データ削除に関するベストプラクティス

March, 2017

原文はこちら

MICRO FOCUS

目次

Vertica 上でのデータの削除

Vertica は、高性能で列ストアの分析データベースです。データを削除する場合、Vertica は従来のデータベースと下記 2 つの点で異なります。

- DELETE 文は、実際にはディスクストレージからデータを削除しません。行を削除済みとしてマークし、履歴クエリでそれらを検索できるようにします。
- UPDATE 文は、次の 2 つのタスクを実行します。新しいデータを書き込み、古いデータに削除済みとしてマークします。

DELETE 文は、ディスク・ストレージからデータを削除しません。代わりに、DELETE 文は、削除されたレコードの位置と、削除がコミットされたときのエポックを記録するデリートベクターを作成します。

データを削除する理由

データを削除する一般的な理由は次のとおりです。

- 履歴データを定期的に削除する必要があります。
- 間違って読み込まれたデータを更新または削除する必要があります。
- ステージングテーブルを削除する必要があります。

異なるタイプの削除

	推奨方法	ロールバックの可否	パフォーマンス	使用用途
単一行削除	WOS	可能	プロジェクション設計に依存	データが移動されたときに、削除された行が 1 つのデリートベクターに結合されるように、WOS 上で削除されることが推奨されます。
トリクルロード	WOS	可能	プロジェクション設計に依存	頻繁な間隔で起こる小さなバッチ処理に適しています。データが移動されたときに、削除された行が 1 つのデリートベクターで結合されるように、WOS 上で削除されることが推奨されます。
バルク削除	ROS へ直接	可能	プロジェクション設計に依存	削除されたとしてマークされるデータを持つ各 ROS に対して 1 つのデリートベクターを作成するために、推奨されます。
パーティション削除	N/A	不可能	高速（カタログ情報の削除の後、バックグランドでストレージ情報は削除されるため）	過去のデータを消去する際に推奨されます。Moveout 処理を強制実行してから、削除するパーティションに属する ROS に挿入されたデータの移動を実行します。

論理削除されたデータを永久に削除する

ディスクスペースを再利用可能にするために、Vertica では、論理削除されたデータをデータベースの物理ストレージから完全に削除することができます。削除されたデータは ROS コンテナから永久に削除されます。Tuple Mover を使用してデータを自動的に削除するか、手動でデータを削除することで、データベースの物理ストレージ内のデータの削除を制御できます。
Ancient History Mark（AHM）は、データがいつ永久に削除されるかを決定します。AHM は、履歴が保持されるまでの時間を表すエポックです。AHM より古いすべての履歴は永久的な削除の対象となります。
Tuple Mover による自動削除の対象となるデータを判別するには、purge（物理削除）のポリシーを設定します。Purge（物理削除）のポリシーを設定するためのベストプラクティスは次のとおりです。

- 削除されたデータを保存する時間を秒単位で指定するには、HistoryRetentionTime という構成パラメーターを使用します。Vertica は、削除されたデータを purge できるかどうかを判断するために、この方法を推奨します。無効にするには、HistoryRetentionTime という構成パラメーターを-1 に設定します。

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '{ <seconds> | -
1 }' );
```

- 保存される履歴エポックの数を指定します。この方法を使用するには、HistoryRetentionTime という構成パラメータを-1 に設定し、保存する履歴エポックの数を設定します。

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', ' -1 ' );
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs',
'{<num_epochs>}');
```

- データを完全に削除するために、削除する必要のある行の割合を指定します。次の例では、削除されるデータのパーセンテージを総データの 20%として設定します。

```
=> SELECT SET_CONFIG_PARAMETER('PurgeMergeoutPercent', '{ Percent| 20
}' );
```

テーブルがパーティション化されていない場合、Mergeout 処理は、特定の ROS で削除された行の PurgeMergeoutPercent を満たすすべての ROS コンテナのデータを永久に削除します。
テーブルがパーティション化されている場合、Mergeout 処理は、PurgeMergeoutPercent の値を満たす非アクティブなパーティションのみのデータを永久に削除します。

デリートベクター管理のアプローチ

削除されたデータを永久に削除する際のオーバーヘッドを考慮して、Vertica はデリートベクターの数を管理することを推奨します。作成されたデリートベクターは、このセクションで説明するように、DELETE 関数の使用を避けるか、バルク削除を実行するかの選択肢を探して管理できます。

データを削除するための代替手段

DELETE 関数を使用せずにデータを削除する別の方法は次のとおりです。

- テーブルからすべてのデータを削除するには、TRUNCATE TABLE 文を使用します。テーブルを truncate すると、そのテーブルとテーブルのプロジェクションに関連するすべてのストレージが削

除されます。TRUNCATE TABLE を使用すると、プロジェクション定義を保持できます。テーブルを truncate すると、カタログ内の依存関係が削除され、バックグラウンドでストレージコンテナが削除されます。

- 履歴データを削除するには、日付フィールドでデータをパーティション化し、DROP PARTITION 関数を使用します。DROP PARTITION は、カタログファイルの依存関係を削除し、Vertica のトランザクションに影響を与えずにバックグラウンドでストレージコンテナを削除します。Vertica は DROP PARTITION を実行する前に Moveout 処理を実行します。削除されるデータが一部のパーティションの場合は、次の手順を実行します。
    1. 元のテーブルと同じテーブル定義を使用してステージングテーブルを作成します。
    2. パーティション内のデータをステージングテーブルに移動します。
    3. データが移動されたら、SWAP_PARTITIONS_BETWEEN_TABLES 関数を使用してパーティションをスワップします。この関数は、データ移動のないカタログ処理です。

たとえば、月ごとにパーティション化されたテーブルのエラーのために先週のデータを再ロードするとします。先週分のデータを削除し、新しいデータを挿入するには、次の手順を実行します。

1. ステージングテーブルを作成します。

```
=> CREATE TABLE store.staging_store_orders_fact like
store.store_orders_fact including projections;
```

2. データをステージングテーブルに移動して、2005-11-20 から 2005-11-27 でデータを削除します。

```
=> INSERT /*+ direct */ into store.staging_store_orders_fact select *
from store.store_orders_fact where date_ordered between '2005-11-01'
and '2005-11-19' or date_ordered between '2005-11-28' and '2005-11-30';
```

3. パーティションをスワップします。

```
=> SELECT
SWAP_PARTITIONS_BETWEEN_TABLES('store.staging_store_orders_fact',200511
,200511,'store.store_orders_fact');
```

4. ステージングテーブルを truncate するか、または drop します。

バルク削除を実行する

DELETE 関数の使用を避けることができない場合は、可能であれば、バルク削除を実行してください。Vertica は、複数の単一削除の代わりにバルク削除を推奨しています。WOS で複数の単一行のデータを削除する必要があります。コミットに失敗すると、DELETE 文ごとに複数のデリートベクターが作成されます。

バルク削除のベストプラクティスは次のとおりです。

- DELETE の Where 句で指定する値を一時テーブルにロードする。
- DELETE の Where 句で指定する値のデータを持つ一時テーブルと、削除するデータを持つテーブルと結合して、1 つの DELETE 文で行を削除する。

たとえば、複数の従業員のレコード（従業員 ID）を削除するとします。1 つの delete 文でレコードを削除すると、1 つの delete 文ごとに 1 つのデリートベクターが作成されます。削除されたデータを含む ROS コンテナごとに 1 つのデリートベクターを作成する 1 つのバルク delete 文を作成することもできます。

次のコードを一部抽出したものは、バルク削除の実行方法を示しています。

```
=> CREATE LOCAL TEMP TABLE data_to_delete (emp_id INT);
CREATE TABLE
=> COPY data_to_delete FROM '/tmp/employee_to_delete.txt' ;
Rows Loaded
-------------
          15
(1 row)

=> DELETE /*+ direct */ FROM store.store_orders_fact WHERE employee_key
IN (SELECT * FROM data_to_delete );
OUTPUT
--------
   1740
(1 row)
=> DROP TABLE data_to_delete ;
DROP TABLE
```

デリートベクターの管理

データを削除する必要がある場合、1 つのデリートベクターを作成してバルク削除を実行するか、複数の単一削除を実行するかを選択できます。複数の単一削除を実行すると、複数のデリートベクターが発生します。

Tuple Mover の mergeout 処理は、次の場合にデリートベクターを削除します。

- 削除は AHM の前に行われる。
- ROS コンテナごとに削除された行は、合計 ROS 行の PurgeMergeoutPercent（デフォルトは 20%）より大きい。

パーティション化されたテーブルでは、mergeout 処理は非アクティブなパーティションのデータを purge しますが、アクティブなパーティションは purge しません。

自動 mergeout がデータを完全に削除していない場合は、デリートベクターを手動で管理する必要があります。ここでは、手動操作について説明します。

デリートベクターと削除された行を確認するには、次のクエリを使用します。このクエリは、イニシエーターノード内の ROS コンテナにデータをフィルターします。ローカルのデータを照会することで、システムテーブルに負荷をかけないようにできます。

```
=> SELECT schema_name
          ,projection_name
          ,count(*) num_ros
          ,sum(total_row_count) num_rows
          ,sum(deleted_row_count) num_deld_rows
          ,sum(delete_vector_count) Num_dv
          ,(sum(deleted_row_count) / sum(total_row_count) * 100)::INT
por_del_rows
```

```
FROM storage_containers
WHERE node_name = ( SELECT local_node_name())
GROUP BY 1, 2
HAVING sum(deleted_row_count) > 0
ORDER BY 5 DESC;

schema_name | projection_name    | num_ros | num_rows | num_deld_rows |
Num_dv |?column?
-------------+----------------------+---------+----------+---------------
---+--------+---
store        | store_orders_fact_b1 | 60 | 200044 |  5636 | 62 |  3
store        | store_orders_fact_b0 | 60 | 200210 |  5618 | 62 |  3
```

テーブルがパーティション化されている場合、次のクエリを使用して、パーティションごとに削除された行を表示することができます。

```
=> SELECT p.node_name
          ,p.Table_schema
          ,p.projection_name
          ,p.partition_key
          ,count(DISTINCT p.ros_id) num_ros -- Number of ROS containers
in projection
          ,sum(p.ros_size_bytes) used_bytes -- Used bytes by the
projecton
          ,sum(p.ros_row_count) num_rows -- Number of rows in projection
          ,sum(p.deleted_row_count) num_del -- Number of deleted rows in
the table
          ,sum(delete_vector_count) cdv-- Number deleted vectors
          ,(sum(sc.deleted_row_count)/sum(p.ros_row_count)*100)::int
por_del_rows -- Percentage of deleted rows per partition.
FROM partitions p inner join storage_containers sc ON ros_id =
storage_oid
WHERE p.node_name = (SELECT local_node_name()) AND sc.node_name = (SELECT
local_node_name())
GROUP BY 1,2,3,4
Having SM(delete_vector_count) > 0
ORDER BY 10 DESC,2,3,4;

node_name|Table_schema|projection_name| partition_key
|num_ros|used_bytes|num_rows|num_del|cdv|por_del_rows
--------------------+-------------+----------------------+-------------
--+---------+-----------+----------
```

```
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200511 | 1 | 244467
| 9978 | 1664 | 9 | 50
v_utn_demo_node0001 | store | store_orders_fact_b1 | 200511 | 1 | 243471
| 9936 | 1650 | 9 | 50
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200311 | 1 | 82143
| 3354 | 84   | 1 | 3
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200407 | 1 | 82799
| 3380 | 90   | 1 | 3
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200601 | 1 | 79787
| 3258 | 82   | 1 | 3
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200703 | 1 | 83788
| 3422 | 88   | 1 | 3
v_utn_demo_node0001 | store | store_orders_fact_b0 | 200712 | 1 | 81095
| 3316 | 92   | 1 | 3
```

上記のクエリの結果に基づいて、次のように 3 つの選択肢があります。

- デリートベクターの Mergeout：デリートベクターが多すぎるが、削除された行が多すぎず、パーティションもない場合、デリートベクターをマージする必要があります。
- パーティションの Purge：パーティションされたデータを purge します（パーシャルパージとも呼ばれます）。
- テーブルの Purge：テーブル全体を書き換えて、データを手動で purge します。

デリートベクターの Mergeout

複数のデリートベクターがあるにもかかわらず、削除された行の割合が表の合計行に比べて少ない場合は、デリートベクターを 1 つのデリートベクターにマージします。デリートベクターのマージは、テーブル全体を purge するよりも優れています。テーブルを purge すると、削除された行がない ROS コンテナのデータセットが書き換えられます。Vertica では、削除された行の数が表の合計行と比較して少ない場合は、テーブルを purge しないことを推奨します。ROS プッシュバックを回避するには、デリートベクターの数を減らしてください。

パフォーマンス低下と ROS プッシュバックを避けるために、mergeout は複数の ROS コンテナを統合し、削除されたレコードを purge します。Tuple Mover は、2 つ以上の ROS コンテナを削除された行なしで 1 つのコンテナに結合することによって自動 mergeout を実行します。ただし、ROS コンテナ当たりのデリートベクターの数が PurgeMergeoutPercent（デフォルトは 20%）より少ない場合、マージアウト処理は削除されたレコードを purge しません。多くの DELETE 文を使用して複数の行を削除すると、削除マークを保持する小さなコンテナが多数作成されます。各コンテナはリソースを消費し、パフォーマンスに影響を与えます。マージのための mergeout サイクルがある場合、Tuple Mover は削除マーカーのコンテナを 1 つの大きなコンテナにマージします。

複数のデリートベクターを 1 つのデリートベクターにマージするには、次のように DO_TM_TASK を使用します。

```
=> SELECT DO_TM_TASK('dvmergeout');
```

パーティションの Purge

1 つのパーティションに削除された行の数が他のパーティションの数より多い場合は、削除された行の数が多い特定のパーティションを purge します。Vertica では、テーブル全体を purge するのではなく、特定のパーティションを purge することを推奨します。テーブル全体を purge すると、削除された行が数個しかない ROS コンテナが書き換えられます。このドキュメントで前述した SELECT schema_name クエリを呼び出して、パーティション内の削除されたデータの場所を返します。cdv と por_del_rows が特定のパーティションの方が高い場合は、そのパーティションだけを purge します。パーティションを purge する手順は次のとおりです。

1. AHM を進ませて、AHM より古い削除マーカーを作成します。AHM が進むと、削除マーカーが削除対象になります。

```
=> SELECT MAKE_AHM_NOW();
```

2. パーティションキーを使用してパーティションを purge します。

```
=> SELECT PURGE_PARTITION('store.store_orders_fact',200511);
```

テーブルデータの Purge

削除されたデータがすべてのパーティション間で分散されているか、またはテーブルがパーティション化されていないが、削除された行が多すぎる場合は、テーブル全体を purge することが最後の選択肢です。

1. AHM を進めて、AHM よりも古い削除マーカーを作成します。AHM が移動すると、削除マーカーが削除対象になります。

```
=> SELECT MAKE_AHM_NOW();
```

2. テーブルを purge、または、プロジェクションを purge します。

```
=> SELECT PURGE('store.store_orders_fact');
OR
=> SELECT PURGE_PARTITION ('store_orders_fact_b0');
```

削除処理のための Vertica の最適化

次のタスクを実行することにより、データベースを削除に最適な状態にすることができます。

- ソート順の終わりにカーディナリティの高い列を使用するか、すべてのプロジェクションに DELETE 文の Where 句で指定する列を含めることで、より良いプロジェクションデザインを作成します。
- データを一時テーブルに挿入して、バッチでデータを削除します。
- 特定のテーブルで 20%以上の行が削除された後にデータを purge します。
- DROP_PARTITION でデータを削除できるように、パーティション化を使用してデータをグループ毎に分けます。
- テーブルを空にする場合は、DELETE 文を使用する代わりに、テーブルを truncate します。

詳細については、Vertica ドキュメントの Optimizing DELETEs and UPDATEs for Performance を参照してください。