオフセット、または ISO-8601 形式のオフセットとして指定できます。

```
SET timezone { TO | = } time_zone
```

`TO` または = を指定せずに SET タイムゾーンコマンドを使用してタイムゾーンを設定する場合、以下に示すように、time_zone を INTERVAL と、タイムゾーン名、POSIX スタイル形式のオフセット、または ISO-8601 形式のオフセットとして指定できます。

```
SET time zone time_zone
```

タイムゾーン形式

Amazon Redshift では、以下のタイムゾーン形式がサポートされます。

- タイムゾーンの名前
- INTERVAL
- POSIX スタイルのタイムゾーン指定
- ISO-8601 オフセット

タイムゾーンの省略形 (PST や PDT など) は、UTC からの固定オフセットとして定義されており、夏時間ルールは含まれていないため、SET コマンドではタイムゾーンの省略形がサポートされていません。

タイムゾーン形式の詳細については、以下を参照してください。

タイムゾーン名 – America/New_York などのフルタイムゾーン名。フルタイムゾーン名には、夏時間ルールを含めることができます。

タイムゾーン名の例を以下に示します。

- Etc/Greenwich
- America/New_York
- CST6CDT
- GB

> **Note**
> EST、MST、NZ、UCT などの多くのタイムゾーン名も省略形です。

有効なタイムゾーン名のリストを表示するには、次のコマンドを実行します。

```
select pg_timezone_names();
```

INTERVAL – UTC からのオフセット。例えば、PST は -8:00、つまり -8 時間です。

INTERVAL タイムゾーンオフセットの例を以下に示します。

- –8:00
- -8 時間
- 30 分

POSIX スタイルの形式 – STDoffset または STDoffsetDST 形式でのタイムゾーン指定。STD はタイムゾーンの省略形、offset は UTC から西方向への時間数の数値オフセット、DST はオプションの夏時間ゾーン省略形です。夏時間は、特定のオフセットより 1 時間早いことを前提としています。

POSIX スタイルのタイムゾーン形式では、グリニッジから西方向に正のオフセットが使用されます。これは、グリニッジから東方向に正のオフセットを使用する ISO-8601 規則とは対照的です。

POSIX スタイルのタイムゾーンの例を以下に示します。

- PST8
- PST8PDT
- EST5
- EST5EDT

> **Note**
>
> Amazon Redshift は、POSIX スタイルのタイムゾーン仕様を検証しないため、タイムゾーンを無効な値に設定する可能性があります。例えば、次のコマンドはタイムゾーンを無効な値に設定しますが、エラーを返しません。
>
> ```
> set timezone to ‘xxx36’;
> ```

ISO-8601 オフセット – ±[hh]:[mm] 形式の UTC からのオフセット。

ISO-8601 オフセットの例を以下に示します。

- -8:00
- +7:30

例

次の例では、現在のセッションのタイムゾーンを New York に設定します。

```
set timezone = 'America/New_York';
```

次の例では、現在のセッションのタイムゾーンを UTC–8 (PST) に設定します。

```
set timezone to '-8:00';
```

次の例では、INTERVAL を使用してタイムゾーンを PST に設定します。

```
set timezone interval '-8 hours'
```

次の例では、現在のセッションのタイムゾーンをシステムデフォルトのタイムゾーン (UTC) に設定します。

```
set timezone to default;
```

データベースユーザーのタイムゾーンを設定するには、ALTER USER … SET ステートメントを使用します。次の例では、dbuser のタイムゾーンを New York に設定します。新しい値は、後のすべてのセッションのユーザーに対して維持されます。

```
ALTER USER dbuser SET timezone to 'America/New_York';
```

use_fips_ssl

値 (デフォルトは太字)

true、false

説明

FIPS 準拠の SSL モードを使用するかどうかを指定するパラメータグループ値。`use_fips_ssl` が `true` の場合は、FIPS 準拠の SSL モードが使用されます。`use_fips_ssl` が `false` の場合、FIPS 準拠の SSL モードは使用されません。詳細については、「Amazon Redshift 管理ガイド」の「接続のセキュリティオプションを設定する」を参照してください。

Amazon Redshift でプロビジョニングされたクラスターのパラメータを設定するには、「Amazon Redshift 管理ガイド」の「パラメータグループについて」を参照してください。Redshift Serverless のパラメータを設定するには、「Amazon Redshift 管理ガイド」の「Amazon Redshift Serverless への FIPS 準拠の SSL 接続の設定」、および「Redshift Serverless API リファレンス」の「CreateWorkgroup」または「UpdateWorkgroup」を参照してください。

wlm_query_slot_count

値 (デフォルトは太字)

～50 (1、1サービスクラスの使用可能なスロットの数 (同時実行レベル) を超えることはできません)

説明

クエリが使用するクエリスロットの数を設定します。

ワークロード管理 (WLM) は、キューに設定された同時実行レベルに従って、サービスクラスのスロットを予約します。例えば、同時実行レベルが 5 に設定されている場合、サービスクラスのスロット数は 5 です。WLM は、サービスクラスが使用できるメモリを各スロットに対して等しく割り当てます。詳細については、「ワークロード管理」を参照してください。

> **Note**
>
> wlm_query_slot_count の値がサービスクラスで使用可能なスロットの数 (同時実行レベル) より大きい場合、クエリは失敗します。エラーが発生した場合は、wlm_query_slot_count を許容される値まで減らします。

割り当てられているメモリの量によってパフォーマンスが大きな影響を受ける処理 (例えば Vacuum など) の場合、wlm_query_slot_count の値を増やすとパフォーマンスが向上することがあります。特に、Vacuum コマンドの処理が遅い場合には、SVV_VACUUM_SUMMARY ビュー内で対応しているレコードを調べます。SVV_VACUUM_SUMMARY ビューで sort_partitions および merge_increments の値が高い (100 近く、または 100 より高い) 場合は、そのテーブルに対して Vacuum を次に実行するときに、wlm_query_slot_count の値を増やしてみてください。

wlm_query_slot_count の値を増やすと、実行できる同時実行クエリの数が制限されます。例えば、サービスクラスの同時実行レベルが 5 で、wlm_query_slot_count が 3 に設定されているものとします。wlm_query_slot_count が 3 に設定されているセッション内で 1 つのクエリが実行されている間

は、このクエリに加えて最大 2 個の同時クエリを同じサービスクラス内で実行できます。その後のクエリは、現在実行中のクエリが完了してスロットが解放されるまで、キューで待機しています。

例

現在のセッションの期間に対して wlm_query_slot_count の値を設定するには、SET コマンドを使用します。

```
set wlm_query_slot_count to 3;
```

ドキュメント履歴

> **Note**
>
> Amazon Redshift の新機能の説明については、「最新情報」を参照してください。

次の表に、2018 年 5 月以前の「Amazon Redshift デベロッパーガイド」における重要なドキュメントの変更点を示します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

API バージョン: 2012-12-01

「Amazon Redshift 管理ガイド」の変更点一覧については、「Amazon Redshift 管理ガイドのドキュメント履歴」を参照してください。

新機能の詳細については、修正点のリストや、各リリースに関連付けられたクラスターバージョン番号も含めて、クラスターバージョンの履歴を参照してください。

変更	説明	日付
空間 3D および 4D ジオメトリと新しい空間関数のサポート	追加の空間関数の使用が可能になり、一部の関数に 3D および 4D ジオメトリのサポートが追加されました。	2021 年 8 月 19 日
自動テーブル最適化のための列圧縮エンコードのサポート	テーブルに ENCODE AUTO オプションを指定して、テーブル内のすべて列に対する圧縮エンコードを自動的に管理することができます。	2021 年 8 月 3 日
Amazon Redshift Data API を使用した複数の SQL ステートメントまたはパラメータが含まれる SQL ステートメントのサポート	Amazon Redshift Data API を使用して、複数の SQL ステートメント、またはパラメータが含まれる SQL ステートメン	2021 年 7 月 28 日

	トを実行できるようになりました。	
列レベルの上書きでの大文字と小文字を区別しない照合順序のサポート	CREATE DATABASE ステートメント内で COLLATE 句を使用して、デフォルトの照合順序を指定できるようになりました。	2021 年 6 月 24 日
アカウント間でのデータ共有のサポート	AWS アカウント間でのデータの共有が可能になりました。	2021 年 4 月 30 日
再帰的 CTE による階層データクエリのサポート	SQL で再帰的な共通テーブル式 (CTE) を使用できるようになりました。	2021 年 4 月 29 日
クロスデータベースクエリのサポート	クラスター内のデータベース全体でデータをクエリできるようになりました。	2021 年 3 月 10 日
COPY コマンドと UNLOAD コマンドに対する、きめ細かなアクセスコントロールのサポート	Amazon Redshift クラスター内の特定のユーザーとグループに COPY および UNLOAD コマンドを実行する権限を付与して、よりきめ細かなアクセス制御ポリシーを作成できるようになりました。	2021 年 1 月 12 日
ネイティブ JSON データと半構造化データのサポート	SUPER データ型を定義できるようになりました。	2020 年 12 月 9 日
MySQL への横串検索クエリのサポート	サポートされている MySQL エンジンに横串検索を書き込めるようになりました。	2020 年 12 月 9 日
データ共有のサポート	Amazon Redshift クラスター間でデータを共有できるようになりました。	2020 年 12 月 9 日

自動テーブル最適化のサポート	自動ディストリビューションキーおよびソートキーを定義できるようになりました。	2020 年 12 月 9 日
Amazon Redshift 機械学習のサポート	機械学習 (ML) モデルを作成、トレーニング、デプロイできるようになりました。	2020 年 12 月 8 日
マテリアライズドビューの自動更新およびクエリ書き換えのサポート	自動更新を使用してマテリアライズドビューを最新の状態に保ち、自動書き換えを使用してクエリのパフォーマンスを向上させることができます。	2020 年 11 月 11 日
TIME および TIMETZ データ型のサポート	TIME および TIMETZ データ型を持つテーブルを作成できるようになりました。TIME データ型は、タイムゾーン情報なしで時刻を保存し、TIMETZ は、タイムゾーン情報を含む時刻を保存します。	2020 年 11 月 11 日
Lambda UDF とトークン化のサポート	Lambda UDF を記述して、データの外部トークン化を有効にできるようになりました。	2020 年 10 月 26 日
テーブルで列エンコードの変更のサポート	テーブルの列エンコードを変更できるようになりました。	2020 年 10 月 20 日
データベース間でのクエリのサポート (プレビュー)	Amazon Redshift は、クラスター内のデータベース間でクエリを実行できるようになりました。	2020 年 10 月 15 日

HyperLogLog スケッチのサポート	Amazon Redshift は、HyperLogLogSketches を保存および処理できるようになりました。	2020 年 10 月 2 日
Apache Hudi および Delta Lake のサポート	Redshift Spectrum の外部テーブルの作成が強化されました。	2020 年 9 月 24 日
空間データのクエリ機能強化のサポート	機能強化には、シェープファイルといくつかの新しい空間 SQL 関数のロードが含まれます。	2020 年 9 月 15 日
マテリアライズドビューによって外部テーブルをサポート	Amazon Redshift では、外部データソースを参照するマテリアライズドビューを作成できます。	2020 年 6 月 19 日
外部テーブルへの書き込みのサポート	CREATE EXTERNAL TABLE AS SELECT を実行して新しい外部テーブルに書き込むか、INSERT INTO を実行して既存の外部テーブルにデータを挿入することで、外部テーブルに書き込むことができます。	2020 年 6 月 8 日
スキーマのストレージコントロールのサポート	スキーマのストレージコントロールを管理するコマンドおよびビューの更新。	2020 年 6 月 2 日
フェデレーションクエリの一般提供のサポート	フェデレーティッドクエリを使用したデータのクエリの実行に関する情報を更新しました。	2020 年 4 月 16 日
追加の空間関数のサポート	追加の空間関数の説明を追加しました。	2020 年 4 月 2 日

マテリアライズドビューの一般可用性のサポート	マテリアライズドビューは、クラスターバージョン 1.0.13059 から一般的に利用が可能です。	2020 年 2 月 19 日
列レベル権限のサポート	クラスターバージョン 1.0.13059 以降では、列レベルの権限を使用できます。	2020 年 2 月 19 日
ALTER TABLE	ALTER TABLE コマンドを ALTER DISTSTYLE ALL 句とともに使用すると、テーブルの分散スタイルを変更できます。	2020 年 2 月 11 日
フェデレーションクエリのサポート	更新された CREATE EXTERNAL SCHEMA を使用したフェデレーションクエリについて説明するように、ガイドを更新しました。	2019 年 12 月 3 日
データレイクエクスポートのサポート	UNLOAD コマンドの新しいパラメータについて説明するようにガイドを更新しました。	2019 年 12 月 3 日
空間データのサポート	空間データのサポートについて説明するようにガイドを更新しました。	2019 年 11 月 21 日
新しいコンソールのサポート	新しい Amazon Redshift コンソールについて説明するようにガイドを更新しました。	2019 年 11 月 11 日
自動テーブルソートのサポート	Amazon Redshift は、テーブルデータを自動でソートすることができます。	2019 年 11 月 7 日

VACUUM BOOST オプションのサポート	テーブルをバキューム処理する際は、BOOST オプションを使用することができます。	2019 年 11 月 7 日
デフォルトの IDENTITY 列に関するサポート	デフォルトの IDENTITY 列でテーブルを作成できます。	2019 年 9 月 19 日
AZ64 圧縮エンコードに関するサポート	一部の列を AZ64 圧縮エンコードでエンコードできます。	2019 年 9 月 19 日
クエリ優先度に関するサポート	自動 WLM キューのクエリ優先度を設定できます。	2019 年 8 月 22 日
AWS Lake Formation のサポート	Lake Formation データカタログは、Amazon Redshift Spectrum で使用できます。	2019 年 8 月 8 日
COMPUPDATE PRESET	COPY コマンドを COMPUPDATE PRESET と共に使用して、Amazon Redshift で圧縮エンコードを選択できます。	2019 年 6 月 13 日
ALTER COLUMN	ALTER TABLE コマンドと ALTER COLUMN を併用して、VARCHAR 列のサイズを大きくできます。	2019 年 5 月 22 日
ストアドプロシージャのサポート	PL/pgSQL ストアドプロシージャを Amazon Redshift で定義できます。	2019 年 4 月 24 日
自動ワークロード管理 (WLM) 設定のサポート	Amazon Redshift を自動 WLM で実行できます。	2019 年 4 月 24 日

Zstandard に対する UNLOAD	UNLOAD コマンドを使用して、Amazon S3 にアンロードされたテキストファイルおよびコンマ区切り値 (CSV) ファイルに Zstandard 圧縮を適用できます。	2019 年 4 月 3 日
同時実行スケーリング	同時実行スケーリングが有効になっていると、同時読み込みクエリの増加を処理する必要がある場合、Amazon Redshift は自動的に追加のクラスター容量を追加します。	2019 年 3 月 21 日
UNLOAD から CSV	UNLOAD コマンドを使用して、CSV テキスト形式のファイルにアンロードできます。	2019 年 3 月 13 日
AUTO 分散スタイル	自動分散を有効にするには、CREATE TABLE ステートメントを使用して AUTO 分散スタイルを指定できます。自動ディストリビューションを有効にすると、Amazon Redshift はテーブルデータに基づいて最適なディストリビューションスタイルを割り当てます。分散の変更は、数秒内にバックグラウンドで発生します。	2019 年 1 月 23 日

Parquet からの COPY が SMALLINT をサポート	COPY が、Parquet 形式のファイルから SMALLINT データ型を使用する列へのロードをサポートするようになりました。 詳細については、「列データ形式の COPY」を参照してください。	2019 年 1 月 2 日
DROP EXTERNAL DATABASE	外部データベースは、DROP SCHEMA コマンドで DROP EXTERNAL DATABASE 句を含めることで、削除できます。	2018 年 12 月 3 日
リージョン間での UNLOAD	REGION パラメータを指定することで、別の AWS リージョンにある Amazon S3 バケットに UNLOAD できます。	2018 年 10 月 31 日
自動バキューム削除	Amazon Redshift はバックグラウンドで自動的に VACUUM DELETE オペレーションを実行するため、DELETE ONLY vacuum の実行はほとんど必要ありません。Amazon Redshift では、負荷が軽減されているときに VACUUM DELETE を実行するようにスケジュールし、負荷が高いときにはオペレーションを一時停止します。	2018 年 10 月 31 日

自動分散	CREATE TABLE ステートメントでディストリビューションスタイルを指定しない場合は、Amazon Redshift はテーブルデータに基づいて最適なディストリビューションスタイルを割り当てます。分散の変更は、数秒内にバックグラウンドで発生します。	2018 年 10 月 31 日
AWS Glue Data Catalog でのきめ細かなアクセスコントロール	AWS Glue Data Catalog に保存されているデータへのアクセスレベルを指定できるようになりました。	2018 年 10 月 15 日
データ型を使用した UNLOAD	UNLOAD コマンドで MANIFEST VERBOSE オプションを指定し、列、ファイルサイズ、行数の名前とデータ型を含め、マニフェストファイルにメタデータを追加できます。	2018 年 10 月 10 日
単一の ALTER TABLE ステートメントを使用して複数のパーティションを追加する	Redshift Spectrum 外部テーブルには、複数の PARTITION 句を単一の ALTER TABLE ADD ステートメントに組み合わせることができます。詳細については、「外部テーブルの変更方法の例」を参照してください。	2018 年 10 月 10 日
ヘッダーを使用した UNLOAD	UNLOAD コマンドで HEADER オプションを指定し、各アウトプットファイルの上部に列名が含まれるヘッダー行を追加できます。	2018 年 9 月 19 日

新しいシステムテーブルおよびビュー	SVL_S3Retries、SVL_USER_INFO、および STL_DISK_FULL_DIAG ドキュメントを追加しました。	2018 年 8 月 31 日
Amazon Redshift Spectrum のネストデータのサポート	Amazon Redshift Spectrum テーブルに保存されているネストデータにクエリを実行できるようになりました。詳細については、「チュートリアル: Amazon Redshift Spectrum を使用したネストデータのサポート」を参照してください。	2018 年 8 月 8 日
SQA はデフォルトでオン	ショートクエリアクセラレーション (SQA) は、すべての新しいクラスターにおいて、デフォルトで有効になりました。SQA では、Machine Learning を使用しており、パフォーマンスの向上と結果生成の高速化を実現し、クエリの実行時間が予測しやすくなりました。詳細については、「ショートクエリアクセラレーション」を参照してください。	2018 年 8 月 8 日

Amazon Redshift Advisor	クラスターのパフォーマンスを高め、Amazon Redshift Advisor から運用コストを下げる方法に関する、調整された推奨事項を取得できるようになりました。詳細については、「Amazon Redshift Advisor」を参照してください。	2018 年 7 月 26 日
即時のエイリアス参照	定義後、即時にエイリアス式を参照できるようになりました。詳細については、「SELECT リスト」を参照してください。	2018 年 7 月 18 日
外部テーブルの作成時に圧縮タイプを指定します	Amazon Redshift Spectrum で外部テーブルを作成するときに圧縮タイプを指定できるようになりました。詳細については、「外部テーブルの作成」を参照してください。	2018 年 27 月 6 日
PG_LAST_UNLOAD_ID	新しいシステム情報関数 PG_LAST_UNLOAD_ID 用に追加されたドキュメント。詳細については、「PG_LAST_UNLOAD_ID」を参照してください。	2018 年 27 月 6 日
ALTER TABLE RENAME COLUMN	ALTER TABLE で外部テーブル用の名前変更列がサポートされるようになりました。詳細については、「外部テーブルの変更方法の例」を参照してください。	2018 年 7 月 6 日

以前の更新

次の表に、2018 年 6 月以前の「Amazon Redshift デベロッパーガイド」の各リリースにおける重要な変更点を示します。

変更	説明	変更日
Parquet からの COPY に SMALLINT が含まれる	COPY が、Parquet 形式のファイルから SMALLINT データ型を使用する列へのロードをサポートするようになりました。 詳細については、「列データ形式の COPY」を参照してください。	2019 年 1 月 2 日
列形式の COPY	COPY では、Parquet および ORC 列データ形式を使用する、Amazon S3 上のファイルのロードがサポートされるようになりました。詳細については、「列データ形式の COPY」を参照してください。	2018 年 5 月 17 日
SQA の動的最大実行時間	ワークロード管理 (WLM) は、デフォルトで、クラスターのワークロードの分析に基づいて、ショートクエリアクセラレーション (SQA) 最大実行時間の値が動的に割り当てられるようになりました。詳細については、「ショートクエリの最大実行時間」を参照してください。	2018 年 5 月 17 日
STL_LOAD_COMMITS の新しい列	STL_LOAD_COMMITS システムテーブルに新しい列 (`file_format`) が追加されました。	2018 年 5 月 10 日
STL_HASHJOIN および他のシステムログテーブルの新しい列	STL_HASHJOIN システムテーブルに新しい 3 つの列 (`hash_segment` 、`hash_step` 、`checksum`) が追加されました。また、`checksum` は STL_MERGEJOIN、STL_NESTLOOP、STL_HASH、STL_SCAN、STL_SORT、STL_LIMIT、および STL_PROJECT にも追加されました。	2018 年 5 月 17 日
STL_AGGR の新しい列	STL_AGGR システムテーブルに新しい 2 つの列 (`resizes` と `flushable`) が追加されました。	2018 年 4 月 19 日

変更	説明	変更日
REGEX 関数の新しいオプション	REGEXP_INSTR と REGEXP_SUBSTR 関数で、どの occurence マッチを使用するか、そして大文字と小文字を区別するか指定できるようになりました。また、REGEXP_INSTR でもマッチの最初の文字位置を戻したり、マッチの最後に続く最初の文字位置を指定することができます。	2018 年 3 月 22 日
システムテーブルの新しい列	tombstonedblocks、tossedblocks、batched_by columns が STL_COMMIT_STATS のシステムテーブルに追加されました。localslice 列は STV_SLICES システムビューに追加されました。	2018 年 3 月 22 日
外部テーブルでの列の追加と削除	ALTER TABLE が Amazon Redshift Spectrum 外部テーブルの ADD COLUMN と DROP COLUMN をサポートするようになりました。	2018 年 3 月 22 日
Redshift Spectrum 用の新しい AWS リージョン	Redshift Spectrum がムンバイ、サンパウロの各リージョンで利用可能になりました。サポートされているリージョンのリストについては「Amazon Redshift Spectrum リージョン」を参照してください。	2018 年 3 月 22 日
テーブルの制限値が 20,000 になりました	8xlarge クラスターノードタイプのテーブルの最大数が 20,000 になりました。large と xlarge のノードタイプの最大数は 9,900 です。詳細については、「制限とクォータ」を参照してください。	2018 年 3 月 13 日
Redshift Spectrum が JSON および Ion をサポート	Redshift Spectrum を使用して、JSON または Ion データ形式のスカラーデータのファイルを参照できます。詳細については、「CREATE EXTERNAL TABLE」を参照してください。	2018 年 2 月 26 日

変更	説明	変更日
Redshift Spectrum の IAM ロールの連鎖	AWS Identity and Access Management (IAM) ロールを連結させて、クラスターにアタッチされていない他のロール (他の AWS アカウントに属するロールなど) を、そのクラスターに引き受けさせることができます。詳細については、「Amazon Redshift Spectrum での IAM ロールの連鎖」を参照してください。	2018 年 2 月 1 日
IF NOT EXISTS をサポートする ADD PARTITION	ALTER TABLE の ADD PARTITION 句が、IF NOT EXISTS オプションをサポートするようになりました。詳細については、「ALTER TABLE」を参照してください。	2018 年 1 月 11 日
外部テーブルの DATE データ	Redshift Spectrum 外部テーブルが、DATE データ型をサポートするようになりました。詳細については、「CREATE EXTERNAL TABLE」を参照してください。	2018 年 1 月 11 日
Redshift Spectrum 用の新しい AWS リージョン	Redshift Spectrum がシンガポール、シドニー、ソウル、フランクフルトの各リージョンで利用可能になりました。サポートされている AWS リージョンの一覧は、「Amazon Redshift Spectrum リージョン」でご確認ください。	2017 年 11 月 16 日
Amazon Redshift ワークロード管理 (WLM) のショートクエリアクセラレーション	ショートクエリアクセラレーション (SQA) は、実行時間が短い一部のクエリを、実行時間が長いクエリよりも優先します。SQA では実行時間が短いクエリを専用領域で実行します。このため SQA クエリは、実行時間が長いクエリをキューで待機するよう強制されません。SQA によって実行時間が短いクエリの実行開始が早くなり、ユーザーへの結果表示も早くなります。詳細については、「ショートクエリアクセラレーション」を参照してください。	2017 年 11 月 16 日

変更	説明	変更日
ホップされたクエリの WLM による再割り当て	Amazon Redshift ワークロード管理 (WLM) は、ホップされたクエリをキャンセルして再開する代わりに、対象のクエリを新しいキューに再割り当てするようになりました。WLM によるクエリの再割り当てでは、WLM がクエリを新しいキューに移動して実行を継続することで、時間とシステムリソースを節約します。ホップされたクエリが再割り当ての対象ではない場合、そのクエリは再開またはキャンセルされます。詳細については、「WLM クエリキューのホッピング」を参照してください。	2017 年 11 月 16 日
ユーザーによるシステムログへのアクセス	デフォルトでは、ユーザーが表示可能なシステムログテーブルの大部分で、別のユーザーによって生成された行は、通常のユーザーには表示されません。別のユーザーによって生成された行を含む、ユーザーが表示可能なテーブルのすべての行を、通常のユーザーが表示できるようにするには、ALTER USER または CREATE USER を実行し、SYSLOG ACCESS パラメータを UNRESTRICTED に設定します。	2017 年 11 月 16 日
結果のキャッシュ	結果のキャッシュ では、クエリを実行すると Amazon Redshift によって結果がキャッシュされます。再度クエリを実行すると、Amazon Redshift は、クエリ結果の有効なキャッシュ済みコピーを確認します。結果のキャッシュに一致するものが見つかった場合、Amazon Redshift はキャッシュ済みの結果を使用して、クエリは実行しません。結果のキャッシュはデフォルトでオンになっています。結果のキャッシュをオフにするには、enable_result_cache_for_session 設定パラメータを `off` に設定します。	2017 年 11 月 16 日
列メタデータ関数	PG_GET_COLS と PG_GET_LATE_BINDING_VIEW_COLS は、Amazon Redshift のテーブル、ビュー、および遅延バインドビューの列メタデータを返します。	2017 年 11 月 16 日

変更	説明	変更日
CTAS をホップする WLM キュー	Amazon Redshift ワークロード管理 (WLM) で、CREATE TABLE AS (CTAS) ステートメントをホップするクエリキュー、および SELECT ステートメントなどの読み込み専用クエリがサポートされるようになりました。詳細については、「WLM クエリキューのホッピング」を参照してください。	2017 年 10 月 19 日
Amazon Redshift Spectrum のマニフェストファイル	Redshift Spectrum の外部テーブルを作成する際に、Amazon S3 でのデータファイルの場所を示すマニフェストファイルを指定できます。詳細については、「CREATE EXTERNAL TABLE」を参照してください。	2017 年 10 月 19 日
Amazon Redshift Spectrum の新しい AWS リージョン	Redshift Spectrum が欧州 (アイルランド) とアジアパシフィック (東京) の各リージョンで利用可能になりました。サポートされている AWS リージョンの一覧は、「Amazon Redshift Spectrum の制限事項」でご確認ください。	2017 年 10 月 19 日
Amazon Redshift Spectrum がファイル形式を追加	Redshift Spectrum の外部テーブルを、Regex、OpenCSV、および Avro データファイル形式に基づいて作成できるようになりました。詳細については、「CREATE EXTERNAL TABLE」を参照してください。	2017 年 10 月 5 日
Amazon Redshift Spectrum の外部テーブルの疑似列	Redshift Spectrum の外部テーブルで `$path` 疑似列と `$size` 疑似列を選択して Amazon S3 の参照データファイルの場所とサイズを確認できます。詳細については、「疑似列」を参照してください。	2017 年 10 月 5 日
JSON を検証する関数	IS_VALID_JSON 関数と IS_VALID_JSON_ARRAY 関数を使用して有効な JSON 形式をチェックできます。他の JSON 関数では、オプションの `null_if_invalid` 引数を使用できるようになりました。	2017 年 10 月 5 日

変更	説明	変更日
LISTAGG DISTINCT	LISTAGG 集計関数と LISTAGG ウィンドウ関数で DISTINCT 句を使用し、指定した式から重複する値を排除した上で連結できます。	2017 年 10 月 5 日
列名の大文字での表示	SELECT 結果の列名を大文字で表示するには、describe_field_name_in_uppercase 設定パラメータを `true` に設定できます。	2017 年 10 月 5 日
外部テーブルでのヘッダー行のスキップ	Redshift Spectrum データファイルの先頭でヘッダー行をスキップするように `skip.header.line.count` コマンドの CREATE EXTERNAL TABLE プロパティを設定できます。	2017 年 10 月 5 日
行数のスキャン	WLM クエリモニタリングルールでは、scan_row_count メトリクスを使用してスキャンステップの行数を返します。行数は、削除対象としてマークされた行 (非実体の行) をフィルタリングする前およびユーザー定義のクエリフィルタが適用される前に出力された合計行数を表します。詳細については、「Amazon Redshift のクエリモニタリングメトリクスのプロビジョニング」を参照してください。	2017 年 9 月 21 日
SQL ユーザー定義関数	スカラー SQL ユーザー定義関数 (UDF) には、関数が呼び出され、単一の値を返すときに実行される SQL SELECT 句が組み込まれています。詳細については、「スカラー SQL UDF」を参照してください。	2017 年 8 月 31 日

変更	説明	変更日
遅延バインドビュー	遅延バインドビューは、テーブルやユーザー定義関数など、基盤となるデータベースオブジェクトにバインドされていません。その結果、ビューと参照先のオブジェクト間には依存関係がありません。参照先のオブジェクトが存在しない場合でも、ビューを作成できます。依存関係がないため、ビューに影響を与えることなく参照先のオブジェクトを削除または変更できます。Amazon Redshift は、ビューがクエリされるまで依存関係をチェックしません。遅延バインドビューを作成するには、CREATE VIEW ステートメントで WITH NO SCHEMA BINDING 句を指定します。詳細については、「CREATE VIEW」を参照してください。	2017 年 8 月 31 日
OCTET_LENGTH 関数	OCTET_LENGTH は、指定された文字列の長さをバイト数として返します。	2017 年 8 月 18 日
サポートされる ORC と Grok ファイルタイプ	Amazon Redshift Spectrum は、Redshift Spectrum データファイル用に ORC と Grok データ形式をサポートするようになりました。詳細については、「Amazon Redshift Spectrum でクエリ用のデータファイルを作成する」を参照してください。	2017 年 8 月 18 日
RegexSerDe がサポートされるようになりました。	Amazon Redshift Spectrum は、RegexSerDe データ形式をサポートするようになりました。詳細については、「Amazon Redshift Spectrum でクエリ用のデータファイルを作成する」を参照してください。	2017 年 7 月 19 日
新しい列が SVV_TABLES と SVV_COLUMNS に追加されました。	`domain_name` と `remarks` 列が SVV_COLUMNS に追加されました。SVV_TABLES に備考列が追加されました。	2017 年 7 月 19 日

変更	説明	変更日
SVV_TABLES と SVV_COLUMNS システムビュー	SVV_TABLES と SVV_COLUMNS システムビューは、ローカルおよび外部テーブルとビューの列に関する情報とそのほかの詳細を提供します。	2017 年 7 月 7 日
VPC は、Amazon EMR Hive メタストアの Amazon Redshift Spectrum には不要となりました。	Redshift Spectrum は、Amazon EMR Hive メタストアを使用時に、Amazon Redshift クラスターと Amazon EMR クラスターが同じ VPC と 同じサブネットであるべき要件を削除しました。詳細については、「Amazon Redshift Spectrum での外部カタログの使用」を参照してください。	2017 年 7 月 7 日
小規模サイズファイルへ UNLOAD	デフォルトでは、UNLOAD は Amazon S3 で 6.2 GB サイズまでの複数のファイルを作成します。小規模なファイルを作成するには、UNLOAD コマンドで MAXFILESIZE を指定します。ファイルの最大サイズは 5 MB から 6.2 GB まで指定できます。詳細については、「UNLOAD」を参照してください。	2017 年 7 月 7 日
TABLE PROPERTIES	CREATE EXTERNAL TABLE または ALTER TABLE に TABLE PROPERTIES numRows パラメータを設定して、テーブル統計を更新してテーブルの行数を反映できるようになりました。	2017 年 6 月 6 日
ANALYZE PREDICATE COLUMNS	時間とクラスターリソースを節約するために、述語として使用される可能性が高い列のみを分析できます。PREDICATE COLUMNS 句を使用して ANALYZE を実行すると、分析操作には、結合、フィルタ条件、または GROUP BY 句で使用された列、またはソートキーや分散キーとして使用される列のみが含まれます。詳細については、「テーブルを分析する」を参照してください。	2017 年 5 月 25 日

変更	説明	変更日
Amazon Redshift Spectrum 用の IAM ポリシー	Redshift Spectrum のみを使用して Amazon S3 バケットへのアクセスを許可するには、ユーザーエージェント `AWS Redshift/Spectrum` のアクセスを許可する条件を含めます。詳細については、「Amazon Redshift Spectrum 用の IAM ポリシー」を参照してください。	2017 年 5 月 25 日
Amazon Redshift Spectrum の再帰スキャン	Redshift Spectrum は、Amazon S3 の指定されたフォルダだけでなくサブフォルダ内のファイルもスキャンするようになりました。詳細については、「Redshift Spectrum 用の外部テーブル」を参照してください。	2017 年 5 月 25 日
クエリのモニタリングルール	WLM クエリモニタリングルールを使用すれば、WLM キューのメトリクスベースのパフォーマンス境界を定義し、クエリがそれらの境界を超えた場合に実行するアクション (ログ、ホップ、または中断) を指定できます。ワークロード管理 (WLM) 構成の一部としてクエリモニタリングルールを定義します。詳細については、「WLM クエリモニタリングルール」を参照してください。	2017 年 4 月 21 日
Amazon Redshift Spectrum	Redshift Spectrum を使用することで、テーブルにデータをロードすることなく、効率的にクエリを実行して Amazon S3 からデータを取得できます。Redshift Spectrum クエリは、大きなデータセットに対して非常に素早く実行されます。これは、Redshift Spectrum が Amazon S3 で直接データファイルをスキャンするためです。処理の多くは Amazon Redshift Spectrum レイヤーで生じ、データの大部分が Amazon S3 に保持されます。複数のクラスターが Amazon S3 で同じデータセットを同時にクエリできます。クラスターごとにデータをコピーする必要はありません。詳細については、「Amazon Redshift Spectrum」を参照してください。	2017 年 4 月 19 日

変更	説明	変更日
Redshift Spectrum をサポートする新しいシステムテーブル	Redshift Spectrum をサポートするために次の新しいシステムビューが追加されています。 • SVL_S3QUERY • SVL_S3QUERY_SUMMARY • SVV_EXTERNAL_COLUMNS • SVV_EXTERNAL_DATABASES • SVV_EXTERNAL_PARTITIONS • SVV_EXTERNAL_TABLES • PG_EXTERNAL_SCHEMA	2017 年 4 月 19 日
APPROXIMATE PERCENTILE_DISC 集計関数	APPROXIMATE PERCENTILE_DISC 集計関数が利用可能になりました。	2017 年 4 月 4 日
KMS でのサーバー側の暗号化	AWS Key Management Service キー (SSE-KMS) を使用したサーバー側の暗号化を使用して、Amazon S3 にデータをアンロードできるようになりました。さらに、COPY で、KMS で暗号化されたデータファイルを Amazon S3 から透過的にロードできるようになりました。詳細については、「UNLOAD」を参照してください。	2017 年 2 月 9 日

変更	説明	変更日
新しい認可構文	IAM_ROLE、MASTER_SYMMETRIC_KEY、ACCESS_KEY_ID、SECRET_ACCESS_KEY、および SESSION_TOKEN の各パラメータを使用して、COPY、UNLOAD、および CREATE LIBRARY コマンドに認可およびアクセス情報を提供できるようになりました。新しい認可構文では、CREDENTIALS パラメータに単一文字列引数を指定するよりも柔軟な選択肢が提供されます。詳細については、「認可パラメータ」を参照してください。	2017 年 2 月 9 日
スキーマの上限が緩和されます	クラスターごとに最大 9,900 個のスキーマを作成できるようになりました。詳細については、「CREATE SCHEMA」を参照してください。	2017 年 2 月 9 日
デフォルトのテーブルのエンコード	CREATE TABLE および ALTER TABLE で、ほとんどの新規列に LZO エンコードが割り当てられるようになりました。ソートキーとして定義されている列、BOOLEAN、REAL、または DOUBLE PRECISION データ型として定義された列、および一時テーブルには、RAW エンコードがデフォルトで割り当てられます。詳細については、「ENCODE」を参照してください。	2017 年 2 月 6 日
ZSTD 圧縮エンコード	Amazon Redshift で ZSTD 列圧縮エンコードがサポートされるようになりました。	2017 年 1 月 19 日
PERCENTILE_CONT および MEDIAN 集計関数	PERCENTILE_CONT および MEDIAN が、ウィンドウ関数だけでなく集計関数としても使用できるようになりました。	2017 年 1 月 19 日

変更	説明	変更日
ユーザー定義関数 (UDF) のユーザーログ記録	Python ロギングモジュールを使用して、ユーザー定義のエラーおよび警告メッセージを UDF 内に作成できます。クエリの実行に続いて、SVL_UDF_LOG システムビューをクエリしてログ記録されたメッセージを取得できます。ユーザー定義メッセージについては、「Python UDF のエラーと警告のログ記録」を参照してください。	2016 年 12 月 8 日
ANALYZE COMPRESSION の予想圧縮率	ANALYZE COMPRESSION コマンドでは、列ごとにディスクスペースの予想圧縮率が報告されるようになりました。詳細については、「ANALYZE COMPRESSION」を参照してください。	2016 年 11 月 10 日
接続制限	ユーザーが同時に開けるデータベース接続の数を制限できるようになりました。また、1 つのデータベースに対する同時接続の数も制限できます。詳細については、CREATE USERおよびCREATE DATABASEを参照してください。	2016 年 11 月 10 日
COPY のソート順序の機能強化	COPY では、ソートキー順序でデータをロードするときに、テーブルのソート済みリージョンに自動的に新しい行が追加されるようになりました。この強化機能を有効にするための具体的な要件については、「ソートキー順序でデータをロードする」を参照してください	2016 年 11 月 10 日
CTAS と圧縮	CREATE TABLE AS (CTAS) では、列のデータ型に基づいて、圧縮エンコーディングが新しいテーブルに自動的に割り当てられるようになりました。詳細については、「列属性とテーブル属性の継承」を参照してください。	2016 年 10 月 28 日

変更	説明	変更日
タイムゾーンデータ型を含むタイムスタンプ	Amazon Redshift では、タイムゾーン (TIMESTAMPTZ) データ型を含むタイムスタンプがサポートされます。さらに、新しいデータ型をサポートするためにいくつかの新機能が追加されています。詳細については、「日付および時刻関数」を参照してください。	2016 年 9 月 29 日
分析のしきい値	ANALYZE オペレーションの処理時間を減らし、全体的なシステムパフォーマンスを向上させるため、最後の ANALYZE コマンドの実行以降に変更された行の割合が、analyze_threshold_percent パラメータで指定された分析のしきい値よりも低い場合、Amazon Redshift はテーブルの分析をスキップします。デフォルトでは、`analyze_threshold_percent` は 10 です。	2016 年 8 月 9 日
新しい STL_RESTARTED_SESSIONS システムテーブル	Amazon Redshift がセッションを再起動すると、STL_RESTARTED_SESSIONS は新しいプロセス ID (PID) と古い PID を記録します。	2016 年 8 月 9 日
日付および時刻関数のドキュメントを更新しました	リンクを含む関数の概要を「日付および時刻関数」に追加し、整合性のために関数のリファレンスを更新しました。	2016 年 6 月 24 日
STL_CONNECTION_LOG の新しい列	STL_CONNECTION_LOG システムテーブルには SSL 接続を追跡する 2 つの新しい列があります。Amazon Redshift テーブルに定期的に監査ログをロードする場合、ターゲットテーブルに sslcompression と sslexpansion という新しい列を追加する必要があります。	2016 年 5 月 5 日
MD5 ハッシュパスワード	パスワードおよびユーザー名の MD5 ハッシュ文字列を指定して、CREATE USER または ALTER USER コマンドのパスワードを指定できます。	2016 年 4 月 21 日

変更	説明	変更日
STV_TBL_PERM の新しい列	backup システムビューの STV_TBL_PERM 列は、テーブルがクラスターのスナップショットに含まれているかどうかを示します。詳細については、「BACKUP」を参照してください。	2016 年 4 月 21 日
バックアップしないテーブル	重要データが含まれない、ステージングテーブルのようなテーブルの場合は、CREATE TABLE または CREATE TABLE AS ステートメントで、BACKUP NO を指定して、自動または手動スナップショットにテーブルを含めることから Amazon Redshift を防ぐことができます。バックアップしないテーブルを使用することで、スナップショットを作成したり、スナップショットから復元したり、Amazon S3 のストレージ領域を削減したりするときに処理時間を短縮できます。	2016 年 4 月 7 日
VACUUM 削除のしきい値	デフォルトでは、VACUUM コマンドは、残りの行の少なくとも 95 パーセントが削除対象としてマークされていない領域を再利用します。そのため、VACUUM では通常、削除された行の 100 パーセントを再利用することに比べて削除フェーズの時間が少なくて済みます。VACUUM コマンドを実行するときに、TO threshold PERCENT パラメータを含めることにより、1 つのテーブルに対するデフォルトのしきい値を変更できます。	2016 年 4 月 7 日
SVV_TRANSACTIONS システムテーブル	SVV_TRANSACTIONS システムビューでは、現在のデータベースのテーブルでロックを保持するトランザクションについての情報が記録されます。	2016 年 4 月 7 日

変更	説明	変更日
IAM ロールを使用して他の AWS リソースにアクセスする	自分のクラスターと他の AWS リソース (Amazon S3、DynamoDB、Amazon EMR、または Amazon EC2 など) との間でデータを移動する場合、クラスターには、リソースにアクセスして必要なアクションを実行するための許可が必要です。COPY、UNLOAD、または CREATE LIBRARY コマンドを使用してアクセスキーペアを提供するための安全な代替方法として、クラスターが認証と認可に使用する IAM ロールを指定できます。詳細については、「ロールベースアクセスコントロール」を参照してください。	2016 年 3 月 29 日
VACUUM ソートしきい値	VACUUM コマンドでは現在、テーブルの行の 95 パーセント以上がすでにソートされているテーブルのソートフェーズをスキップします。VACUUM コマンドを実行する際に、TO threshold PERCENT パラメータを含めることにより、1 つのテーブルに対するデフォルトのソートしきい値を変更できます。	2016 年 3 月 17 日
STL_CONNECTION_LOG の新しい列	STL_CONNECTION_LOG システムテーブルには 3 つの新しい列が追加されました。Amazon Redshift テーブルに定期的に監査ログをロードする場合、ターゲットテーブルに sslversion と sslcipher、mtu という新しい列を追加する必要があります。	2016 年 3 月 17 日
bzip2 圧縮を使用した UNLOAD	bzip2 圧縮を使用した UNLOAD オプションが利用できるようになりました。	2016 年 2 月 8 日
ALTER TABLE APPEND	ALTER TABLE APPEND は、既存のソーステーブルのデータを移動して、ターゲットテーブルに行を追加します。通常、ALTER TABLE APPEND は、データを複製するのではなく移動するため、同様の CREATE TABLE AS または INSERT の INTO オペレーションよりもはるかに高速です。	2016 年 2 月 8 日

変更	説明	変更日
WLM クエリキューのホッピング	ワークロード管理 (WLM) によって読み取り専用クエリ (SELECT ステートメントなど) がキャンセルされた場合、WLM のタイムアウトにより、WLM は次に一致するキューにクエリのルーティングを試みます。詳細については、「WLM クエリキューのホッピング」を参照してください。	2016 年 1 月 7 日
ALTER DEFAULT PRIVILEGES	ALTER DEFAULT PRIVILEGES コマンドを使用して、指定したユーザーによって今後作成されるオブジェクトに対して、デフォルトで適用するアクセス権のセットを定義します。	2015 年 12 月 10 日
bzip2 ファイル圧縮	COPY コマンドは、bzip2 で圧縮されたファイルからのデータのロードをサポートします。	2015 年 12 月 10 日
NULLS FIRST および NULLS LAST	ORDER BY 句で、結果セットで NULL を最初にランク付けするか最後にランク付けするかを指定できます。詳細については、ORDER BY 句およびウィンドウ関数の構文の概要を参照してください。	2015 年 11 月 19 日
CREATE LIBRARY ライブラリの REGION キーワード	UDF ライブラリファイルを保持する Amazon S3 バケットが、Amazon Redshift クラスターと同じ AWS リージョンに存在しない場合は、REGION オプションを使用して、データが置かれているリージョンを指定できます。詳細については、「ライブラリを作成する」を参照してください。	2015 年 11 月 19 日
ユーザー定義のスカラー関数 (UDF)	Python 2.7 の標準ライブラリで Amazon Redshift がサポートするモジュールまたは Python のプログラミング言語に基づいた独自のカスタム UDF により、ユーザー定義のスカラー関数を作成して、非 SQL 処理機能を実行できるようになりました。詳細については、「Amazon Redshift のユーザー定義関数」を参照してください。	2015 年 9 月 11 日

変更	説明	変更日
WLM 設定の動的プロパティ	WLM 設定パラメータは一部のプロパティの動的な適用をサポートするようになりました。他のプロパティは静的なままであり、設定の変更を適用するには、該当するクラスターを再起動する必要があります。詳細については、WLM の動的設定プロパティと静的設定プロパティおよびワークロード管理を参照してください。	2015 年 8 月 3 日
LISTAGG 関数	LISTAGG 関数 と LISTAGG ウィンドウ関数 は、列の一連の値を連結することで作成された文字列を返します。	2015 年 7 月 30 日
廃止されたパラメータ	max_cursor_result_set_size 設定パラメータは廃止されました。カーソルの結果セットのサイズはクラスターノードタイプに基づいて制限されます。詳細については、「カーソルの制約」を参照してください。	2015 年 7 月 24 日
COPY コマンドの改訂版ドキュメント	COPY コマンドリファレンスは、分かりやすくまた検索しやすくなるように大幅に改訂されました。	2015 年 7 月 15 日
Avro 形式からの COPY	COPY コマンドは、SSH を使用して Amazon S3、Amazon EMR、リモートホストからデータファイルを Avro 形式でロードする処理をサポートします。詳細については、AVROおよびAvro の例からのコピーを参照してください。	2015 年 7 月 8 日
STV_STARTUP_RECOVERY_STATE	STV_STARTUP_RECOVERY_STATE システムテーブルは、クラスターの再起動オペレーション中に一時的にロックされているテーブルの状態を記録します。Amazon Redshift は、クラスターの再起動後、古いトランザクションを解決するために処理中のテーブルを一時的にロックします。	2015 年 5 月 25 日

変更	説明	変更日
ORDER BY はランク付け関数に省略可能です。	ORDER BY 句は特定のウィンドウランク付け関数に省略可能になりました。詳細については、「CUME_DIST ウィンドウ関数」、「DENSE_RANK ウィンドウ関数」、「RANK ウィンドウ関数」、「NTILE ウィンドウ関数」、「PERCENT_RANK ウィンドウ関数」、「ROW_NUMBER ウィンドウ関数」を参照してください。	2015 年 5 月 25 日
インターリーブソートキー	インターリーブソートキーはソートキーの各列に等しい重みを割り当てます。デフォルトの複合キーの代わりにインターリーブソートキーを使用すると、制限された述語を 2 番目のソート列で使用するクエリのパフォーマンスが、サイズの大きいテーブルに対しては特に、大幅に向上します。また、インターリーブソートを使用すると、複数のクエリが同じテーブルの異なる列でフィルタリングされるときの全体的なパフォーマンスが向上します。詳細については、ソートキーおよびCREATE TABLEを参照してください。	2015 年 5 月 11 日
「クエリパフォーマンスのチューニング」トピックの改訂	「クエリパフォーマンスのチューニング」が拡張され、クエリパフォーマンスなどの例を分析するクエリが含められました。さらに、このトピックはより明確で詳細になるように改訂されました。「Amazon Redshift クエリの設計のベストプラクティス」には、パフォーマンスを向上させるにクエリを記述する方法についての詳細な情報が記載されています。	2015 年 3 月 23 日
SVL_QUERY_QUEUE_INFO	SVL_QUERY_QUEUE_INFO ビューでは、WLM クエリキューまたはコミットキューで時間がかかったクエリの詳細が示されます。	2015 年 2 月 19 日

変更	説明	変更日
SVV_TABLE_INFO	SVV_TABLE_INFO ビューを使用して、クエリのパフォーマンスに影響する可能性のあるテーブル設計の問題を診断し、対応できます。これには、圧縮エンコード、分散キー、ソートスタイル、データ分散スキュー、テーブルサイズ、および統計情報が含まれます。	2015 年 2 月 19 日
UNLOAD でサーバー側ファイル暗号化機能を使用	UNLOAD コマンドで、自動的に Amazon S3 サーバー側の暗号化 (SSE) 機能が使用されてすべてのアンロードデータファイルが暗号化されるようになりました。サーバー側暗号化機能により、パフォーマンスがほとんど、またはまったく変化することなく、データのセキュリティが一段と強化されます。	2014 年 10 月 31 日
CUME_DIST ウィンドウ関数	CUME_DIST ウィンドウ関数 は、ウィンドウまたはパーティション内の値の累積分布を計算します。	2014 年 10 月 31 日
MONTHS_BETWEEN 関数	MONTHS_BETWEEN 関数 は、2 つの日付の間の月数を算出します。	2014 年 10 月 31 日
NEXT_DAY 関数	NEXT_DAY 関数 は、指定の日付より後に指定の曜日となる最初の日付を返します。	2014 年 10 月 31 日
PERCENT_RANK ウィンドウ関数	PERCENT_RANK ウィンドウ関数 は、指定の行のパーセントランクを計算します。	2014 年 10 月 31 日
RATIO_TO_REPORT ウィンドウ関数	RATIO_TO_REPORT ウィンドウ関数 は、ウィンドウまたはパーティションの値の合計に対する、ある値の比率を計算します。	2014 年 10 月 31 日
TRANSLATE 関数	TRANSLATE 関数 は、任意の式内の指定された文字をすべて、指定された別の文字に置き換えます。	2014 年 10 月 31 日
NVL2 関数	NVL2 関数 は、指定された式を評価すると NULL になるか NOT NULL になるかに基づいて、2 つの値のどちらかを返します。	2014 年 10 月 16 日

変更	説明	変更日
MEDIAN ウィンドウ関数	MEDIAN ウィンドウ関数 は、ウィンドウまたはパーティションの値の範囲について、その中央値を計算します。	2014 年 10 月 16 日
GRANT および REVOKE コマンドの ON ALL TABLES IN SCHEMA schema_name 句	GRANT および REVOKE コマンドが更新され、ON ALL TABLES IN SCHEMA schema_name 句が追加されました。この句を使用すると、1 つのコマンドで、スキーマ内のすべてのテーブルに対する権限を変更できます。	2014 年 10 月 16 日
DROP SCHEMA、DROP TABLE、DROP USER、DROP VIEW コマンドの IF EXISTS 句	DROP SCHEMA、DROP TABLE、DROP USER、DROP VIEW の各コマンドが更新され、IF EXISTS 句が追加されました。この句を使用すると、指定したオブジェクトが存在しない場合、コマンドはエラーで終了するのではなく、何も変更しないでメッセージを返します。	2014 年 10 月 16 日
CREATE SCHEMA および CREATE TABLE コマンドの IF NOT EXISTS 句	CREATE SCHEMA および CREATE TABLE コマンドが更新され、IF NOT EXISTS 句が追加されました。この句を使用すると、指定したオブジェクトが既に存在する場合、コマンドはエラーで終了するのではなく、何も変更しないでメッセージを返します。	2014 年 10 月 16 日
COPY が UTF-16 エンコードをサポート	COPY コマンドが、UTF-8 エンコードに加えて、UTF-16 エンコードを使用するデータファイルからのロードもサポートするようになりました。詳細については、「ENCODING」を参照してください。	2014 年 9 月 29 日
新しいワークロード管理チュートリアル	「チュートリアル: 手動ワークロード管理 (WLM) キューの設定」で、ワークロード管理 (WLM) キューを設定してクエリ処理とリソース割り当てを改善する方法について順を追って説明しています。	2014年 9 月25日

変更	説明	変更日
AES 128 ビット暗号化	COPY コマンドが、Amazon S3 クライアント側暗号化機能を使って暗号化されたデータファイルからロードするときに、AES 256 ビット暗号化に加えて、AES 128 ビット暗号化もサポートするようになりました。詳細については、「暗号化されたデータファイルを Amazon S3 からロードする」を参照してください。	2014 年 9 月 29 日
PG_LAST_UNLOAD_COUNT 関数	PG_LAST_UNLOAD_COUNT 関数は、一番最近の UNLOAD 操作で処理された行数を返します。詳細については、「PG_LAST_UNLOAD_COUNT」を参照してください。	2014 年 9 月 15 日
クエリのトラブルシューティングのセクションの新規追加	「クエリのトラブルシューティング」では、Amazon Redshift クエリで発生する可能性のある一般的な問題と重大な問題を特定し、それらの問題に対処するためのクイックリファレンスを提供しています。	2014 年 7 月 7 日
データのロードのチュートリアルの新規追加	「チュートリアル: Amazon S3 からデータをロードする」では、Amazon S3 バケット内のデータファイルから Amazon Redshift データベースのテーブルにデータをロードするプロセスを最初から最後まで説明しています。	2014 年 7 月 1 日
PERCENTILE_CONT ウィンドウ関数	PERCENTILE_CONT ウィンドウ関数 は、連続型分散モデルを前提とする逆分散関数です。これは、パーセンタイル値とソート仕様を取得し、ソート仕様を基準として、そのパーセンタイル値に該当する補間値を返します。	2014 年 6 月 30 日
PERCENTILE_DISC ウィンドウ関数	PERCENTILE_DISC ウィンドウ関数 は、離散型分散モデルを前提とする逆分散関数です。パーセンタイル値とソート指定を受け取り、データセットから該当する要素を返します。	2014 年 6 月 30 日
GREATEST および LEAST 関数	GREATEST および LEAST 関数 関数は、式のリストから最大値または最小値を返します。	2014 年 6 月 30 日

変更	説明	変更日
リージョン間の COPY	COPY コマンドでは、Amazon Redshift クラスターとは異なるリージョンにある Amazon S3 バケットや Amazon DynamoDB テーブルのデータのロードがサポートされています。詳細については、COPY コマンドのリファレンスの「REGION」を参照してください。	2014 年 6 月 30 日
ベストプラクティスの拡充	Amazon Redshift のベストプラクティス は拡充および再編成され、より見つけやすいようにナビゲーション階層の最上位に移動されました。	2014 年 5 月 28 日
1 つのファイルに対する UNLOAD	UNLOAD コマンドでは、必要に応じて PARALLEL OFF オプションを追加することで、テーブルデータを Amazon S3 上の 1 つのファイルに、順次にアンロードできます。データのサイズがファイルの最大サイズの 6.2 GB よりも大きい場合は、UNLOAD によって追加のファイルが作成されます。	2014 年 5 月 6 日
REGEXP の関数	REGEXP_COUNT、REGEXP_INSTR、REGEXP_REPLACE の関数は、正規表現のパターンマッチングに基づいて文字列を操作します。	2014 年 5 月 6 日
Amazon EMR からの COPY	COPY コマンドは、Amazon EMR クラスターからのデータの直接ロードをサポートします。詳細については、「Amazon EMR からのデータのロード」を参照してください。	2014 年 4 月 18 日
WLM 同時実行の制限の緩和	ユーザー定義クエリキューで最大 50 個のクエリを同時に実行するようにワークロード管理 (WLM) を設定できるようになりました。この緩和により、WLM 設定を変更してシステムのパフォーマンスをより柔軟に管理できます。詳細については、「手動 WLM を実装する」を参照してください。	2014 年 4 月 18 日

変更	説明	変更日
カーソルサイズを管理するための新しい設定パラメータ	`max_cursor_result_set_size` 設定パラメータは、データの最大サイズを MB 単位で定義します。これは、より大きいクエリのカーソル結果セットごとに取得できます。このパラメータ値は、クラスターの同時カーソルの数にも影響するため、クラスターのカーソル数を増減する値を設定できます。 詳細については、このガイドの「DECLARE」と、「Amazon Redshift 管理ガイド」の「カーソル結果セットの最大サイズの設定」を参照してください。	2014 年 3 月 28 日
JSON 形式からの COPY	COPY コマンドは、SSH を使用して Amazon S3 のデータファイルとリモートホストからデータを JSON 形式でロードする処理をサポートします。詳細については、「使用に関する注意事項」の「JSON 形式からの COPY」を参照してください。	2014 年 3 月 25 日
新しいシステムテーブル STL_PLAN_INFO	STL_PLAN_INFO テーブルはクエリプランを参照する別の方法として EXPLAIN コマンドを補完します。	2014 年 3 月 25 日
新しい関数 REGEXP_SUBSTR	REGEXP_SUBSTR 関数 は、正規表現パターンを検索して文字列から抽出した文字を返します。	2014 年 3 月 25 日
STL_COMMIT_STATS の新しい列	STL_COMMIT_STATS テーブルには新しい 2 つの列 (`numxids` と `oldestxid`) が追加されました。	2014 年 3 月 6 日
gzip と lzop に対する SSH からの COPY のサポート	COPY コマンドは、SSH 接続でデータをロードするときに gzip および lzop 圧縮をサポートします。	2014 年 2 月 13 日

変更	説明	変更日
新しい関数	ROW_NUMBER ウィンドウ関数 は、現在の行の番号を返します。STRTOL 関数 は、基数が指定された数の文字列式を、相当する整数値に変換します。PG_CANCEL_BACKEND と PG_TERMINATE_BACKEND は、クエリとセッション接続をキャンセルできるようにします。LAST_DAY 関数は、Oracle との互換性のために追加されています。	2014 年 2 月 13 日
新しいシステムテーブル	STL_COMMIT_STATS システムテーブルは、コミットのさまざまなステージのタイミングやコミットされるブロックの数など、コミットのパフォーマンスに関連するメトリクスを提供します。	2014 年 2 月 13 日
単一ノードクラスターでの FETCH	単一ノードクラスターでカーソルを使用する場合、FETCH コマンドを使用して取得できる行の最大数は 1,000 です。単一ノードクラスターでは、FETCH FORWARD ALL はサポートされません。	2014 年 2 月 13 日
DS_DIST_ALL_INNER 再分散戦略	説明プランの出力の DS_DIST_ALL_INNER は、外部テーブルで DISTSTYLE ALL が使用されているために内部テーブル全体が単一スライスに再分散されたことを示しています。詳細については、結合の種類の例およびクエリプランの評価を参照してください。	2014 年 1 月 13 日
クエリ用の新しいシステムテーブル	Amazon Redshift では、チューニングとトラブルシューティングの目的でクエリ実行を評価するために使用できる新しいシステムテーブルが追加されました。詳細については、「SVL_COMPILE」、「STL_SCAN」、「STL_RETURN」、「STL_SAVE」、「STL_ALERT_EVENT_LOG」を参照してください。	2014 年 1 月 13 日

変更	説明	変更日
単一ノードカーソル	カーソルが単一ノードクラスターでサポートされるようになりました。単一ノードクラスターでは、一度に 2 個のカーソルを開くことができ、結果セットの最大サイズは 32 GB です。単一ノードクラスターでは、ODBC Cache Size パラメータを 1,000 に設定することをお勧めします。詳細については、「DECLARE」を参照してください。	2013 年 12 月 13 日
ALL 分散形式	ALL 分散では、特定の種類のクエリの実行時間を劇的に短縮できます。テーブルで ALL 分散スタイルを使用すると、テーブルのコピーはすべてのノードに分散されます。テーブルは他のすべてのテーブルと効果的にコロケーテッド結合されるため、クエリ実行時に再分散は不要です。ALL 分散は、ストレージ要件とロード時間が増すため、すべてのテーブルに適しているわけではありません。詳細については、「クエリ最適化のためのデータのディストリビューション」を参照してください。	2013 年 11 月 11 日
リモートホストからの COPY	COPY コマンドは、Amazon S3 のデータファイルおよび Amazon DynamoDB テーブルからテーブルをロードするだけでなく、SSH 接続を使用して Amazon EMR クラスター、Amazon EC2 インスタンス、およびその他のリモートホストからテキストデータをロードできます。Amazon Redshift は、複数の同時 SSH 接続を使用して、データの読み込みとロードを並行して行います。詳細については、「リモートホストからデータをロードする」を参照してください。	2013 年 11 月 11 日
WLM メモリの使用率	ワークロード管理 (WLM) 設定でキューごとにメモリの割合を指定することで、ワークロードを分散することができます。詳細については、「手動 WLM を実装する」を参照してください。	2013 年 11 月 11 日

変更	説明	変更日
APPROXIMATE COUNT(DISTINCT)	APPROXIMATE COUNT(DISTINCT) を使用するクエリは、実行がはるかに高速で、相対エラーは約 2% です。APPROXIMATE COUNT(DISTINCT) 関数は HyperLogLog アルゴリズムを使用します。詳細については、「COUNT 関数」を参照してください。	2013 年 11 月 11 日
最近のクエリの詳細を取得する新しい SQL 関数	4 個の新しい SQL 関数は、最近のクエリと COPY コマンドに関する詳細を取得します。この新しい関数により、システムログテーブルに対するクエリを実行しやすくなり、多くの場合、システムテーブルにアクセスしなくても必要な詳細を得られます。詳細については、「PG_BACKEND_PID」、「PG_LAST_COPY_ID」、「PG_LAST_COPY_COUNT」、「PG_LAST_QUERY_ID」を参照してください。	2013 年 11 月 1 日
UNLOAD の MANIFEST オプション	UNLOAD コマンドの MANIFEST オプションは、COPY コマンドの MANIFEST オプションを補完します。UNLOAD で MANIFEST オプションを使用すると、アンロードオペレーションによって Amazon S3 で作成されたデータファイルを明示的に一覧表示するマニフェストファイルが自動的に作成されます。その後、同じマニフェストファイルを COPY コマンドで使用してデータをロードできます。詳細については、データを Amazon S3 にアンロードするおよびUNLOAD の例を参照してください。	2013 年 11 月 1 日
COPY の MANIFEST オプション	COPY コマンドに MANIFEST オプションを付けることで、Amazon S3 からロードされるデータファイルを明示的にリスト化できます。	2013 年 10 月 18 日

変更	説明	変更日
トラブルシューティングクエリのシステムテーブル	トラブルシューティングクエリに使用するシステムテーブルに関する追加ドキュメント。ログ記録のための STL ビュー セッションに、以下のシステムテーブルに関するドキュメントが追加されました: STL_AGGR、STL_BCAST、STL_DIST、STL_DELETE、STL_HASH、STL_HASHJOIN、STL_INSERT、STL_LIMIT、STL_MERGE、STL_MERGEJOIN、STL_NESTLOOP、STL_PARSE、STL_PROJECT、STL_SCAN、STL_SORT、STL_UNIQUE、STL_WINDOW。	2013 年 10 月 3 日
CONVERT_TIMEZONE 関数	CONVERT_TIMEZONE 関数 は、ひとつのタイムゾーンから別のタイムゾーンに自動的にタイムスタンプを変換します。夏時間の設定も自動的に調整されます。	2013 年 10 月 3 日
SPLIT_PART 関数	SPLIT_PART 関数 は指定した区切り記号を使って文字列を分割し、指定した位置の文字列を返します。	2013 年 10 月 3 日
STL_USERLOG システムテーブル	STL_USERLOG データベースユーザーが作成、変更、または削除されたときに発生する変更の詳細を記録します。	2013 年 10 月 3 日
LZO 列エンコーディングと LZOP ファイル圧縮。	LZO 列圧縮エンコーディングは、非常に高い圧縮率と高いパフォーマンスを同時に実現します。Amazon S3 からの COPY では、LZOP 圧縮によって圧縮したファイルのロードをサポートしています。	2013 年 9 月 19 日
JSON、正規表現、およびカーソル	JSON 文字列の解析、正規表現を使ったパターンマッチング、および ODBC 接続でカーソルを使用した大規模データの取得がサポートされるようになりました。詳細については、JSON 関数、パターンマッチング条件、およびDECLAREを参照してください。	2013 年 9 月 10 日
COPY の ACCEPTINVCHAR オプション	COPY コマンドに ACCEPTINVCHAR オプションを付けて実行すると、無効な UTF-8 文字を含むデータも問題なくロードすることができます。	2013 年 8 月 29 日

変更	説明	変更日
COPY の CSV オプション	COPY コマンドで、CSV 形式の入力ファイルをロードできるようになりました。	2013 年 8 月 9 日
CRC32	CRC32 関数 は周期的な冗長チェックを実行します。	2013 年 8 月 9 日
WLM ワイルドカード	ワークロード管理 (WLM) でクエリにユーザーグループやクエリグループを追加する際に、ワイルドカードを使用できます。詳細については、「ワイルドカード」を参照してください。	2013 年 8 月 1 日
WLM タイムアウト	各キューについて WLM タイムアウト値を設定することで、特定の WLM キュー内でクエリが使用できる時間を制限することができます。詳細については、「WLM タイムアウト」を参照してください。	2013 年 8 月 1 日
新しい COPY オプション: auto と epochsecs	COPY コマンドは、日付と時刻の形式を自動的に認識します。新しい時刻形式である「epochsecs」と「epochmillisecs」により、COPY コマンドでエポック形式のデータをロードすることができます。	2013 年 7 月 25 日
CONVERT_TIMEZONE 関数	CONVERT_TIMEZONE 関数 は、ひとつのタイムゾーンから別のタイムゾーンにタイムスタンプを変換します。	2013 年 7 月 25 日
FUNC_SHA1 関数	FUNC_SHA1 関数 は、SHA1 アルゴリズムを使用して文字列を変換します。	2013 年 7 月 15 日
max_execution_time	クエリが使用できる時間を制限するために、WLM 設定で max_execution_time パラメータを使用することができます。詳細については、「WLM 設定の変更」を参照してください。	2013 年 7 月 22 日
4 バイト UTF-8 文字	VARCHAR データタイプで 4 バイトの UTF-8 文字を使用できるようになりました。5 バイト以上の UTF-8 文字はサポートされません。詳細については、「ストレージと範囲」を参照してください。	2013 年 7 月 18 日

変更	説明	変更日
SVL_QERROR	SVL_QERROR システムビューは廃止されました。	2013 年 7 月 12 日
ドキュメント履歴の改訂	ドキュメント履歴ページに、ドキュメント更新の日付が表示されるようになりました。	2013 年 7 月 12 日
STL_UNLOAD_LOG	STL_UNLOAD_LOG はアンロード処理の詳細を記録します。	2013 年 7 月 5 日
JDBC フェッチサイズパラメータ	JDBC を使用して大きなデータセットを取得する際にユーザー側でメモリ不足エラーが発生することを避けるため、JDBC フェッチサイズパラメータを指定して、クライアントがデータをバッチ単位でフェッチするように設定できます。詳細については、「JDBC フェッチサイズパラメータの設定」を参照してください。	2013 年 6 月 27 日
UNLOAD 暗号化ファイル	UNLOAD で、Amazon S3 での、テーブルデータから暗号化ファイルへのアンロードがサポートされるようになりました。	2013 年 5 月 22 日
一時認証情報	COPY と UNLOAD で、一時認証情報が使用できるようになりました。	2013 年 4 月 11 日
説明の追加	テーブルの設計とデータのロードについての、より明確で詳細な説明が追加されました。	2013 年 2 月 14 日
ベストプラクティスの追加	Amazon Redshift テーブル設計のベストプラクティス と データをロードするための Amazon Redshift のベストプラクティス が追加されました。	2013 年 2 月 14 日
パスワード規定の明確化	CREATE USER および ALTER USER でのパスワード規定がより明確になりました。また、細かい改訂が加えられました。	2013 年 2 月 14 日
新規ガイド	こちらは 「Amazon Redshift デベロッパーガイド」の初回リリースです。	2013 年 2 月 14 日