Amazon Redshift のクエリのベストプラクティス

AWS 規範ガイダンス

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 規範ガイダンス: Amazon Redshift のクエリのベストプラクティス

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

Amazon Redshift のクエリのベストプラクティス

Amazon Web Services (AWS)、Ethan Stark

2024 年 6 月 (ドキュメント履歴)

概要

このガイドでは、Amazon Redshift でクエリとテーブルのパフォーマンスを最適化するための推奨事項とベストプラクティスについて説明します。Amazon Redshift では、標準 SQL を使用して、データウェアハウスとデータレイク全体でペタバイトの構造化データおよび半構造化データをクエリできます。また、このガイドでは、Amazon Redshift データウェアハウスのコアアーキテクチャコンポーネントの概要についても説明します。テーブルプロパティ、クラスター設定、クエリ構造などのクエリパフォーマンス要因についての理解とともに、この知識は、Amazon Redshift データウェアハウスの効率的かつ効果的なテーブルとクエリの設計に役立ちます。

対象者

このガイドは、Amazon Redshift でテーブルとクエリを設計または使用するデータエンジニア、データアーキテクト、データアナリストを対象としています。

目的

このガイドは、お客様と組織が以下の目標を達成することに役立ちます。

- データストレージと取得オペレーションを最適化するためのテーブルを設計する
- 最適なパフォーマンスとコスト削減のためのクエリを設計する
- Amazon Redshift Spectrum のパフォーマンスを最適化して、Amazon Simple Storage Service (Amazon S3) のファイルから直接データをクエリする

Amazon Redshift データウェアハウスのアーキテクチャコンポーネント

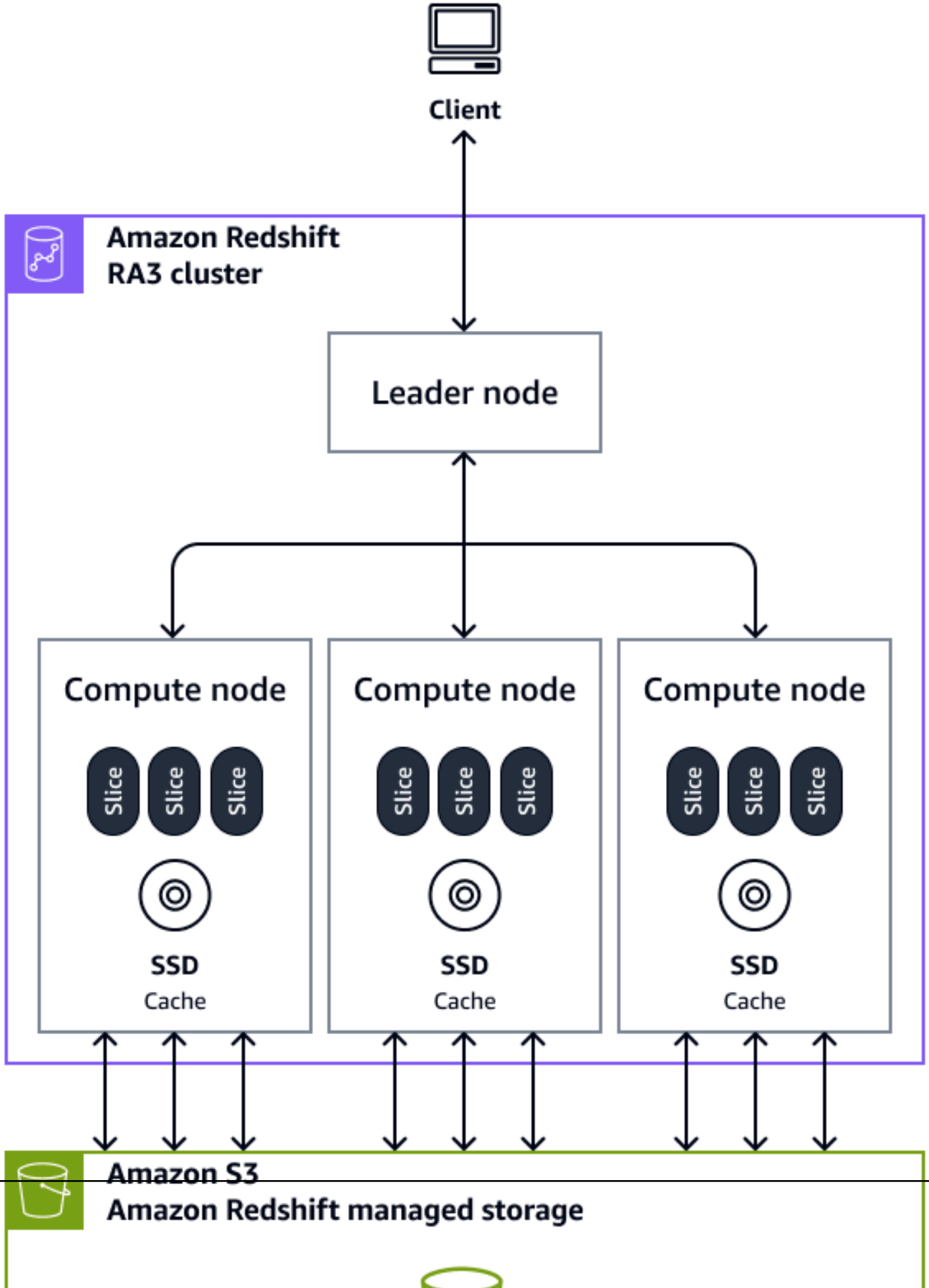
Amazon Redshift データウェアハウスのコアアーキテクチャコンポーネントの基本事項について理解しておくことをお勧めします。この知識は、最適なパフォーマンスを実現するためにクエリとテーブルを設計する方法をよりよく理解するのに役立ちます。

Amazon Redshift のデータウェアハウスは、次のコアアーキテクチャコンポーネントで構成されます。

- クラスター – 1 つ以上のコンピューティングノードで構成されるクラスターは、Amazon Redshift データウェアハウスのコアインフラストラクチャコンポーネントです。コンピューティングノードは外部アプリケーションに対して透過的ですが、クライアントアプリケーションはリーダーノードとのみ直接やり取りします。一般的なクラスターには、2 つ以上のコンピューティングノードがあります。コンピューティングノードは、リーダーノードを介して調整されます。
- リーダーノード – リーダーノードは、クライアントプログラムとすべてのコンピューティングノードの通信を管理します。また、リーダーノードは、クエリがクラスターに送信されるたびにクエリを実行するためのプランも準備します。プランの準備が整うと、リーダーノードでコードをコンパイルし、コンパイル済みのコードをコンピューティングノードに配布してから、データの一部分を各コンピューティングノードに割り当てて、クエリ結果を処理します。
- コンピューティングノード – コンピューティングノードでクエリを実行します。リーダーノードは、クエリ実行プランの個々の要素にコードをコンパイルし、コードを個々のコンピューティングノードに割り当てます。コンピューティングノードは、コンパイル済みのコードを実行し、最終的な集計のために中間結果をリーダーノードに返送します。各コンピューティングノードには、専用の CPU、メモリ、接続されているディスクストレージがあります ワークロードが増えるに従って、ノードの数を増やすかノードの種類をアップグレードして、または両方を行って、クラスターのコンピューティング能力とストレージ容量を拡張できます。
- ノードスライス – コンピューティングノードはスライスと呼ばれる単位に分割されます。コンピューティングノードの各スライスにノードのメモリとディスク容量の一部が割り当てられ、ノードに割り当てられたワークロードの一部分を処理します。スライスは、並列処理を行って操作を完了します。データは、特定のテーブルの分散スタイルと分散キーに基づいて、スライス間で分散されます。データを均等に分散することで、Amazon Redshift がワークロードをスライスに均等に割り当てられるようになり、並列処理の利点を最大化できます。コンピューティングノードあたりのスライス数は、ノードのタイプに基づいて決定されます。詳細については、Amazon Redshift ドキュメントの「Amazon Redshift のクラスターとノード」を参照してください。

- 超並列処理 (MPP) – Amazon Redshift は MPP アーキテクチャを使用して、複雑なクエリや膨大な量のデータであっても、データを迅速に処理します。複数のコンピューティングノードが一部のデータに同じクエリコードを実行して、並列処理を最大化します。
- クライアントアプリケーション – Amazon Redshift は、さまざまなデータロード、抽出、変換、およびロード (ETL)、ビジネスインテリジェンス (BI) レポート、データマイニング、そして分析ツールと統合されています。すべてのクライアントアプリケーションは、リーダーノードを介してのみクラスターと通信します。

次の図は、Amazon Redshift データウェアハウスのアーキテクチャコンポーネントがどのように連携してクエリを高速化するかを示しています。


Client
Amazon Redshift
RA3 cluster
Leader node
Compute node
Slice
Slice
Slice
SSD
Cache
Compute node
Slice
Slice
Slice
SSD
Cache
Compute node
Slice
Slice
Slice
SSD
Cache
Amazon S3
Amazon Redshift managed storage

クエリのライフサイクルには、次の 7 つのステージがあります。

1. クエリの受信と解析:
   - リーダーノードはクエリを受け取り、SQL を解析します。
   - パーサーは、元のクエリの論理構造を表す初期クエリツリーを生成します。
   - Amazon Redshift は、このクエリツリーをクエリオプティマイザに入力します。
2. クエリの最適化:
   - オプティマイザを使用してクエリを評価し、必要なときはクエリを書き換えて効率性を最大限に高めます。
   - このプロセスにより、関連するクエリが複数作成されて、単一のクエリが置き換えられることがあります。
3. クエリプランの生成:
   - オプティマイザによって、実行用のクエリプラン (または必要に応じて複数のプラン) が生成されます。
   - クエリプランは、結合の種類、結合の順序、集計方法、データ分散要件などの実行オプションを指定します。
4. 実行エンジンの変換:
   - 実行エンジンは、クエリプランをステップ、セグメント、ストリームに変換します。
     - ステップ – クエリの実行中に必要な個々のオペレーションを表します。コンピューティングノードはステップを組み合わせることによってクエリ、結合、または他のデータベース操作を実行できます。
     - セグメント – 1 つのプロセスで実行できる複数のステップを組み合わせます。セグメントは、コンピューティングノードのスライスによって実行可能な最小コンパイルユニットです (スライスは、Amazon Redshift の並列処理単位です)。
     - ストリーム – 使用可能なコンピューティングノードのスライスに分配されたセグメントのコレクション。
   - 実行エンジンは、ステップ、セグメント、ストリームに基づいてコンパイルされたコードを生成します。コンパイルされたコードの実行は解釈されたコードよりも速く、使用するコンピューティングキャパシティも少なくなります。
   - リーダーノードは、コンパイルされたコードをコンピューティングノードにブロードキャストします。
5. 同時実行:
   - このステップは、ストリームごとに 1 回実行されます。

- コンピューティングノードのスライスは、クエリセグメントを並列的に実行します。
- このプロセスの間は、Amazon Redshift でネットワーク通信、メモリ使用量、ディスク管理を最適化し、クエリプランのステップから次のステップに中間結果を渡します。
- この最適化は、クエリ実行の高速化に役立ちます。

6. ストリーミング処理:
   - このステップは、ストリームごとに 1 回実行されます。
   - エンジンでストリームごとに実行可能セグメントを作成し、効率的な並列処理を実現します。
7. 最終的なソートと集計:
   - リーダーノードは、クエリに必要な最終的なソートまたは集計に対処します。
   - 完了したら、リーダーノードからクライアントに結果が返されます。

アーキテクチャコンポーネントについては、Amazon Redshift ドキュメントの「データウェアハウスシステムのアーキテクチャ」を参照してください。

Amazon Redshift でのクエリのパフォーマンス要因

いくつかの要因がクエリパフォーマンスに影響を与える可能性があります。データ、クラスター、データベース操作の次の側面はすべて、クエリ処理の速度に影響を与えます。

- テーブルプロパティ
  - ソートキー (Amazon Redshift Advisor)
  - データ圧縮 (自動)
  - データのディストリビューション (自動)
  - テーブルのメンテナンス (自動)
- クラスターの設定
  - ノードタイプ
  - ノードサイズ、ノード数、スライス数
  - ワークロード管理 (自動)
  - ショートクエリアクセラレーション (自動)
- SQL クエリ
  - クエリ構造
  - コードコンパイル

テーブルプロパティ

Amazon Redshift テーブルは、Amazon Redshift にデータを保存するための基本的な単位であり、各テーブルには、その動作とアクセシビリティを決定する一連のプロパティがあります。これらのプロパティには、ソート、分散スタイル、圧縮エンコーディングなどが含まれます。これらのプロパティを理解することは、Amazon Redshift テーブルのパフォーマンス、セキュリティ、コスト効率を最適化するために不可欠です。

ソートキー

Amazon Redshift は、テーブルのソートキーに応じたソート順で、データをディスクに保存します。クエリオプティマイザとクエリプロセッサは、コンピューティングノード内のデータの保存場所に関する情報を使用して、スキャンする必要があるブロックの数を減らします。これにより、処理するデータの量が減り、クエリ速度が大幅に向上します。ソートキーを使用して、WHERE 句のフィル

ターを容易にすることをお勧めします。詳細については、Amazon Redshift のドキュメントの「ソートキーの使用」を参照してください。

データ圧縮

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

データのディストリビューション

Amazon Redshift は、テーブルの分散スタイルに応じて、データをコンピューティングノードに保存します。クエリを実行すると、クエリオプティマイザが結合と集計を実行するための必要に応じて、データをコンピューティングノードに再分散します。テーブルに適した分散スタイルを選択すると、結合を実行する前にデータを必要な場所に配置しておくことによって、再分散ステップの影響を最小限に抑えることができます。最も一般的な結合を容易にするために、分散キーを使用することをお勧めします。詳細については、Amazon Redshift のドキュメントの「データディストリビューションスタイルの操作」を参照してください。

テーブルのメンテナンス

Amazon Redshift はほとんどのワークロードで業界標準のパフォーマンスを提供しますが、Amazon Redshift クラスターを正常に実行し続けるにはメンテナンスが必要です。データを更新および削除すると、バキューム処理が必要なデッド行が作成され、追加順序がソートキーと一致しない場合は、追加専用テーブルも再ソートする必要があります。

Vacuum

Amazon Redshift でのバキューム処理プロセスは、Amazon Redshift クラスターのヘルスとメンテナンスに不可欠です。また、クエリのパフォーマンスにも影響します。削除や更新によって古いデータにフラグが付きますが、実際に削除されたわけではないため、バキューム処理を使用して、前の UPDATE および DELETE 操作で削除対象としてマークされたテーブル行によって占有されたディス

ク領域を再利用する必要があります。Amazon Redshift は、バックグラウンドでテーブルを自動的にソートし、VACUUM DELETE オペレーションを実行できます。

ロードまたは一連の増分更新の後にテーブルをクリーンアップするには、データベース全体または個々のテーブルに対して VACUUM コマンドを実行することもできます。テーブルにソートキーがあり、テーブルのロードが挿入時にソートするように最適化されていない場合は、バキュームを使用してデータを再ソートする必要があります (これはパフォーマンスにとって重要です)。詳細については、Amazon Redshift ドキュメントの「Vacuuming tables」を参照してください。

分析

ANALYZE 操作によって、Amazon Redshift データベース内のテーブルの統計メタデータを更新します。統計を最新状態に保つことで、クエリプランナーが最適なプランを選択できるようになるため、クエリのパフォーマンスが向上します。Amazon Redshift はデータベースを継続的にモニタリングし、バックグラウンドで自動的に分析オペレーションを実行します。システムパフォーマンスへの影響を最小限にするために、ANALYZE 操作はワークロードが軽い期間に自動で実行されます。ANALYZE を明示的に実行することを選択した場合は、以下を実行する必要があります。

- クエリを実行する前に ANALYZE コマンドを実行します。
- 定期的なロードまたは更新サイクルが終わるたびに、データベースで ANALYZE コマンドを定期的に実行します。
- 作成した新しいテーブルと大幅に変更された既存のテーブルまたは列で ANALYZE コマンドを実行します。
- クエリでの使用と変更傾向に基づき、異なるタイプのテーブルおよび列に対し、異なるスケジュールで ANALYZE 操作を実行することを考慮します。
- 時間とクラスターリソースを節約するには、ANALYZE コマンドを実行するときに PREDICATE COLUMNS 句を使用します。

クラスターの設定

クラスターは、データの実際の保存と処理を実行するノードのコレクションです。以下を実現するには、Amazon Redshift クラスターを正しい方法で設定してください。

- 高いスケーラビリティと同時実行性
- Amazon Redshift の効率的な使用
- パフォーマンスの向上

- コストの削減

ノードタイプ

Amazon Redshift クラスターは、複数のノードタイプ (RA3、DC2、DS2) のいずれかを使用できます。各ノードタイプは様々なサイズを提供し、クラスターを適切に拡張することができるように制限します。ノードサイズによって、クラスター内の各ノードのストレージ容量、メモリ、CPU、および料金が決まります。コストとパフォーマンスの最適化は、適切なノードタイプとサイズを選択することから始まります。ノードタイプの詳細については、Amazon Redshift ドキュメントの「Amazon Redshift クラスターの概要」を参照してください。

ノードサイズ、ノード数、スライス数

コンピューティングノードはスライスに分割されています。ノードが多いということは、プロセッサとスライスが多いことを意味するため、クエリの各部分をスライス間で同時実行することによりプロセスをすばやく処理することが可能になります。ただし、ノード数が多いほど、コストも高くなります。つまり、システムに適したコストとパフォーマンスのバランスを見つける必要があります。Amazon Redshift クラスターアーキテクチャの詳細については、Amazon Redshift ドキュメントの「データウェアハウスシステムのアーキテクチャ」を参照してください。

ワークロード管理

Amazon Redshift のワークロード管理 (WLM) により、ユーザーはワークロードのキューに優先順位を付けて柔軟に管理することが可能になります。これにより、実行速度が高く処理時間の短いクエリが、処理時間の長いクエリの後に滞らないようにできます。自動 WLM は、機械学習 (ML) アルゴリズムを使用してクエリをプロファイリングし、クエリの同時実行とメモリ割り当てを管理しながら、適切なリソースを使用して適切なキューに配置します。WLM の詳細については、Amazon Redshift ドキュメントの「ワークロード管理の実装」を参照してください。

ショートクエリアクセラレーション

ショートクエリアクセラレーション (SQA) は、実行時間が短いクエリを、実行時間が長いクエリよりも優先します。SQA では実行時間が短いクエリを専用領域で実行します。このため SQA クエリは、実行時間が長いクエリをキューで待機するよう強制されません。SQA は、実行時間が短く、ユーザー定義のキュー内にあるクエリのみを優先します。SQA を使用すると、実行時間が短いクエリの実行開始が早くなり、ユーザーへの結果表示も早くなります。SQA を有効にすると、実行時間の短いクエリ専用の WLM キューを減らす、またはなくすことができます。さらに、長時間実行されるクエリは、WLM キュー内のスロットに対して競合する必要はありません。つまり、より

少ないクエリスロットを使用するように WLM キューを設定できます。同時実行数が減るとクエリのスループットが向上し、大部分のワークロードに関するシステム全体のパフォーマンスも向上します。SQA の詳細については、Amazon Redshift ドキュメントの「「ショートクエリアクセラレーションを使用する」を参照してください。

SQL クエリ

データベースクエリは、データベースからのデータのリクエストです。リクエストは SQL を使用して Amazon Redshift クラスターに送信される必要があります。Amazon Redshift は、Java Database Connectivity (JDBC) および Open Database Connectivity (ODBC) を介して接続する SQL クライアントツールをサポートします。JDBC または ODBC ドライバーをサポートするほとんどの SQL クライアントツールを使用できます。

クエリ構造

クエリの書き込み方法は、そのパフォーマンスに大きな影響を与えます。クエリを作成して処理し、ニーズに合わせて必要な最小限のデータを返すことをお勧めします。クエリの構造方法の詳細については、このガイドの「Amazon Redshift クエリの設計のベストプラクティス」セクションを参照してください。

コードコンパイル

Amazon Redshift は、クエリ実行プランごとに最適化されたコードを生成してコンパイルします。コンパイル済みコードは、インタプリタを使用してオーバーヘッドを排除するため、高速で実行されます。コンパイルされたコードのパフォーマンス上の利点を維持しながら、新しいクエリのレイテンシーを最小限に抑えるために、Amazon Redshift はコンポジションと呼ばれる手法を使用します。コンポジションは、既存のロジックの軽量な配置を生成して新しいクエリをすぐに処理し、同時に高度に最適化されたクエリ固有のコードをバックグラウンドでコンパイルします。これにより、クエリ実行のクリティカルパスからコンパイルが削除されます。つまり、新しいクエリはより高速に開始され、後続の実行と一貫したパフォーマンスを提供します。

また、Amazon Redshift はサーバーレスコンパイルサービスを使用して、Amazon Redshift クラスターのコンピューティングリソースを超えてクエリコンパイルをスケールします。コンパイルされたコードセグメントは、クラスター上のローカルキャッシュと、クラスターの再起動後も保持される実質的に無制限のリモートキャッシュの両方にキャッシュされます。同じクエリをそれ以降に実行すると、コンパイルフェーズをスキップできるため、高速になります。スケーラブルなコンパイルサービスを使用することで、Amazon Redshift はコードを並行してコンパイルし、一貫して高速なパフォーマンスを提供します。

Amazon Redshift テーブル設計のベストプラクティス

このセクションでは、データベーステーブルの設計に関するベストプラクティスの概要を説明します。クエリのパフォーマンスと効率を最適化するには、以下のベストプラクティスに従うことをお勧めします。

ソートキーの仕組みを理解する

Amazon Redshift は、ソートキーに応じたソート順でデータをディスクに保存します。Amazon Redshift クエリオプティマイザは、最適なクエリプランを決定する際にソート順を使用します。ソートキーを効果的に使用するには、以下を実行することをお勧めします。

- テーブルは可能な限りソートされたままにします。
- VACUUM ソートを使用して最適なパフォーマンスを復元します。
- ソートキー列を圧縮しないでください。
- ソートキーが圧縮され、sortkey1_skew 比率が著しく高い場合は、ソートキーで圧縮を有効にせずにテーブルを再作成します。
- ソートキー列に関数を適用しないでください。例えば、次のクエリでは、trans_dt : TIMESTAMPTZ ソートキー列を DATE にキャストする場合、そのソートキー列は使用されません。

```
select order_id, order_amt
from sales
where trans_dt::date = '2021-01-08'::date
```

- ソートキーの順序で INSERT オペレーションを実行します。
- 可能な場合は、GROUP BY 句でソートキーを使用します。

クエリ調整のヒント

クエリを調整するには、以下を実行することをお勧めします。

- 最適な効果を得るには、常に複合ソートキーをカーディナリティの低いものから高いものの順に並べます。
- 複合ソートキーの先頭キーが比較的一意である (つまり、カーディナリティが高い) 場合は、ソートキーに列を追加しないでください。列を追加してもクエリのパフォーマンスにはほとんど影響せず、メンテナンスコストが増えていきます。

ソートキーの有効性を評価する

クエリを最適化するには、クエリの有効性を評価できるようにする必要があります。SVL_QUERY_SUMMARY ビューを使用して、クエリの実行についての全般的な情報を確認することをお勧めします。このビューでは、IS_RRSCAN 属性を使用して、EXPLAIN プランのステップで範囲制限付きスキャンを使用しているかどうかを判断できます。また、rows_pre_filter 属性を使用して、ソートキーの選択性を決定することもできます。

v_my_last_query_summary という GitHub の管理者ビューを使用することもできます。ビューには、最後に実行されたクエリの情報が表示されます。

次のステートメントは、クエリの実行に関する一般的な情報を検索する方法を示しています。

```
select lpad(' ',stm+seg+step) || label as label,
       rows,
       bytes,
       is_diskbased,
       is_rrscan,
       rows_pre_filter
from svl_query_summary
where query = pg_last_query_id()
order by stm, seg, step;
```

前述のクエリは、次のサンプル出力を返します。

label	rows	bytes	is_diskbased	is_rrscan	rows_pre_filter
scan tbl=163860 name=orders	1500000	24000000	f	f	1500000
project	1500000	0	f	f	0
project	1500000	0	f	f	0
hash tbl=968	1500000	24000000	f	f	0
scan tbl=163852 name=lineitem	6001215	144029160	f	t	6001215
project	6001215	0	f	f	0
project	6001215	0	f	f	0
hjoin tbl=968	6001215	0	f	f	0
project	6001215	0	f	f	0
project	6001215	0	f	f	0

テーブルについて理解する

テーブルの重要なプロパティを理解することが重要です。テーブルの詳細については、以下を実行してください。

- PG_TABLE_DEF を使用して、テーブル列に関する情報を表示します。
- SVV_TABLE_INFO を使用すると、データ分散スキュー、キー分散スキュー、テーブルサイズ、統計情報など、テーブルに関するより包括的な情報を表示することができます。

適切なテーブルの分散スタイルを選択する

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

適切なテーブル分散スタイルを選択するには、次のアプローチをお勧めします。

- 同じノード内の行をコロケーションすることで、クエリ実行プランでのブロードキャストと再分散を回避します。例えば、DISTKEY を選択すると、ファクトテーブルと 1 次元テーブルを共通の列に分散できます。フィルタリングされたデータセットのサイズに基づいて最大ディメンションを選択します。結合で使用されている行のみが分散される必要があるため、テーブルのサイズではなく、フィルタリング後のデータセットのサイズを考慮します。
- 分散キーが作成された列に歪みがないことを確認します。歪みがあると、1 つのコンピューティングノードが他のコンピューティングノードよりも重い負荷をかける可能性があります。歪みがあることに気付いた場合は、分散キー列の変更を検討してください。列の値が均一に分散されているか、カーディナリティ値が高い場合、列を分散キーの候補と見なすことができます。
- 結合条件で使用されるテーブルが小さい (1 GB 未満) 場合は、分散スタイル ALL を検討してください。
- 分散キーを圧縮することはできますが、ソートキー列 (特にソートキーの最初の列) を圧縮しないようにする必要があります。

> **Note**
>
> 自動テーブル最適化を使用する場合、テーブルの分散スタイルを選択する必要はありません。詳細については、Amazon Redshift ドキュメントの「Working with automatic table optimization」を参照してください。Amazon Redshift が適切なディストリビューションスタイルを選択できるようにするには、ディストリビューションスタイルに AUTO を指定します。

Amazon Redshift クエリの設計のベストプラクティス

このセクションでは、クエリの設計に関するベストプラクティスの概要を説明します。最適なクエリのパフォーマンスと効率性を実現するには、このセクションのベストプラクティスに従うことをお勧めします。

SELECT * FROM ステートメントを使用しない

SELECT * FROM ステートメントは使用しないことをお勧めします。代わりに、分析する列を必ず一覧表示してください。これにより、クエリの実行時間が短縮され、Amazon Redshift Spectrum クエリのスキャンコストが削減されます。

避けるべき内容の例

```
select *
from sales;
```

ベストプラクティスの例

```
select sales_date, sales_amt
from sales;
```

クエリの問題を特定する

STL_ALERT_EVENT_LOG ビューを確認し、クエリで発生する可能性のある問題を特定して修正することをお勧めします。

クエリの概要情報を取得する

クエリの概要情報を取得するには、SVL_QUERY_SUMMARY ビューと SVL_QUERY_REPORT ビューを使用することをお勧めします。この情報を使用して、クエリを最適化できます。

クロス結合を避ける

やむを得ない場合を除き、クロス結合を使用することは避けてください。クロス結合は、2 つのテーブルの直積集合を算出する結合条件のない結合です。クロス結合は通常ネステッドループ結合として実行されますが、これは使用可能な結合タイプで最も低速な結合です。

避けるべき内容の例

```
select c.c_name,
          n.n_name
from tpch.customer c,
        tpch.nation n;
```

ベストプラクティスの例

```
select c.c_name,
           n.n_name
from tpch.customer c,
join tpch.nation n
  on n.n_nationkey = c.c_nationkey;
```

クエリ述語で関数を使用しない

クエリ述語では関数を使用しないことをお勧めします。通常、関数は各行に処理オーバーヘッドを追加し、クエリの実行全体を遅くするため、クエリ述語で関数を使用すると、パフォーマンスに悪影響を及ぼす可能性があります。

避けるべき内容の例

```
select sum(o_totalprice)
from tpch.orders
where datepart(year, o_orderdate) = 1992;
```

ベストプラクティスの例

```
select sum(o_totalprice)
from tpch.orders
where o_orderdate between '1992-01-01' and '1992-12-31';
```

不要なキャスト変換を避ける

データ型のキャストには時間とリソースがかかり、クエリの実行が遅くなるため、クエリで不要なキャスト変換を使用しないことをお勧めします。

避けるべき内容の例

```
select sum(o_totalprice)
from tpch.orders
where o_ordertime::date = '1992-01-01';
```

ベストプラクティスの例

```
select sum(o_totalprice)
from tpch.orders
where o_ordertime between '1992-01-01 00:00:00' and '1992-12-31 23:59:59';
```

複雑な集計には CASE 式を使用する

同じテーブルから複数回選択するのではなく、CASE 式を使用して複雑な集計を実行することをお勧めします。

避けるべき内容の例

```
select sum(sales_amt) as us_sales
from sales
where country = 'US';

select sum(sales_amt) as ca_sales
from sales
where country = 'CA';
```

ベストプラクティスの例

```
select sum(case when country = 'US' then sales_amt end) as us_sales,
           sum(case when country = 'CA' then sales_amt end) as ca_sales
from sales;
```

サブクエリを使用する

クエリのテーブル 1 つが述語条件のみで使用されている場合には、サブクエリを使用することをお勧めします。この場合、サブクエリはわずかな数の行を返します (ほぼ 200 行未満)。

避けるべき内容の例

サブクエリが 200 行未満を返す場合:

```
select sum(order_amt) as total_sales
from sales
where region_key IN
        (select region_key
         from regions
         where state = 'CA');
```

ベストプラクティスの例

サブクエリが 200 行以上を返す場合:

```
select sum(o.order_amt) as total_sales
from sales o
join regions r
  on r.region_key = o.region_key
  and r.state = 'CA';
```

述語を使用する

述語を使用してデータセットをできるだけ制限することをお勧めします。述語は SQL で使用され、クエリで返されるデータをフィルタリングおよび制限します。述語で条件を指定することで、指定された条件に基づいてクエリ結果に含める必要がある行を指定できます。これにより、関心のあるデータのみを取得でき、クエリの効率と精度が向上します。詳細については、Amazon Redshift ドキュメントの「Conditions」を参照してください。

述語を追加して結合を使用するテーブルをフィルタリングする

述語が同じフィルタを適用する場合でも、述語を追加して結合に関与するテーブルをフィルタリングすることをお勧めします。述語を使用して SQL で結合を使用するテーブルをフィルタリングすると、処理する必要があるデータの量と中間結果セットのサイズを減らして、クエリのパフォーマンスを向上させることができます。WHERE 句で結合オペレーションの条件を指定することで、クエリ実行エンジンは、結合前に条件に一致しない行を削除できます。これにより、結果セットが小さくなり、クエリの実行が高速化されます。

避けるべき内容の例

```
select p.product_name, sum(o.order_amt)
from sales o
```

```
join product p
   on r.product_key = o.product_key
where o.order_date > '2022-01-01';
```

ベストプラクティスの例

```
select p.product_name, sum(o.order_amt)
from sales o
join product p
  on p.product_key = o.product_key
  and p.added_date > '2022-01-01'
where o.order_date > '2022-01-01';
```

述語には最も安価な演算子を使用する

述語では、できる限りコストのかからない演算子を使用します。比較条件演算子は LIKE 演算子よりも優先されます。LIKE 演算子は、SIMILAR TO 演算子または POSIX 演算子よりも優先されます。

GROUP BY 句でソートキーを使用する

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

マテリアライズドビューの利点

可能であれば、複雑なコードをマテリアライズドビューに置き換えてクエリを書き換えます。これにより、クエリのパフォーマンスが大幅に向上します。詳細については、Amazon Redshift のドキュメントの「Amazon Redshift でのマテリアライズドビューの作成」を参照してください。

GROUP BY 句と ORDER BY 句の列に注意する

ORDER BY 句と GROUP BY 句の両方を使用する場合は、列を GROUP BY 句と ORDER BY 句の両方に同じ順序で配置してください。GROUP BY では、データを暗黙的にソートする必要があります。ORDER BY 句が異なる場合は、データを 2 回ソートする必要があります。

避けるべき内容の例

```
select a, b, c, sum(d)
from a_table
group by b, c, a
order by a, b, c
```

ベストプラクティスの例

```
select a, b, c, sum(d)
from a_table
group by a, b, c
order by a, b, c
```

Amazon Redshift Spectrum を使用する際のベストプラクティス

このセクションでは、Amazon Redshift Spectrum を使用するためのベストプラクティスの概要について説明します。Redshift Spectrum を使用する際に最適なパフォーマンスを実現するには、以下のベストプラクティスに従うことをお勧めします。

- ファイルタイプが Redshift Spectrum クエリのパフォーマンスに重大な影響を与えることを考慮してください。パフォーマンスを向上させるには、ORC や Parquet などの列エンコードされたファイルを使用し、CSV 形式はごく小さなディメンションテーブルにのみ使用します。
- プレフィックスベースのパーティショニングを使用して、パーティションプルーニングを活用します。つまり、データレイク内のパーティションにキー指定されたフィルターを使用します。
- Redshift Spectrum は大規模なリクエストを処理するために自動的にスケールされるため、Redshift Spectrum ではできるだけ多くの処理を行います (述語のプッシュダウンなど)。
- 頻繁にフィルタリングされる列のパーティションファイルには注意してください。データが 1 つ以上のフィルタリングされた列によってパーティション分割されている場合、Redshift Spectrum はパーティションプルーニングを活用し、不要なパーティションやファイルのスキャンをスキップできます。一般的な方法では、時間に基づいてデータをパーティション化します。
- 次のクエリを使用して、パーティションの有効性と Redshift Spectrum クエリの効率性を確認できます。

```
Select query,
           segment,
           max(assigned_partitions) as total_partitions,
           max(qualified_partitions) as qualified_partitions
From svl_s3partition
Where query=pg_last_query_id()
Group by 1,2;
```

前述のクエリは以下を示しています。

- total_partitions – によって認識されるパーティションの数 AWS Glue Data Catalog
- qualified_partitions – Redshift Spectrum クエリでアクセスされる Amazon Simple Storage Service (Amazon S3) のプレフィックスの数

- また、SVL_S3QUERY_SUMMARY システムテーブルをチェックして、パーティションの有効性と Redshift Spectrum クエリの効率性を確認することもできます。これを行うには、以下のステートメントを実行します。

```
Select *
From svl_s3query_summary
Where query=pg_last_query_id();
```

  前述のクエリは、パーティションプルーニングの効率性を示すファイルに加えて、is_partitioned、s3_scanned_rows/bytes、および s3_returned_rows/bytes の値を含むさらに多くの情報を返します。

Redshift Spectrum での述語プッシュダウン

述語プッシュダウンを使用すると、Amazon Redshift クラスターでのリソースの消費を回避できます。多くの SQL オペレーションを Redshift Spectrum レイヤーにプッシュダウンできます。これを可能な限り活用することをお勧めします。

以下に留意してください。

- Redshift Spectrum レイヤー内では、次のようないくつかのタイプの SQL オペレーションを完全に評価できます。
  - GROUP BY 句
  - 比較条件とパターンマッチング条件 (例: LIKE)
  - 集計関数 (例: COUNT、SUM、AVG、MIN、MAX)
  - regex_replace、to_upper、date_trunc、およびその他の関数
- DISTINCT や ORDER BY を含む一部のオペレーションを Redshift Spectrum レイヤーにプッシュすることはできません。ソートはリーダーノードで実行されるため、可能であればクエリの最上位レベルでのみ ORDER BY を実行してください。
- EXPLAIN クエリプランを調べて、述語プッシュダウンが有効かどうかを確認します。EXPLAIN コマンドで Redshift Spectrum 部分を検索するには、以下のステップを参照してください。
  - S3 Seq Scan
  - S3 HashAggregate
  - S3 Query Scan
  - Seq Scan PartitionInfo

  - Partition Loop
- クエリで使用する列数を最小限にします。Redshift Spectrum では、データが Parquet 形式または ORC 形式の場合、スキャンする列を減らすことができます。
- 並列処理とパーティション排除のためにパーティションをフル活用し、可能であればファイルサイズを 64 MB 以上に維持します。
- `CREATE EXTERNAL TABLE` または `ALTER TABLE` を使用する場合は、`TABLE PROPERTIES 'numRows'='nnn'` を設定します。Amazon Redshift は、外部テーブルを分析して、クエリオプティマイザがクエリプランを生成するために使用するテーブル統計を生成することはありません。統計情報が設定されていない場合、Amazon Redshift は、外部テーブルの方が大きくローカルテーブルの方が小さいということを前提にします。

Redshift Spectrum のクエリ調整のヒント

クエリを調整するときは、次の点に注意することをお勧めします。

- Amazon Redshift クラスターがクエリにエンゲージできる Redshift Spectrum ノードの数は、クラスター内のスライスの数に関連付けられます。
- クラスターのサイズを変更すると、クラスターのローカルコンピューティングプロファイル、ストレージプロファイル、Amazon S3 データレイククエリのクエリ機能にメリットがあります。
- Amazon Redshift クエリプランナーは、述語と集計を可能な限り Redshift Spectrum クエリレイヤーにプッシュします。
- Amazon S3 から大量のデータが返されると、クラスターのリソースによって処理が制限されます。
- Redshift Spectrum は大規模なリクエストを処理するように自動的にスケールするため、Redshift Spectrum レイヤーに処理をプッシュできるたびに全体的なパフォーマンスが向上します。

リソース

- Amazon Redshift のベストプラクティス (Amazon Redshift ドキュメント)
- Amazon Redshift クエリの設計のベストプラクティス (Amazon Redshift ドキュメント)
- クエリパフォーマンスのチューニング (Amazon Redshift ドキュメント)
- クエリプラン (Amazon Redshift ドキュメント)
- Amazon Redshift Spectrum クエリパフォーマンスの向上 (Amazon Redshift ドキュメント)
- Amazon Redshift のクエリライフサイクルについて (AWS 規範ガイダンス)

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
AQUA を削除	Advanced Query Accelerator (AQUA) に関する情報を削除しました。	2024 年 6 月 14 日
初版発行	—	2023 年 2 月 3 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。