# クエリ最適化のためのプロジェクション再設計

**March, 2017**

原文はこちら

MICRO FOCUS

# 目次

## 概要

この文書は、次の記事を含むパフォーマンスチューニングに関する 3 つのシリーズの第 3 回目の文書です。

パート 1: Reading Query Plans

パート 2: Troubleshooting Vertica Query Performance with System Tables

パート 3: クエリ最適化のためのプロジェクション再設計

Vertica はクエリを実行した際、自動的にクエリオプティマイザーにて最適な実行計画を作成します。実行計画は、一連の一式の操作を考慮し計算します。データベースで定義したプロジェクションのプロパティに応じで、クエリオプティマイザーは、効率的、且つ、高速な操作を選択します。このように、プロジェクションを最適化することは、クエリパフォーマンスを最適化するのに重要です。

異なるプロジェクション設計は、パフォーマンスやクエリ実行に必要なリソースに異なる影響をします。

以下の表では、クエリパフォーマンスに影響のある項目をプラス(+)、マイナス(-)で表現しました。例えば、エンコーディングすることで、ネットワーク使用率を削減し、プラス(＋)の影響を得ることができます。

| | 影響 | | | | | |
|---|---|---|---|---|---|---|
| **プロジェクションのプロパティ** | | メモリ | CPU | I/O | ネットワーク | パフォーマンス |
| | エンコーディング | + データエンコード時<br>- データマテリアライゼーション後 | +/- エンコーディングタイプに依存 | + | + | + |
| | Order by | +join もしくは、group by | - ロード、マージアウト | 影響無し | 影響無し | + |
| | セグメンテーション | 影響無し | - ロード、マージアウト | + | + ロード時<br>- join もしくは、group by | + テーブルサイズが大きく、分散している場合<br>- join |

## エンコーディングとデータ圧縮

カラム型指向データベースとして、Vertica はデータタイプに応じたエンコーディングを適用します。エンコーディングのゴールは、ストレージサイズを減らすことです。そうすることで、ロードやリードに必要な I/O を削減でき、パフォーマンスを改善できます。ディスクサイズの容量を減らすメリットの一方、エンコーディングで CPU 使用率が上がるかもしれません。
エンコーディングに追加して、Vertica はデータ圧縮を適用します。LZO アルゴリズムを利用してデータ圧縮を実施します。
Vertica は、マテアライゼーションするまで、エンコードしたデータで処理します。しかし、圧縮データは、クエリ実行時に解凍し、非圧縮データにして処理します。

### *デフォルト(オート)エンコーディング*

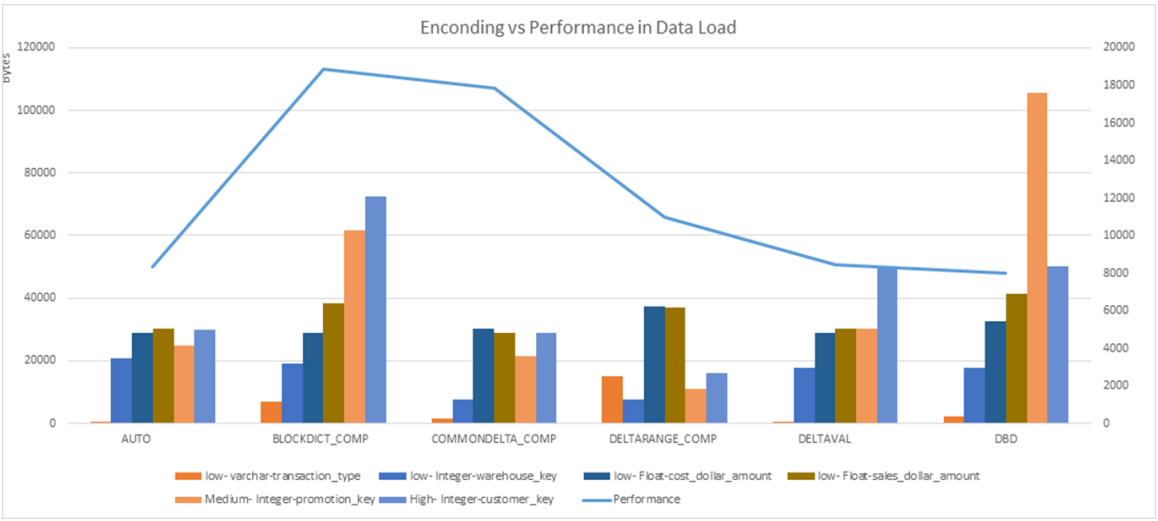
エンコーディングのタイプを指定しない場合、Vertica は自動でエンコーディングを決定します。デフォルトのエンコーディングのタイプは、データの特定に依存します。
例えば、TIMESTAMP 例のデータは、連続したデータのため差分をベースとしたデータ圧縮になります。
CHAR や VARCHAR 型のデータを格納している列は、Vertica は、LZO 圧縮を適用します。以下のリンクよりデフォルトのエンコーディングとデータタイプを確認することができます。詳細情報はこちら。

### *データベースデザイナーでエンコーディング*

データベースデザイナーを実行することで、使用するプロジェクションに最適なエンコーディングを確認することができます。データベースデザイナーは、1%のサンプルデータを基に、列のタイプ、カーディナリティを特定します。これらのパラメーターをベースに、データベースデザイナーは、異なるエンコードオプションをテストし、最適な1つを選択します。

また、「DESIGNER_DESIGN_PROJECTION_ENCODINGS」機能を特定のプロジェクションに対して実行することで、最適なエンコーディングを計算し、スクリプトを作成します。また、その推奨のエンコーディングを適用することもできます。

**例:データロード時のエンコーディングとパフォーマンスの関係**



## *役立つシステムテーブル*

PROJECTION_COLUMNS：エンコーディングのタイプ、ソートオーダー、統計タイプ、最後に更新した統計のタイムスタンプを表示します。

COLUMN_STORAGE: ノードごとの各列のディスク容量を表示します。

**サンプルクエリ:**

```
SELECT * FROM PROJECTION_COLUMNS pc, COLUMN_STORAGE cs
                WHERE pc.column_id = cs.column_id
                AND pc.projection_id = cs.projection_id
                AND pc.projection_name ilike '<projection_name>'
                GROUP BY 1,2,3,4,5,6,7,9,10,11
                ORDER BY pc.projection_name, pc.sort_position ;
-[ RECORD 1 ]----------------+-------------------------------------------
-----
projection_name              |
online_sales_fact_DBD_15_seg_vertica_7_2_3_8_b0
projection_column_name       | product_key
column_position              | 2
sort_position                |
encoding_type                | DELTAVAL
encodings                    | Int_Delta
compressions                 | none
sum                          | 40112451252
access_rank                  | 0
statistics_type              | FULL
statistics_updated_timestamp | 2016-10-17 18:44:27.255243-04
…
```

## Order By 句のプロジェクション動作

## *マージ結合 vs. ハッシュ結合*

Join 処理のパフォーマンスを改善するためにいくつかの方法があります。一つは、Vertica に最適なインナー、アウターテーブルを選択することで、追加のソートやデータ転送の操作をすることなく、高速な処理を実行することができます。

結合を実行するために Vertica は、以下の 2 つのアルゴリズムを使用します。

- マージ結合（Merge join）
- ハッシュ結合（Hash join）

### *長所と短所*

| | 長所 | 短所 |
|---|---|---|
| マージ結合<br>（Merge join） | • 高速<br>• 小さいメモリサイズ | • ソート済みのカラム（列）を Join 句に入れる |
| ハッシュ結合<br>（Hash join） | • ソートは不要<br>• インナーテーブルのサイズが小さい場合、高速処理ができる。また、小さいメモリサイズで実行できる。 | • 大容量のメモリサイズ<br>• インナーテーブルがメモリ領域に収まらない場合、クエリ実行は失敗する。失敗したクエリは、ディスク領域を使用して再実行する。処理速度は遅くなる。 |

オプティマイザーは、クエリやプロジェクションをベースに自動的に適切なアルゴリズムを選択します。ハッシュ結合のアルゴリズムでは、Vertica は、小さなサイズのテーブル（インナー）をベースに、結合用テーブルをメモリ上に作成します。Vertica は、大きなサイズのテーブル（アウター）をスキャンし、ハッシュテーブルをプローブします。プロジェクションを事前に結合キーでソートしている場合、オプティマイザーは、マージ結合を選択し、処理を高速化することができます。

もし、アウターテーブルをソートしている場合、Vertica はソートされたデータを利用して、インナーテーブルとマージ結合するかもしれません。いくつかのケースでは、メモリ上でインナーテーブルをすべてハッシュするより、少ないメモリで実行することができます。

### *ヒント*

オプティマイザーに以下のヒントを指定することが可能です。

/*+JType(M)*/ :マージ結合を強制実行する。もし、結合キーがソートされていない場合、事前にソート処理し、結合を実行します。

/*+JType(H)*/ :ハッシュ結合を強制実行する。

## Group By 句のプロジェクション動作

### *PIPELINED vs. HASH*

クエリに Group By 句が含まれる場合、Vertica は、GROUP BY PIPELINED もしくは GRUPBY　HASH アルゴリズムのどちらかで結果を計算します。両方のアルゴリズムが同じ結果となり、クエリに多くの Group が含まれる場合、GROUPBY PIPELINED は少ないメモリサイズで実行でき、高速に処理できます。

### *長所と短所*

| | 長所 | 短所 |
|---|---|---|
| PIPELINED | • 少ないメモリで実行でき処理速度が速い（グループ数が多い場合） | • インプットは、Group By の列をソートする必要あり |

| | | |
|---|---|---|
| HASH | • ソート不要<br>• Aggregation 内にいくつかの distinct がある場合、パフォーマンス向上 | • ·大きなメモリが必要 |

大量の Distinct グループがあるクエリの性能を向上させたい場合、次のベストプラクティスを適用することにより、GROUPBY PIPELINED アルゴリズムを使用

- GROUP BY 句で使用しているすべての列がソートされているかプロジェクションを確認する。
- プロジェクションの ORDER BY 句が、クエリの GROUP BY 句の数より多い場合、GROUP BY 句の列が ORDER BY 句の中で初めに指定されていることを確認してください。例外の場合もあり、次のセクションを確認ください。
- クエリの GROUP BY 句に指定されている列が、プロジェクションの ORDER BY 句の最初に指定されていない場合は、ORDER BY 句の最初に指定されている列が、クエリの WHERE 句で定数条件として指定されている列として含まれていることを確認してください。

## プロジェクションのレプリケーション vs. セグメンテーション

データベースデザイナーは、統計情報とクエリをベースにプロジェクションを作成します。クエリパフォーマンスを最適化した設計をするため、データベースデザイナーは、テーブルを調査し、プロジェクションをセグメンテーション（ノードにデータ分割して分散）するか、レプリケーション（すべてのノードに同じデータをコピー）をするかを決定します。

| | **条件** |
|---|---|
| レプリケーション | • サイズが小さなテーブル Small tables<br>• LONG VARCHAR や LONG BINARY の列<br>• 以下の条件を満たすプロジェクション：<br>**（最大の行数は、テーブル内の最も大きな列の行数）**<br>○ 最大の行数が 100 万未満、且つ、最大の行数の 10%以下のテーブル<br>○ 最大の行数が 1000 万未満、且つ、最大の行数の 1%以下のテーブル<br>• 行数が 10 万以下のテーブル |
| セグメンテーション | • ·複数ノードで構成するクラスター内のサイズが大きなテーブル |

## *オートプロジェクションのセグメンテーション*

オートプロジェクションは、自動的にスーパープロジェクションを作成します。オートプロジェクションは、テーブルがプライマリキー（主キー）を定義した場合、プロジェクションはプライマリキーでセグメンテーションされます。もし、プライマリキーが存在しない場合、初めの 32 列までの情報を利用してプロジェクションはセグメンテーションされます。

## データ再配布の最小化

クエリに依存しますが、オプティマイザーは実行時にデータを再配布する必要があることがあります。このプロセスは、過度のネットワークトラフィックや多くのメモリ領域が必要になります。

オプティマイザーは、以下の 2 つの方法でデータを再配布します。

- ブロードキャスティング：中間結果をクラスター内のすべてのノードに完全コピーします。
- リセグメンテーション：既存のプロジェクション、もしくは、中間データセットを取得し、そのデータをすべてのノードに再配布します。

**Join 句では**

| | **条件** |
|---|---|
| ブロードキャスト | • 一つのテーブルが、他のテーブルサイズと比較し、非常に小さい場合、ブロードキャストする。<br>• リセグメンテーションで大きなデータ転送は避けるため、ブロードキャストする。<br>• 外部結合(OUTER JOIN 句) もしくはサブクエリは、結合する片方のテーブルデータをブロードキャストする。 |
| リセグメンテーション | • 分散した結合で利用するローカル結合用のデータが、セグメンテーションされていない場合、リセグメンテーション（再配布）する。 |

複数のテーブルを結合のパフォーマンスを最適化するため、「結合キーでセグメンテーションする」、もしくは、「インナーテーブルをリプリケートする」の方法があります。このどちらかの方法を用いることで、結合処理は、各ノードのローカルで実行され、ノード間のデータ転送をすることなく、クエリ処理をすることができます。

プロジェクションが、クエリの結合キーでセグメンテーションされているかを EXPLAIN のクエリプランで確認することができます。もし、クエリプランに、RESEGMENT もしくは BROADCAST が含まれる場合、プロジェクションは最適化したセグメンテーションをしていません。

## セグメンテーションと Group By

リセグメンテーションを避けるために、GROUP BY 句を含むすべてのセグメンテーション列を確認します。他の列が含まれている可能性もあります。結合を実行する前に、利用している結合キーでその他のノードにブロードキャストしてください。

## *ネットワーク使用量を抑えたデータ再配布*

状況に応じて、ネットワーク負荷がかかることがあります。もし、ネットワーク帯域が問題の場合、CompressNetworkData を 1 に設定することで、その他のノードへのデータ圧縮し、ネットワークの使用量を

抑えることが可能です。この圧縮は、ネットワーク処理を高速化することができますが、CPU の使用量が増加します。

## プロジェクション数のベストプラクティス

特定のクエリでチューニングしたプロジェクションは、クエリのパフォーマンスを向上することができます。しかし、その一方、ディスクの使用量、そして、データロードの時間が増加します。

最適なプロジェクション数をテストする必要があります。クエリスペシフィックプロジェクションは、最適なパフォーマンスで Vertica を運用するため、2〜3 のクエリスペシフィックプロジェクションのみで運用するケースが一般的です。

## 詳細情報

- Reading Query Plans
- Troubleshooting Query Performance with System Tables
- Optimizing Projections
- Working with Projections
- Creating Tables and Projections