クエリのパフォーマンスを分析するためのシステムテーブル

July, 2018

原文はこちら

MICRO FOCUS

目次

システムテーブルからパフォーマンスを評価 4
本ドキュメントについて 4
本ドキュメントに含まれる内容 4
本ドキュメントに含まれない内容 4
クエリの例で vsql の変数を使用 4
最も遅いクエリの特定 5
QUERY_PROFILES システムテーブル 5
クエリ例 5
結果分析 6
パフォーマンスチューニングのためのクエリ実行の分析 7
クエリプランのレビュー 7
QUERY_PLAN_PROFILES システムテーブル 7
クエリ例 7
結果分析 9
クエリ実行のプロファイリング 9
EXECUTION_ENGINE_PROFILES システムテーブル 10
クエリ例 10
結果分析 10
クエリイベントの識別 11
QUERY_EVENTS システムテーブル 12
クエリ例 12
結果分析 14
クエリ実行時間の決定 14
DC_QUERY_EXECUTIONS テーブル 14
クエリ例 14
結果分析 15
トップオペレータの使用状況の特定 16
EXECUTION_ENGINE_PROFILES システムテーブル 16
クエリ例：最も遅いパスと演算子の特定 17
結果分析 18
クエリ例：ノードごとに最も遅いパスと演算子を特定 18
結果分析 19
クエリ例：カウンタの詳細情報の表示 19
結果分析 23
SIPs のパフォーマンスのレビュー 24
DC_SIPS_STATISTICS システムテーブル 24
クエリ例 24
結果分析 25
クエリで使用されるプロジェクションの理解 25
PROJECTION_USAGE システムテーブル 25
クエリ例 25
結果分析 26
PROJECTIONS システムテーブル 26
クエリ例 26

結果分析 27
PROJECTION_COLUMNS システムテーブル 28
クエリ例 28
結果分析 29
PROJECTIONS_STORAGE システムテーブル 30
結果分析 31
テーブルパーティションのレビュー 31
TABLES システムテーブル 31
クエリ例 32
結果分析 32
リソース割り当てのレビュー 32
RESOURCE_ACQUISITIONS システムテーブル 32
クエリ例 32
結果分析 33
本ドキュメントの推奨事項 34
詳細情報 34

システムテーブルからパフォーマンスを評価

クエリのパフォーマンス上の問題が発生した場合、まずVertica Knowledge BaseのRedesigning Projections for Query Optimizationに詳細があるように、実行計画を分析する必要があります。実行計画では、クエリがアクセスするプロジェクションとテーブルの統計情報に基づいて、オプティマイザが最も低コストであると判断する内容が選択されます。しかしながら、実行計画では、クエリの実際の実行内容に関する情報は提供されません。
クエリのパフォーマンスに影響を与える可能性のある問題の詳細については、クエリ実行の詳細を示すいくつかのシステムテーブルが提供されています。
Verticaは、サイズ制限なしにシステムテーブル情報を保持するわけではありません。単位時間当たりに実行されるクエリの数が非常に多い場合、少し経過してから、Verticaがこの情報を破棄する場合があります。システムテーブルのデータ保持サイズを設定する方法については、Vertica documentationのConfiguring Data Retention Policiesを参照してください。

本ドキュメントについて

本ドキュメントに含まれる内容

本ドキュメントでは、クエリのパフォーマンスに関して有用な情報を含むVerticaのシステムテーブルについて説明します。
クエリを実行すると、Verticaは特定のシステムテーブルに情報を格納します。この情報は、クエリの実行中に発生する処理、つまりクエリが実行しているタスク、クエリが使用しているリソース、およびボトルネックが存在する場所を示します。
各クエリのパフォーマンスチューニングのオプションは異なるため、本ドキュメントの推奨事項を各々の特定のクエリに合わせて調整してください。

本ドキュメントに含まれない内容

システムテーブルに加えて、Verticaマネージメントコンソール（MC）は、クエリパフォーマンス分析を容易にするユーザーインターフェイスを提供します。
本ドキュメントでは、MCについては説明しません。本ドキュメントでは、vsqlでSQLクエリを実行して、システムテーブルからクエリのパフォーマンスデータを取得します。
マネージメントコンソールの詳細については、Vertica documentationのManaging Queries in MCを参照してください。

クエリの例で vsql の変数を使用

本ドキュメントでは、VMartスキーマを対象としたサンプルクエリを分析します。これらのサンプルクエリのパフォーマンスを評価するために、Verticaシステムテーブルを照会する多くのSQLの例を掲載しています。これらのサンプルクエリでは、transaction_idフィールドとstatement_idフィールドを使用して、特定のクエリに関するデータにアクセスします。
これらの例を簡単に実行するには、変数を使用してtransaction_idとstatement_idを置き換えます。これらの値を変数に割り当て、サンプルクエリを実行します。

```
\set t_id <your_transaction_id>
\set s_id <your_statement_id>
```

最も遅いクエリの特定

Verticaデータベースでクエリのパフォーマンスを最適化するには、次の2つの方法のいずれかを実行します。

- システムで最も遅いクエリの最適化
- 特定のクエリの最適化

本トピックでは、分析するクエリを特定する方法について説明します。該当のクエリのtransaction_idとstatement_idを取得後、それらのIDを使用して、その特定のクエリを分析するために必要な情報を正確に抽出します。

QUERY_PROFILES システムテーブル

最も遅い問合せを識別するには、QUERY_PROFILES システムテーブルに対してクエリを実行します。

クエリ例

遅いクエリを特定するには、次の手順を実行します。

1. 3時間の間にstore_sales_factテーブルに対する最も遅いクエリを取得するには、QUERY_PROFILESテーブルに対して次のようなクエリを実行します。

```
=> SELECT DATE_TRUNC('second',query_start::TIMESTAMP),
session_id,
transaction_id,
statement_id,
node_name,
LEFT(query,100),
ROUND((query_duration_us/1000000)::NUMERIC(10,3),3
duration_sec
FROM query_profiles
WHERE query ILIKE '% store_sales_fact %' AND
query_start BETWEEN '2016-07-24 19:00:00' AND '2016-07-24 22:00:00'
ORDER BY duration_sec DESC;
```

2. session_idを保持している場合、これらの結果を絞り込んでいきます。

   注意:これらの例では、session_id に変数 sess_id を割り当てたと仮定しています。

   QUERY_PROFILESテーブルに対してクエリを実行し、該当のセッション中に実行されたすべてのクエリおよび実行に要した時間を返します。

```
=> SELECT DATE_TRUNC('second',query_start::TIMESTAMP),
session_id,
transaction_id,
statement_id,
node_name,
```

```
LEFT(query,100),
ROUND((query_duration_us/1000000)::NUMERIC(10,3),3) duration_sec
FROM query_profiles WHERE session_id = :sess_id
ORDER BY duration_sec DESC;
```

3. 指定された時間内に最も遅いクエリを特定するには、次のようなクエリを実行します。

```
=> SELECT (query_duration_us/1000000)::numeric(10,3) duration_sec,
session_id,
transaction_id,
statement_id,
node_name,
left(query,100)
FROM query_profiles
WHERE query_start BETWEEN '2016-07-24 19:00:0' AND '2016-07-24
22:00:00'
ORDER BY duration_sec DESC;
```

結果分析

どのクエリを分析するのかを特定した後、そのクエリのtransaction_idとstatement_idを使用して完全なクエリ文を抽出します。

```
=> SELECT query FROM query_profiles WHERE
transaction_id = :t_id and statement_id = :s_id;
query
-------
SELECT * FROM online_sales_fact
(1 row)
```

クエリ文を保持している場合、PROFILEキーワードを使用してクエリを実行してください。PROFILE文は、すべてのクエリ実行情報をEXECUTION_ENGINE_PROFILESシステムテーブルに保存します。

```
=> PROFILE SELECT * FROM online_sales.online_sales_fact;
NOTICE 4788: Statement is being profiled
HINT: Select * from execution_engine_profiles where transaction_id=:t_id and
statement_id=s_id;
NOTICE 3557: Initiator memory for query: [on pool general: 69620 KB, minimum:
69620 KB]
-[ RECORD 1 ]--------------+---------
sale_date_key | 1730
ship_date_key | 1735
product_key | 7285
```

```
product_version | 3
customer_key | 23378
call_center_key | 61
online_page_key | 447
shipping_key | 66
warehouse_key | 66
promotion_key | 629
pos_transaction_number | 4730816
sales_quantity | 5
sales_dollar_amount | 292
...
```

パフォーマンスチューニングのためのクエリ実行の分析

パフォーマンスの低下の原因を特定するには、実行時間のかかるクエリを特定した後、詳細情報が必要になります。次のトピックでは、パフォーマンスチューニングに役立つ情報を提供するVerticaシステムテーブルのクエリとデータを紹介します。

クエリプランのレビュー

クエリの実行中にデータフローを理解できるように、クエリプランをレビューすると同時にクエリ実行を分析します。

注意：マネージメントコンソールを使用してクエリプランを表示するには、Vertica documentation の Accessing Query Plans in Management Console を参照してください。

QUERY_PLAN_PROFILES システムテーブル

クエリを実行すると、VerticaはクエリプランをQUERY_PLAN_PROFILESシステムテーブルに保存します。

クエリ例

次のクエリは、クエリプランと、特定のパスに割り当てられた実行時間やメモリなどの情報を返します。このクエリでは、表現上の目的でパスの説明が120文字で切り捨てられています。

```
=> SELECT path_id,path_line_index pos, running_time, memory_allocated_bytes as mem_alloc,
read_from_disk_bytes read_from_disk, LEFT(path_line, 70)
FROM query_plan_profiles WHERE transaction_id = :t_id AND statement_id = :s_id ORDER BY
path_id,path_line_index;
path_id | pos | running_time | mem_alloc | read_from_disk | left
---------+-----+----------------+-----------+----------------+--------------------------
-----------------------------------------
2 | 1 | 00:00:00.765351 | 746200704 | | +-GROUPBY HASH (SORT OUTPUT) (GLOBAL RESEGMENT
GROUPS) (LOCAL RESEGMEN
2 | 2 | | | | Group By: s.product_key, p.product_description
```

```
2 | 3 | | | | | Execute on: All Nodes
3 | 1 | 00:00:00.732285 | 28023488 | | | +---> JOIN HASH [Semi] [Cost: 21K, Rows: 2M]
(PATH ID: 3) Inner (BRO
3 | 2 | | | | | | Join Cond: (s.store_key = VAL(3))
3 | 3 | | | | | | Materialize at Input: s.store_key
3 | 4 | | | | | | Materialize at Output: s.product_key
3 | 5 | | | | | | Execute on: All Nodes
4 | 1 | 00:00:00.72732 | 37829248 | | | | +-- Outer -> JOIN HASH [Cost: 14K, Rows: 5M]
(PATH ID: 4) Inner (B
4 | 2 | | | | | | | Join Cond: (s.product_key = p.product_key) AND (s.product_v
4 | 3 | | | | | | | Execute on: All Nodes
5 | 1 | 00:00:00.591365 | 15884032 | 0 | | | | +-- Outer -> STORAGE ACCESS for s [Cost:
6K, Rows: 5M] (PATH ID:
5 | 2 | | | | | | | | Projection: store.store_sales_fact_b0
5 | 3 | | | | | | | | Materialize: s.product_key, s.product_version
5 | 4 | | | | | | | | Execute on: All Nodes
5 | 5 | | | | | | | | Runtime Filters: (SIP2(HashJoin): s.product_key), (SIP3(H
6 | 1 | 00:00:00.613828 | 28816384 | 0 | | | | +-- Inner -> STORAGE ACCESS for p [Cost:
266, Rows: 60K] (PATH I
6 | 2 | | | | | | | | Projection: public.product_dimension_b0
6 | 3 | | | | | | | | Materialize: p.product_key, p.product_version, p.product_
6 | 4 | | | | | | | | Execute on: All Nodes
8 | 1 | 00:00:00.51101 | 32135168 | 0 | | | | +---> STORAGE ACCESS for store_dimension
[Cost: 34, Rows: 16] (P
8 | 2 | | | | | | | | Projection: store.store_dimension_b0
8 | 3 | | | | | | | | Materialize: store_dimension.store_key
8 | 4 | | | | | | | | Filter: (store_dimension.store_state = 'MA')
8 | 5 | | | | | | | | Execute on: All Nodes
(25 rows)
```

クエリプランが大きすぎて分析できない場合、最も遅いパスと最も遅いパスに隣接するパスに集中して分析します。

```
=> \set path_id 4
=> SELECT path_id,path_line_index pos, running_time, memory_allocated_bytes
as mem_alloc, read_from_disk_bytes read_from_disk, LEFT(path_line, 70)
FROM query_plan_profiles WHERE transaction_id = :t_id AND statement_id =
:s_id AND path_id BETWEEN :path_id -2 AND :path_id +2 ORDER BY path_id,path_
line_index;
path_id | pos | running_time | mem_alloc | read_from_disk | left
---------+-----+-----------------+-----------+----------------+--------------
-------------------------------------------------------------
```

```
2 | 1 | 00:00:00.765351 | 746200704 | | +-GROUPBY HASH (SORT OUTPUT) (GLOBAL
RESEGMENT GROUPS) (LOCAL RESEGMEN
2 | 2 | | | | | Group By: s.product_key, p.product_description
2 | 3 | | | | | Execute on: All Nodes
3 | 1 | 00:00:00.732285 | 28023488 | | | +---> JOIN HASH [Semi] [Cost: 21K,
Rows: 2M] (PATH ID: 3) Inner (BRO
3 | 2 | | | | | | Join Cond: (s.store_key = VAL(3))
3 | 3 | | | | | | Materialize at Input: s.store_key
3 | 4 | | | | | | Materialize at Output: s.product_key
3 | 5 | | | | | | Execute on: All Nodes
4 | 1 | 00:00:00.72732 | 37829248 | | | | +-- Outer -> JOIN HASH [Cost: 14K,
Rows: 5M] (PATH ID: 4) Inner (B
4 | 2 | | | | | | | Join Cond: (s.product_key = p.product_key) AND
(s.product_v
4 | 3 | | | | | | | Execute on: All Nodes
5 | 1 | 00:00:00.591365 | 15884032 | 0 | | | | +-- Outer -> STORAGE ACCESS
for s [Cost: 6K, Rows: 5M] (PATH ID:
5 | 2 | | | | | | | | Projection: store.store_sales_fact_b0
5 | 3 | | | | | | | | Materialize: s.product_key, s.product_version
5 | 4 | | | | | | | | Execute on: All Nodes
5 | 5 | | | | | | | | Runtime Filters: (SIP2(HashJoin): s.product_key),
(SIP3(H
6 | 1 | 00:00:00.613828 | 28816384 | 0 | | | | +-- Inner -> STORAGE ACCESS
for p [Cost: 266, Rows: 60K] (PATH I
6 | 2 | | | | | | | | Projection: public.product_dimension_b0
6 | 3 | | | | | | | | Materialize: p.product_key, p.product_version,
p.product_
6 | 4 | | | | | | | | Execute on: All Nodes
(20 rows)
```

結果分析

クエリの実行時間、使用されたメモリの量、およびディスクから読み取られたデータに注意して、異常な内容がクエリのパフォーマンスに影響を与えているかどうかを確認します。

クエリ実行のプロファイリング

リアルタイムのプロファイリングにより、実行中の長時間実行クエリをモニターすることができます。特定のSQL文でPROFILEキーワードを使用してプロファイリングを有効にするか、データベースおよび/または現セッションをプロファイルできます。プロファイリングを有効にしていない場合、ステートメントの完了後にプロファイリングのカウンタ情報が利用できません。

EXECUTION_ENGINE_PROFILES システムテーブル

EXECUTION_ENGINE_PROFILESシステムテーブルには、クエリの実行に関するプロファイル情報が含まれています。

クエリ例

PROFILEキーワードを使用してクエリを実行します。

```
\timing -- to return the timing
\pset format unaligned -- To minimize perturbations due to result set alignment, esp. very large
result sets
\o /dev/null -- to discard the output and not present it when executing
profile -- keyword to force storing the information in execution_engine_profiles.
SELECT DISTINCT s.product_key, p.product_description
FROM store.store_sales_fact s, public.product_dimension p
WHERE s.product_key = p.product_key
AND s.product_version = p.product_version
AND s.store_key IN (
SELECT store_key
FROM store.store_dimension
WHERE store_state = 'MA')
ORDER BY s.product_key;
\o
```

結果分析

前述のクエリは、次の情報を返し、プロファイリング情報をEXECUTION_ENGINE_PROFILESシステムテーブルに格納します。

```
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996274879950 and statement_id=1;
NOTICE 3557: Initiator memory for query: [on pool general: 1548488 KB, minimum: 1209481 KB] <==
The query executed in the GENERAL resource pool
NOTICE 5077: Total memory required by query: [1548488 KB] <== Memory required for the query, which is limited by the pool where the query is executing
Time: First fetch (1000 rows): 718.507 ms. All rows formatted: 832.033 ms <= How long it took to execute the query
```

プロファイル情報は、リソースプールによって提供されるメモリがクエリを実行するのに十分であるかどうかを示します。メモリが不足している場合、プロファイル分析のために別のプールを使用します。別のプールを割り当てるには、次のステートメントを使用します。該当のリソースプールを使用するために適切な権限設定が必要です。

```
=> SET SESSION RESOURCE_POOL = <desired_pool>
```

クエリイベントの識別

実行されたクエリごとに2つのタイプのイベントが発生します。

- オプティマイザイベントは、オプティマイザがクエリプランを準備している間に発生します。
- 実行イベントは、クエリの実行中に発生します。

クエリイベントは、正または負のイベントが含まれます。オプティマイザのクエリイベントの例を次の表に示します。

オプティマイザイベント

ネガティブオプティマイザイベント	説明
DELETE_WITH_NON_OPTIMIZED_PROJECTION	オプティマイザは、プロジェクションの行を削除するために、より遅いパスを使用する必要がありました。
MEMORY_LIMIT_HIT	オプティマイザは、クエリプラン作成中に割り当てられたすべてのメモリを使用しました。
NO_HISTOGRAM	オプティマイザは、ヒストグラムを持たない列に対して述語を検出しました。

ポジティブオプティマイザイベント	説明
GROUPBY_PUSHDOWN	オプティマイザは GroupBy をジョインを超えてプッシュしました。
NO_GROUPBY_PUSHDOWN	オプティマイザは、GroupByをジョインを超えてにプッシュできませんでした。
NODE_PRUNING	オプティマイザは、特定の数の Vertica ノードをプロジェクションアクセスからプルーニングしました。
TRANSITIVE_PREDICATE	オプティマイザは結合条件のために Transitive Predicate を作成しました。

実行エンジンイベント

ネガティブ実行エンジンイベント	説明
GROUP_BY_SPILLED	GROUP BYキーセットがメモリに収まらなかった。外部のソートグループを使用します。WOSがいっぱいである。
JOIN_SPILLED	内部結合はメモリに収まりませんでした。外部ソートマージ結合を使用します。
RESEGMENTED_MANY_ROWS	プラン実行中に多くの行が再分散されました。
WOS_SPILL	データが新しい ROS コンテナにスピルしています。

ポジティブ実行エンジンイベント	説明
GROUP_BY_PREPASS_FALLBACK	メモリ不足のためにインメモリのprepassが無効になりました。
MERGE_CONVERTED_TO_UNION	MERGE は UNION に変換され、その後に SORT が続きます。
PARTITIONS_ELIMINATED	関連するデータが含まれていないため、一部のストレージコンテナは処理されませんでした。
RLE_OVERRIDDEN	平均実行回数が十分でないため、一部の列で圧縮実行が使用されていませんでした。

SEQUENCE_ CACHE_ REFILLED	シーケンスのセッションキャッシュが使い尽くされました。GLOBAL CATALOG X ロックを取ってキャッシュを補充します。
SIP_FALLBACK	効果がないため、Sidewise Information Passing（SIPs）フィルタが無効になりました。
SMALL_MERGE_REPLACED	SmallStorageMerge は効率化のために StorageUnion に置き換えられました。

QUERY_EVENTS システムテーブル

QUERY_EVENTSは、クエリの実行中に発生したすべてのイベントをリストします。

クエリ例

特定のトランザクションとステートメントのすべてのクエリイベントを表示するには、次のクエリを使用します。

```
=> SELECT event_type, event_category, COUNT(DISTINCT node_name), COUNT(*) FROM
query_events WHERE
transaction_id = :t_id AND statement_id = :s_id GROUP BY 1,2 ORDER BY 2;
event_type | event_category | count | count
-------------------------+----------------+-------+-------
MERGE_CONVERTED_TO_UNION | EXECUTION | 3 | 3
SMALL_MERGE_REPLACED | EXECUTION | 3 | 9
(2 rows)
```

イベントが発生した際のパスの詳細を確認するには、次のクエリを使用します。

注意：パスは、クエリプランのサブオペレーションを示す Vertica のコストベースのクエリオプティマイザの実行方法です。

```
=> SELECT node_name, event_type, event_description, operator_name, path_id,
event_details,
suggested_action FROM execution_engine_events WHERE transaction_id = :t_id
AND statement_id = :s_id
ORDER BY node_name;
-[ RECORD 1 ]-----+-------------------------------------------------
node_name | v_vmart_node0001
event_type | MERGE_CONVERTED_TO_UNION
event_description | Merge converted to union, followed by sort.
operator_name | Sort
path_id | -1
event_details | Projection: public.result_table_b0
suggested_action |
-[ RECORD 2 ]-----+-------------------------------------------------
node_name | v_vmart_node0001
event_type | SMALL_MERGE_REPLACED
```

```
event_description | Small StorageMerge replaced with StorageUnion for
efficiency
operator_name | StorageMerge
path_id | 4
event_details | Projection: public.new_addresses_b0
suggested_action |
-[ RECORD 3 ]-----+-------------------------------------------------node_name |
v_vmart_
node0001
event_type | GROUP_BY_SPILLED
event_description | GROUP BY key set did not fit in memory, using external
sort grouping.
operator_name | GroupByHash
path_id | 4
event_details |
suggested_action | Consider a sorted projection. Increase memory available to
the plan.
-[ RECORD 4 ]-----+-------------------------------------------------node_name |
v_vmart_
node0001
event_type | RESEGMENTED_MANY_ROWS
event_description | Many rows were resegmented during plan execution.
operator_name | NetworkSend
path_id | 2
event_details |
suggested_action | Consider different projection segmentation.
-[ RECORD 5 ]----+-----------------------------------------------
...
-[ RECORD 11 ]----+----------------------------------------------
node_name | v_vmart_node0003
event_type | SMALL_MERGE_REPLACED
event_description | Small StorageMerge replaced with StorageUnion for
efficiency
operator_name | StorageMerge
path_id | 4
event_details | Projection: public.new_addresses_b0
suggested_action |
...
```

結果分析
すべてのネガティブイベントについては、suggested_actionフィールドを確認してください。該当のクエリのパフォーマンスが向上するかどうかを確認するための提案が記載されています。

クエリ実行時間の決定
正常なシステムでは、クエリの実行はExecutePlanフェーズで実行されます。クエリ実行の他のフェーズで多大な時間を費やしている場合、Verticaサポートチームによる詳細な分析が必要なシステムの問題が発生している可能性があります。

DC_QUERY_EXECUTIONS テーブル
各クエリフェーズの実行時間を確認するには、DC_QUERY_EXECUTIONSテーブルに対してクエリを実行します。イニシエーターノードの実行時間は、イニシエーターノードが余分なタスクを実行する必要があるため、非イニシエータノードとは異なります。

クエリ例
次のクエリでは、v_vmartdb_node0001（ローカルノード）がイニシエータノードです。予想通り、ExecutePlanフェーズは他のフェーズよりもはるかに長いです。

```
=> SELECT dc_query_executions.node_name, dc_query_executions.transaction_id,
dc_query_executions.statement_id, dc_query_executions.execution_step,
((dc_query_executions.completion_time - dc_query_executions."time")) AS duration FROM
v_internal.dc_query_executions WHERE transaction_id = :t_id
AND statement_id = :s_id and node_name = (select local_node_name()) ORDER BY
dc_query_executions."time";
node_name | transaction_id | statement_id | execution_step | duration
--------------------+-------------------+--------------+------------------------------+---
--------------
v_vmartdb_node0001 | 45035996274879950 | 1 | Plan | 00:00:00.020269
v_vmartdb_node0001 | 45035996274879950 | 1 | InitPlan | 00:00:00.001941
v_vmartdb_node0001 | 45035996274879950 | 1 | SerializePlan | 00:00:00.001025
v_vmartdb_node0001 | 45035996274879950 | 1 | PreparePlan | 00:00:00.01296
v_vmartdb_node0001 | 45035996274879950 | 1 | PreparePlan:TakeTableLocks | 00:00:00.000005
v_vmartdb_node0001 | 45035996274879950 | 1 | PreparePlan:DistPlanner | 00:00:00.000565
v_vmartdb_node0001 | 45035996274879950 | 1 | PreparePlan:LocalPlan | 00:00:00.001329
v_vmartdb_node0001 | 45035996274879950 | 1 | PreparePlan:EEcompile | 00:00:00.00162
v_vmartdb_node0001 | 45035996274879950 | 1 | CompilePlan | 00:00:00.009477
v_vmartdb_node0001 | 45035996274879950 | 1 | CompilePlan:ReserveResources |
00:00:00.000046
v_vmartdb_node0001 | 45035996274879950 | 1 | CompilePlan:EEpreexecute | 00:00:00.005646
v_vmartdb_node0001 | 45035996274879950 | 1 | ExecutePlan | 00:00:00.756467
v_vmartdb_node0001 | 45035996274879950 | 1 | AbandonPlan | 00:00:00.004101
(13 rows)
```

同様のクエリは、実行ノードv_test_db_node0002が次のフェーズで時間を費やしたことを示しています。

```
v_test_db_node0002 | 45035996273718437 | 8 | PreparePlan:DeserializePlan |
00:00:00.009016
v_test_db_node0002 | 45035996273718437 | 8 | PreparePlan:TakeTableLocks |
00:00:00.000009
v_test_db_node0002 | 45035996273718437 | 8 | PreparePlan:DistPlanner |
00:00:00.000168
v_test_db_node0002 | 45035996273718437 | 8 | PreparePlan:LocalPlan |
00:00:00.002264
v_test_db_node0002 | 45035996273718437 | 8 | PreparePlan:EEcompile |
00:00:00.009847
v_test_db_node0002 | 45035996273718437 | 8 | CompilePlan:ReserveResources |
00:00:00.000116
v_test_db_node0002 | 45035996273718437 | 8 | CompilePlan:EEpreexecute |
00:00:00.019912
```

結果分析

これらの結果を理解するには、各クエリ実行フェーズで何が起こっているのかを理解する必要があります。次の表に、各フェーズとそのフェーズでクエリのパフォーマンスに影響を与える可能性のある内容を示します。

フェーズ	説明	パフォーマンス問題の原因
Plan, InitPlan, Serialize Plan, AbandonPlan	これらのフェーズは、イニシエーターノードでのみ発生します。	これらのフェーズでの速度の低下は、オプティマイザがクエリの実行計画を作成するのに予想以上に時間がかかったことを意味します。 これらのフェーズでの速度の低下は、一般的に同時実行性に関係します。オプティマイザは、クエリの実行計画を作成するためにカタログロックをとる必要があります。 あるいは、これらのフェーズでの速度の低下は、Vertica が spread を使用して他のノードに実行計画を送信したり、AbandonPlan メッセージを他のノードに送信したりするときに、UDP の問題を示唆している可能性があります。
ExecutePlan	このフェーズは、クエリの実際の実行です。 Verticaはクエリの実行に関する詳細をEXECUTION_ENGINE_PROFILESに保存します	ExecutePlanフェーズでクエリの速度の低下が見られる場合、EXECUTION_ENGINE_PROFILESテーブルの次のいくつかのクエリを使用して、速度の低下の根本原因を特定します。
CompilePlan	このフェーズは、すべてのノードで実行され、2 つのパートで構成されます。 • 最初のパート、ReserveResources フェーズでは、Vertica プロセスがリソースを予約するのに要した時間を示します。 • 2 番目のパートは EEpreeexecute フェーズです。EEpreexecute フェーズは、特定のオペレータを実行するためにシステムを準備します。	EEPrexecute 時間の長さは、オペレータによって異なり、メモリの割り当て、スレッドの開始、ネットワーク接続の開始などのタスクを含む可能性があります。

トップオペレータの使用状況の特定

オペレータは、データを処理し、データを次のオペレータに移動するExecution Engineコンポーネントです。特定のオペレータでクエリが費やした時間を確認すると、パフォーマンスの問題の原因を特定できます。

EXECUTION_ENGINE_PROFILES システムテーブル

EXECUTION_ENGINE_PROFILESテーブルには、各オペレータのカウンタ初期化時間（us）カラムに各オペレータが費やした時間が格納されます。ExecutePlanフェーズでクエリが遅い場合、そのフェーズに関する詳細情報をEXECUTION_ENGINE_PROFILESシステムテーブルで確認できます。このテーブルには何千もの行が含まれる可能性があるため、データを集約してこのテーブルの内容を調べることをお薦めします。
最良の結果を得るには、オペレータ、クエリプランパスID、およびノード名で行を集計します。
クエリプランには、次のオペレータが表示されることがあります。

オペレータ	説明
Copy	ロード中、バディープロジェクションのためにデータのコピーを作成します。
DataTarget	ロード中、WOSまたはROSにデータを書き込みます。
ExprEval	式を評価します（例：Column1 + Column2）。クエリプランでは、Vertica が式を評価するために必要な列のみが選択されます。
Filter	タプルを次のオペレータにフィルタリングします。
GroupByHash	タプルをメモリー内のハッシュに集約します。 GroupByHash は使用可能なすべてのメモリを使用します。 十分なメモリがない場合、データがディスクにスピルします。このオペレータは、次のオペレータが開始する前に完了しなければなりません。
GroupByPipe	データを次のオペレータにストリームするためにソートされたタプルを集約します。 GroupbyHash より少ないメモリを使用します。
JoinMergejoin	事前ソートされたタプルに結合します。 ハッシュ結合より少ないメモリを使用します。
JoinHash	結合の内側をメモリにロードすることで、事前ソートされていないタプルを結合します。 内部結合が大きく、メモリに収まらない場合、クエリは失敗します。 内部結合が小さい場合、JoinHash は JoinMerge より高速になります。
Load	ディスクからデータをロードし、入力を解析します。
Merge	データストリームを 1 つのソートされたストリームにマージします。
NetworkRecv NetworkSend	他のノードとの間で送受信されるデータ量。 データは 1 つのスレッドでストリームされます。各ペアのネットワークオペレータ（送受信）に対して、Vertica はクエリに追加メモリを予約します。 データバッファのメモリ要求は、ノードの数に比例して増加します。
ParallelMerge	ソートされたデータストリームを結合します。
ParallelUnion	必ずしもソートされていないデータのストリームを結合します。
Root	クエリ実行の最初のオペレータ。

Scan	ディスクからデータを読み取り、フィルタを適用します。
Sort	データストリームをソートします。
StorageMerge	ソート順を保持して、ストレージを結合します。
StorageUnion	ソート順を保持せずにストレージを結合します。
TopK	上位 N 個のタプルを返す解析関数。
ValExpr	tableA.C1 = tableB.c2 などの結合の式を評価します。

クエリ例:最も遅いパスと演算子の特定

次のクエリは、最も遅いパスと、クラスター内のそのパスで実行していたExecution Engineオペレータを識別するのに役立ちます。多数のパスとオペレータを使用する遅いクエリでは、パフォーマンスの問題が発生する可能性が高いパスやオペレータに分析を集中させます。

注意:このクエリの集計は、並列に実行されているノードまたはスレッドの数を考慮せずに各オペレータの合計を計算するため誤解を招く可能性があります。

```
=> SELECT operator_name, path_id, SUM(counter_value) FROM execution_engine_profiles WHERE
transaction_id = :t_id AND statement_id = :s_id AND counter_name ILIKE 'execution%' GROUP
BY
operator_name, path_id ORDER BY 3 DESC LIMIT 20;
operator_name | path_id | sum
---------------+---------+---------
Scan | 5 | 1501914
Join | 4 | 659055
GroupByHash | 2 | 307442
StorageUnion | 2 | 120715
Join | 3 | 111232
Root | -1 | 34964
ExprEval | 3 | 31785
NetworkSend | 2 | 27947
ParallelUnion | 2 | 13525
GroupByPipe | 2 | 7594
NetworkRecv | 2 | 6775
Scan | 6 | 6292
NetworkSend | 6 | 5328
NetworkRecv | 6 | 3683
GroupByPipe | 8 | 1865
StorageUnion | 6 | 1312
```

```
NetworkRecv | 8 | 1118
NewEENode | -1 | 955
Scan | 8 | 879
ParallelMerge | 2 | 715
(20 rows)
```

結果分析

このデータを使用すると、クラスター全体で実行するのに長時間かかるオペレータを識別できます。次のトピックのクエリを使用して、どのノードのクエリが遅いかを絞り込みます。

クエリ例：ノードごとに最も遅いパスと演算子を特定

ノードごとに最も遅いパスとオペレータを表示するには、次の例のようにnode_nameで集計します。1つのノードが他のノードよりも実行時間が長い場合、その原因としてはデータのスキューなどが考えられます。

```
=> SELECT node_name, operator_name, path_id, SUM(counter_value) sum_time, COUNT(DISTINCT
operator_id) num_operators
FROM dc_execution_engine_profiles WHERE transaction_id = :t_id AND statement_id = :s_id
AND counter_name ILIKE
'execution%' GROUP BY node_name, operator_name, path_id ORDER BY 4 DESC LIMIT 20;
node_name | operator_name | path_id | sum_time | num_operators
-------------------+---------------+---------+----------+---------------
v_vmartdb_node0001 | Scan | 5 | 544160 | 2
v_vmartdb_node0003 | Scan | 5 | 498164 | 2
v_vmartdb_node0002 | Scan | 5 | 459590 | 2
v_vmartdb_node0001 | Join | 4 | 225466 | 2
v_vmartdb_node0002 | Join | 4 | 223134 | 2
v_vmartdb_node0003 | Join | 4 | 210455 | 2
v_vmartdb_node0002 | GroupByHash | 2 | 106011 | 4
v_vmartdb_node0001 | GroupByHash | 2 | 105288 | 4
v_vmartdb_node0003 | GroupByHash | 2 | 96143 | 4
v_vmartdb_node0001 | StorageUnion | 2 | 40551 | 2
v_vmartdb_node0002 | StorageUnion | 2 | 40251 | 2
v_vmartdb_node0003 | StorageUnion | 2 | 39913 | 2
v_vmartdb_node0001 | Join | 3 | 37540 | 2
v_vmartdb_node0003 | Join | 3 | 37466 | 2
v_vmartdb_node0002 | Join | 3 | 36226 | 2
v_vmartdb_node0001 | Root | -1 | 34964 | 1
v_vmartdb_node0001 | ExprEval | 3 | 10858 | 2
v_vmartdb_node0001 | NetworkSend | 2 | 10780 | 2
v_vmartdb_node0003 | ExprEval | 3 | 10543 | 2
v_vmartdb_node0002 | ExprEval | 3 | 10384 | 2
(20 rows)
```

結果分析

num_operators列は、クエリ結果を計算するために並列に実行されたオペレータの数を示します。EXECUTIONPARALLELISMリソースプールパラメーターを使用して、この数を管理できます。ただし、ROSコンテナの数などの他の条件に応じて、同時オペレータの数は、リソースプールパラメーターが指定する数より少なくなることがあります。
SCANオペレータが遅く、1ノードが他のノードより遅い場合、そのノードは他のノードより遅いディスクを持つ可能性があります。その場合、vioperfを使用してノードのI / Oスループットを確認します。
NetworkSendオペレータが遅い場合、ネットワークに問題がある可能性があります。netstatを使用して、TCPパッケージの問題がないかどうかを確認します。

クエリ例:カウンタの詳細情報の表示

最も遅いパスを特定したら、そのパスの各オペレータのカウンタから詳細を取得します。
各オペレータには異なるカウンタがあります。次の表は、クエリのパフォーマンスについていくつかの洞察を与えるカウンタについて説明しています。

カウンタ	説明
execution time (us)	待ち時間を除いた、スレッドによって費やされたCPU時間。
clock time (us)	待ち時間を含むオペレータの時間間隔。
initialization time (us)	オペレータの初期化に費やされた時間。 この時間には、メモリの割り当て、スレッドの開始、ネットワーク接続の開始などのタスクが含まれます。各オペレータは異なるタスクを実行します。
start time/end time (us)	1 つのオペレータの開始/停止時間。
rows processed	オペレータによって処理されたデータ行。
input queue wait (us)	実行エンジンが上流のオペレータを待っている時間。
memory reserved (bytes)	オプティマイザによって要求されたメモリ。 オプティマイザは統計を使用して、必要なメモリ量を見積もります。
memory allocated (bytes)	クエリを実行するときに実行エンジンのオペレータによって割り当てられたメモリ。クエリが開始されると、メモリは予約されますが、オペレータが必要とするまでメモリは割り当てられません。
file handles	開く必要があるファイルの数。この数は、必要な情報を読み取るために開く必要のある列とROSコンテナの数によって異なります。
bytes received/bytes sent	クエリによって送受信されたバイト数。
rows received/rows sent	クエリによって送受信されたデータ行数。
RLE rows produced	ディスクに格納されているRLE形式のままでオペレータによって生成されたタプルの数。Verticaがまだ列をマテリアライズしておらず、オペレータが圧縮データを処理できることを示しています。
rows produced	オペレータによって生成された論理行数。
consumer stall (us)	前のオペレータからデータを取得するためにオペレータが待機した時間。
size of raw temp data (bytes)	counter_valueは、ディスクにspillされたデータのサイズを示します。

次のクエリは、ローカルノード上のローカルパス内のすべてのカウンタの詳細と平均値を返します。

すべてのノード上の実行時間が類似している場合、ローカルノードだけでデータをフィルタリングして、分析クエリの実行時間を短縮し、リソースを削減します。解析をローカルノードだけにフィルタリングするには、LOCAL_NODE_NAME関数を使用します。あるノードが他のノードよりも遅い場合、最も遅いノードで分析を実行します。
count列は、並列で実行された各オペレータのインスタンス数を示します。avg列は、すべてのノードのカウンタあたりの平均値を表します。

```
=> \set path_id 5
=> SELECT operator_name,counter_name,path_id, COUNT(DISTINCT operator_id),
AVG(counter_value) FROM execution_engine_profiles WHERE transaction_id =
:t_idAND statement_id = :s_idAND path_id = :path_id AND node_name = ( select
LOCAL_NODE_NAME()) GROUP BY 1,2,3 HAVINGSUM(counter_value) > 0 ORDER BY
1;
operator_name | counter_name | path_id | count | avg
---------------+-----------------------------------------------+---------+-------+--------
----------
Scan | blocks analyzed by SIPs expression | 5 | 2 | 40636
Scan | bytes read from cache | 5 | 2 | 23560280
Scan | bytes read from disk | 5 | 2 | 7970589
Scan | clock time (us) | 5 | 2 | 647620
Scan | current memory padding (bytes) | 5 | 2 | 2592620
Scan | current unbalanced memory allocations (count) | 5 | 2 | 330
Scan | current unbalanced memory capacity (bytes) | 5 | 2 | 4210688
Scan | current unbalanced memory overhead (bytes) | 5 | 2 | 6720
Scan | current unbalanced memory padding (bytes) | 5 | 2 | 996
Scan | current unbalanced memory requested (bytes) | 5 | 2 | 2026748
Scan | end time | 5 | 2 | 2914941832314339
Scan | estimated rows produced | 5 | 2 | 60000000
Scan | execution time (us) | 5 | 2 | 544160
Scan | initialization time (us) | 5 | 2 | 430
Scan | memory allocated (bytes) | 5 | 2 | 4488072
Scan | number of cancel requests received | 5 | 2 | 12
Scan | peak file handles | 5 | 2 | 12
Scan | peak memory allocations (count) | 5 | 2 | 54
Scan | peak memory padding (bytes) | 5 | 2 | 2592620
Scan | peak memory requested (bytes) | 5 | 2 | 1693456
Scan | peak unbalanced memory allocations (count) | 5 | 2 | 332
Scan | peak unbalanced memory capacity (bytes) | 5 | 2 | 4210688
Scan | peak unbalanced memory overhead (bytes) | 5 | 2 | 6720
Scan | peak unbalanced memory padding (bytes) | 5 | 2 | 996
Scan | peak unbalanced memory requested (bytes) | 5 | 2 | 2092284
```

```
Scan | rle rows produced | 5 | 2 | 404250
Scan | rows filtered by SIPs expression | 5 | 2 | 9595008
Scan | rows processed | 5 | 2 | 9999258
Scan | rows processed by SIPs expression | 5 | 2 | 10060698
Scan | rows produced | 5 | 2 | 404250
Scan | start time | 5 | 2 | 2914941831237368
(31 rows)
```

次のクエリは、パスとオペレータを簡単に比較できるように、特定のカウンタ値をピボットされた値として識別します。あなたの特定のユースケースはあなたが興味を持っているカウンタを指示します。

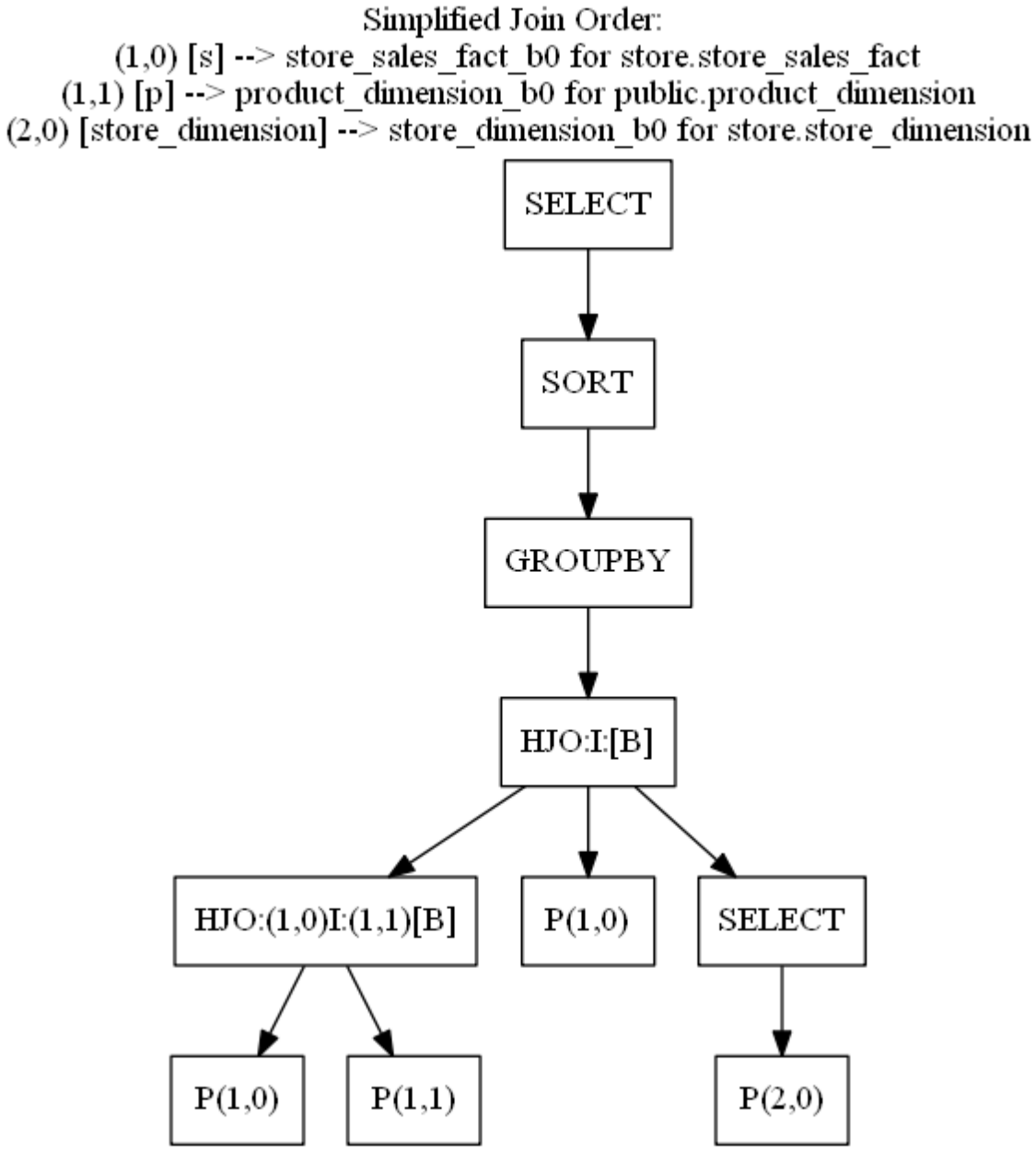
```
=> SELECT path_id, operator_name, COUNT(DISTINCT operator_id) num_operators,
COUNT(DISTINCT node_name) num_nodes, SUM(DECODE(counter_name, 'bytes
received', counter_value, NULL)) AS 'bytes_received', SUM(DECODE(counter_name, 'bytes
sent', counter_value, NULL)) AS 'bytes_sent'
, SUM(DECODE(counter_name, 'execution time (us)', counter_value, NULL)) AS
'execution_time_us'
, SUM(DECODE(counter_name, 'rows received', counter_value, NULL)) AS 'rows_received'
, SUM(DECODE(counter_name, 'rle rows produced', counter_value, NULL)) AS
'rle_rows_produced'
, SUM(DECODE(counter_name, 'rows produced', counter_value, NULL)) AS 'rows_prod'
, SUM(DECODE(counter_name, 'consumer stall (us)', counter_value, NULL)) AS 'cons_stall'
, SUM(DECODE(counter_name, 'producer stall (us)', counter_value, NULL)) AS 'prod_stall'
, SUM(DECODE(counter_name, 'cumulative size of raw temp data (bytes)', counter_value,
NULL)) AS ' temp_data' FROM execution_engine_profiles WHERE
transaction_id=:t_idAND statement_id=:s_idAND node_name = (SELECT LOCAL_NODE_NAME())
GROUP BY 1, 2 ORDER BY 1,2 ;
path_id | operator_name | num_operators | num_nodes | bytes_received | bytes_sent |
execution_time_us | rows_received | rle_rows_produced | rows_
produced | consumer_stall | producer_stall | cumulative size of raw temp data
---------+---------------+---------------+-----------+----------------+------------+------
-------------+---------------+------------------+---------
------+----------------+----------------+----------------------------------
-1 | NewEENode | 1 | 1 | | | 907 | | 55619 |
55619 | | |
-1 | Root | 1 | 1 | | | 34964 | | |
55619 | | |
2 | GroupByHash | 4 | 1 | | | 105288 | | 56086 |
56086 | | | 0
2 | GroupByPipe | 2 | 1 | | | 2356 | | 404250 |
404250 | | |
```

```
2 | NetworkRecv | 2 | 1 | 1659961 | | 4353 | 93335 | 93335 |
93335 | | |
2 | NetworkSend | 2 | 1 | | 673086 | 10780 | | 31144 |
31144 | 2539735 | 67437 |
2 | ParallelMerge | 1 | 1 | | | 239 | | 18621 |
18621 | | |
2 | ParallelUnion | 3 | 1 | | | 4138 | | 75181 |
75181 | | |
2 | StorageUnion | 2 | 1 | | | 40551 | | 404250 |
404250 | | |
3 | ExprEval | 2 | 1 | | | 10858 | | 404250 |
404250 | | |
3 | Join | 2 | 1 | | | 37540 | | 404250 |
404250 | | | 0
4 | Join | 2 | 1 | | | 225466 | | 404250 |
404250 | | | 0
5 | Scan | 2 | 1 | | | 544160 | | 404250 |
404250 | | |
6 | NetworkRecv | 1 | 1 | 1073992 | | 635 | 60000 | 60000 |
60000 | | |
6 | NetworkSend | 1 | 1 | | 1075688 | 2887 | | 20045 |
20045 | 34962 | 0 |
6 | Scan | 1 | 1 | | | 2550 | | 20045 |
20045 | | |
6 | StorageUnion | 1 | 1 | | | 428 | | 20045 |
20045 | | |
8 | GroupByHash | 1 | 1 | | | 21 | | 4 |
4 | | | 0
8 | GroupByPipe | 1 | 1 | | | 612 | | 8 |
8 | | |
8 | NetworkRecv | 1 | 1 | 100 | | 417 | 10 | 10 |
10 | | |
8 | NetworkSend | 1 | 1 | | 102 | 11 | | 4 |
4 | 15432 | 0 |
8 | Scan | 1 | 1 | | | 354 | | 4 |
8 | | |
8 | StorageUnion | 1 | 1 | | | 165 | | 8 |
8 | | |
(23 rows)
```

データをピボット形式にすることで、オペレータ間のカウンタ値の遷移を簡単に確認し、クエリの実行中に発生する可能性のあるデータフローの問題を特定できます。クエリはボトムアップから実行されますが、一部のパスは並行して実行されることがあります。
たとえば、前のクエリでは、パス3の結合は、実行可能な前のパスが完了するのを待つ必要があり、パス4とパス8は並行して実行できると結論づけることができるでしょう。

結果分析

これらのサンプルクエリを使用すると、クエリがオペレータをどのようにパスするかを知ることができ、潜在的なボトルネックとなるノードのカウンタを特定できます。
クエリのグラフは、Vertica Knowledge BaseのReading Query Plansで説明されているように、オペレータが並行してまたは連続してどのように動作するかを確認するのに役立ちます。クエリグラフの例については、次のSimplified Joinグラフを参照してください。簡略化されたプランにはパス名はありませんが、このグラフは、どのオペレータが並行して実行され、どのオペレータが前のオペレータのデータを必要とするかを明確に示しています。



SIPs のパフォーマンスのレビュー

SIPs（Sidewise Information Passing）が有効になっている場合（デフォルト）、Verticaは結合オペレータを経由する前に結合の外部テーブルからタプルをフィルタリングします。フィルタリングされるレコードは、結合の内部結合テーブルの述語に基づいています。これらの述語は結合キーの一部ではない列にあります。
Verticaは結合の条件を満たす列のレコードのみをマテリアライズするため、結合前にタプルをフィルタリングするとパフォーマンスが向上します。
Vertica 7.2.xには、SIPs機能に3つの拡張機能が含まれています。

- Verticaは、SIPs最適化をマージ結合とハッシュ結合の両方に適用します。
- EXECUTION_ENGINE_PROFILESテーブルにcounter_tag列が追加されました。counter_tagは、異なるインスタンスを区別する必要のあるオペレータのカウンタを一意に識別する文字列です。
- 新しいSIPs関連のカウンタが追加されました。
    - o SIPsProcessedRows: SIPs式によって処理された行数
    - o SIPSPrunedRows: SIPs式によってフィルタリングされた行数

SIPsオペレータのカウンタを見ると、外部結合の余分なフィルタによってクエリが処理する必要があるタプルの数が減ることがわかります。場合によっては、減らされたタプルの数が余りにも少なく、余分なフィルタを正当化できません。そのような場合、その特定のクエリのSIPsを無効にして、パフォーマンスを向上させます。
特定のクエリのSIPsを無効にするには、クエリに次のヒントを追加します。

```
/*+add_vertica_options(BASIC,DISABLE_SIPS) */
```

DC_SIPS_STATISTICS システムテーブル

SIPsの情報の分析を完了するために、Verticaは新しいデータコレクタテーブルを7.2.xでDC_SIPS_STATISTICSとして追加しました。このテーブルでは、SIPsがプルーニングを処理した行数を確認できます。プルーニングされた行数によっては、SIPsが行をプルーニングするのにかかる時間を相殺しない場合があります。この場合、パフォーマンスの向上はそれほど重要ではなく、SIPsを無効にする必要があります。

クエリ例

```
=> SELECT node_name, time, sip_expr_id, sip_entries, rows_processed, rows_pruned,
blocks_processed,
blocks_pruned, blocks_pruned_BY_valuelist FROM dc_sips_statistics WHERE transaction_id =
:t_id and
statement_id = :s_id and node_name = (select LOCAL_NODE_NAME());
-[ RECORD 1 ]--------------+------------------------------
node_name | v_vmartdb_node0001
time | 2016-03-08 21:01:56.275266+00
sip_expr_id | 2
sip_entries | 20154
rows_processed | 20480
rows_pruned | 0
blocks_processed | 10159
blocks_pruned | 0
blocks_pruned_by_valuelist | 0
-[ RECORD 2 ]--------------+------------------------------
node_name | v_vmartdb_node0001
```

```
time | 2016-03-08 21:01:56.275297+00
sip_expr_id | 3
sip_entries | 5
rows_processed | 20480
rows_pruned | 0
blocks_processed | 10159
blocks_pruned | 0
blocks_pruned_by_valuelist | 0
-[ RECORD 3 ]--------------+-------------------------------
node_name | v_vmartdb_node0001
time | 2016-03-08 21:01:56.275355+00
sip_expr_id | 2
sip_entries | 4
rows_processed | 60000
rows_pruned | 20480
blocks_processed | 0
blocks_pruned | 10159
blocks_pruned_by_valuelist | 0
…
```

結果分析

SIPsが多数の行をプルーニングした場合、SIPsに関連するオーバーヘッドには価値があります。SIPsが行を処理しないまたは少数の行を処理する場合、SIPsプロセスはパフォーマンス上の利点を提供しません。

クエリで使用されるプロジェクションの理解

クエリのパフォーマンスを評価する場合、クエリがアクセスしているプロジェクションを確認することが重要です。プロジェクションについての確認項目：

- リフレッシュ済みかどうか
- 統計情報を保持しているかどうか
- セグメンテーションされているかどうか。されている場合、分散キーはどうか。
- プリプロジェクションかどうか
- スーパープロジェクションかどうか
- データを均等に分散し、クエリ中のセグメンテーションを減らすフィールドで分散かされているかどうか
- ライブアグリゲートプロジェクションかどうか

PROJECTION_USAGE システムテーブル

PROJECTIONS_USAGEテーブルには、実行されたクエリごとにVerticaが使用したプロジェクションに関する情報が含まれています。

クエリ例

次のクエリは、特定のクエリに使用されたプロジェクションを返します。通常、Verticaは同じオフセット（b0、b1など）のプロジェクションを使用します。ノードが停止している場合、異なるオフセットのプロジェクションが混在することがあります。

```
=> SELECT node_name, projection_name, anchor_table_name FROM projection_usage WHERE
transaction_id
= :t_id AND statement_id = :s_id;
node_name | projection_name | anchor_table_name
-------------------+----------------------+------------------
v_vmartdb_node0001 | product_dimension_b0 | product_dimension
v_vmartdb_node0001 | store_dimension_b0 | store_dimension
v_vmartdb_node0001 | store_sales_fact_b0 | store_sales_fact
(3 rows)
```

前述のクエリ結果を使用して、テーブルとプロジェクション情報を変数に保存できます。

```
Note: The query examples in this topic use the following variables:
\set t_proj '''store_sales_fact_b0'''
\set t_name '''store_sales_fact'''
\set t_schema '''store'''
```

結果分析

通常、Verticaクエリは同じオフセットのプロジェクションを使用します（product_dimension_b0など）。クエリがproduct_dimension_b0やproduct_dimension_b1などの異なるオフセットのプロジェクションにアクセスすると、クエリのパフォーマンスに影響する可能性があります。

PROJECTIONS システムテーブル

PROJECTIONSシステムテーブルには、プロジェクションに関する重要な情報が含まれています。
PROJECTIONSシステムテーブルのcreated_epoch列は、どのエポックでプロジェクションが作成されたかを示します。

クエリ例

テーブルのスキーマと名前を使用して、そのテーブルのすべてのプロジェクションに関する情報を取得します。

```
=> \x
=> SELECT projection_schema, projection_name, is_prejoin, is_up_to_date, has_statistics,
is_
segmented, segment_expression,is_super_projection, created_epoch FROM projections where
anchor_
table_name = :t_name and projection_schema = :t_schema;
-[ RECORD 1 ]-------+-----------------------------------------------------------------------
---------
-------
projection_schema | store
projection_name | store_sales_fact_b0
is_prejoin | f
is_up_to_date | t
```

```
has_statistics | t
is_segmented | t
segment_expression | hash(store_sales_fact.date_key, store_sales_fact.product_key,
store_sales_
fact.product_version, store_sales_fact.store_key, store_sales_fact.promotion_key,
store_sales_
fact.customer_key, store_sales_fact.employee_key, store_sales_fact.pos_transaction_number,
store_
sales_fact.sales_quantity, store_sales_fact.sales_dollar_amount,
store_sales_fact.cost_dollar_
amount, store_sales_fact.gross_profit_dollar_amount, store_sales_fact.transaction_time,
store_
sales_fact.tender_type, store_sales_fact.transaction_type)
is_super_projection | t
created_epoch | 12
-[ RECORD 2 ]-------+-----------------------------------------------------------
projection_schema | store
projection_name | store_sales_fact_b1
is_prejoin | f
is_up_to_date | t
has_statistics | t
is_segmented | t
segment_expression | hash(store_sales_fact.date_key, store_sales_fact.product_key,
store_sales_
fact.product_version, store_sales_fact.store_key, store_sales_fact.promotion_key,
store_sales_
fact.customer_key, store_sales_fact.employee_key, store_sales_fact.pos_transaction_number,
store_
sales_fact.sales_quantity, store_sales_fact.sales_dollar_amount,
store_sales_fact.cost_dollar_
amount, store_sales_fact.gross_profit_dollar_amount, store_sales_fact.transaction_time,
store_
sales_fact.tender_type, store_sales_fact.transaction_type)
is_super_projection | t
created_epoch | 12
```

結果分析

PROJECTIONSシステムテーブルでプロジェクション情報を確認する際は、次の点を考慮してください。

- デフォルトでは、COPYおよびINSERT INTOステートメントから自動プロジェクションを作成すると、Verticaは自動的に最初の32列でデータを分散します。この作業は、データスキューを防ぐのに役立ちます。
  ただし、32列で分散化すると、ハッシュアルゴリズムが複雑になり、特に複数のセグメント化された列がVARCHAR（1000）である場合、CPUインテンシブになる可能性があります。これらのプロジェクシ

ョンでは、デフォルトのセグメンテーションを確認して、プロジェクションがより良いセグメンテーションになるかどうかを確認してください。
- テーブルに統計がある場合、オプティマイザは、クエリに最適なプロジェクションにアクセスすること選択し、低コストのクエリプランを作成します。最適なプランを立てるには、すべてのテーブルの統計を作成または更新します。

PROJECTION_COLUMNS システムテーブル

プロジェクションのORDER BY句をチェックするには、PROJECTION_COLUMNSシステムテーブルを照会します。どの列がORDER BY句の一部で、どの位置にあるのかを確認できます。さらに、ORDER BY列に統計情報がある場合、統計の種類、列に入っているデータの種類、および使用されているエンコーディングの種類（存在する場合）を確認することができます。

クエリ例

次のクエリでは、ORDER BY句の一部であるテーブルの列のみが表示されます。すべての列を表示する場合は、述部でsort_position> = 0を削除します。

```
=> SELECT projection_name, projection_column_name, column_position, sort_position,
encoding_type, access_rank, statistics_type, statistics_updated_
timestamp FROM projection_columns WHERE sort_position >= 0 AND projection_name = :t_proj
AND table_schema = :t_schema ORDER BY projection_name, sort_
position;
projection_name | projection_column_name | column_position | sort_position | encoding_type
| access_rank | statistics_type | statistics_
updated_timestamp
--------------------+---------------------------+-----------------+---------------+-----
----------+-------------+-----------------+----------------
---------------
store_sales_fact_b0 | employee_key | 6 | 0 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | customer_key | 5 | 1 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | promotion_key | 4 | 2 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | store_key | 3 | 3 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | product_key | 1 | 4 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | product_version | 2 | 5 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | date_key | 0 | 6 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | pos_transaction_number | 7 | 7 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | sales_quantity | 8 | 8 | AUTO | 0 | ROWCOUNT | 2016-02-26
```

```
14:44:41.275664+00
store_sales_fact_b0 | sales_dollar_amount | 9 | 9 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | cost_dollar_amount | 10 | 10 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | gross_profit_dollar_amount | 11 | 11 | AUTO | 0 | ROWCOUNT | 2016-
02-26
14:44:41.275664+00
store_sales_fact_b0 | transaction_type | 12 | 12 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | transaction_time | 13 | 13 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
store_sales_fact_b0 | tender_type | 14 | 14 | AUTO | 0 | ROWCOUNT | 2016-02-26
14:44:41.275664+00
(15 rows)
```

結果分析

エンコーディングは、ディスクに格納されたデータのフットプリントを減らすため、クエリの実行中にディスクから読み取るバイト数が少なくなります。Verticaは、Vertica documentationのEncoding-Typeで説明されているいくつかのエンコーディングアルゴリズムを使用します。

データベースデザイナー（DBD）は、データの1%サンプルを使用して、各プロジェクションの列の最適なエンコーディングを定義します。最高のエンコーディングは、フィールドタイプとカーディナリティに基づいています。エンコーディングが定義されていない場合、VerticaはプロジェクションのDDLでAUTOを指定します。この設定は、カーディナリティを考慮せずに、Verticaがデータ型で最適なエンコーディングを使用するように指定します。

ただし、DBDはストレージを最小限に抑えようとしますが、パフォーマンスに影響する可能性があります。DBDを実行する前に、最適化の目的を特定してください。

次の表に、Verticaのデータ型ごとの最適なエンコードとデフォルトのエンコードを示します。

Data Type + ENCODING hint	INTEGER, NUMERIC(<=18), DATE, TIMESTAMP, etc.	NUMERIC(19+)	FLOAT	CHAR/VARCHAR	BOOLEAN	注釈
<default> aka AUTO	Delta Int Pack	LZO	LZO	String+LZO	LZO	以下に該当するものがない場合に使用します。
NONE	Delta Int Pack	LZO	LZO	LZO	LZO	使用しないでください。
RLE	RLE + LZO	RLE + LZO	RLE + LZO	String RLE + LZO	RLE + LZO	列がソートされ、レコードの繰り返し数が平均10を超える場合に使用します。
BLOCK_DICT	Block Dict	Block Dict	Block Dict	Block Dict	Block Dict	ブロックごとに異なる値がほとんど

						ない場合に使用します。
BLOCKDICT_COMP	Block Dict Comp	Block Dict Comp	Block Dict Comp	Block Dict Comp	Block Dict Comp	ブロックごとに異なる値が少なく、スキューが高い場合に使用します。
DELTAVAL	Block Delta Val	LZO	LZO	LZO	LZO	整数が狭い範囲にある場合に使用します。
GCDDELTA	Block GCD Delta Val	LZO	LZO	LZO	LZO	タイムスタンプや整数が共通の要素の倍数の場合に使用します。
COMMONDELTA_COMP	Common Delta	LZO	Common Delta	LZO	LZO	ブロックあたりのデルタの数がブロック内の値の範囲より小さく、ブロック内の個別の値の数よりも少ない場合に使用します。
DELTARANGE_COMP	Delta Range	LZO	Delta Range	LZO	LZO	浮動小数点またはブロックごとに異なる値が多い整数データに使用します。
DELTARANGE_COMP_SP	Delta Range	LZO	Delta Range	LZO	LZO	単精度浮動小数点データで使用します。

列をエンコードしてマテリアライズするには、VerticaはCPUサイクルを使用するエンコーディングアルゴリズムを適用する必要があります。したがって、クエリまたはロードがCPUにバインドされている場合、エンコーディングを削除するとパフォーマンスが向上する可能性があります。
クエリがCPUバウンドかI/Oバウンドかを判断することが重要です。クエリがCPUにバインドされている場合、不適切なエンコーディングを使用している可能性があります。

PROJECTIONS_STORAGE システムテーブル

PROJECTIONS_STORAGEシステムテーブルを照会して、各ノードのプロジェクションがどれだけ大きいかを確認します。

```
=> SELECT node_name, projection_schema, projection_name, SUM(row_count) row_count,
ROUND(SUM(used_bytes)/1024^2::NUMERIC(10,3),3) used_GB, COUNT
(DISTINCT node_name) num_nodes, SUM(ros_count) ros_count FROM projection_storage WHERE
node_name IN (SELECT node_name FROM nodes WHERE is_ephemeral =
'f' ) AND projection_name = :t_proj AND projection_schema = :t_schema GROUP BY node_name,
projection_schema, projection_name ORDER BY 2,3,1;
```

```
node_name | projection_schema | projection_name | row_count | used_GB | num_nodes |
ros_count
-------------------+------------------+--------------------+-----------+---------+-----
------+-----------
v_test_db_node0001 | store | store_sales_fact_b0 | 1250032 | 29.586 | 1 | 1
v_test_db_node0002 | store | store_sales_fact_b0 | 1250571 | 29.583 | 1 | 1
v_test_db_node0003 | store | store_sales_fact_b0 | 1250344 | 29.594 | 1 | 1
v_test_db_node0004 | store | store_sales_fact_b0 | 1249053 | 29.565 | 1 | 1
(4 rows)
```

結果分析

PROJECTION_STORAGEテーブルのrow_count列を見ると、あるノードに他のノードより多くのデータがあることがわかります。これは、データが偏っていることを意味します。プロジェクションのセグメンテーションを確認してください。

1つのノードのros_countがはるかに高い場合、そのノードでTuple Moverが正しく動作していない可能性があります。

Verticaはクラスター内で最も遅いノードに速度がひきずられます。1つのノードに大量のデータなどの問題がある場合、PROJECTION_STORAGEテーブルはその情報を提供します。

テーブルパーティションのレビュー

パーティショニングはテーブルのプロパティです。テーブルがパーティションされている場合、そのテーブルのすべてのプロジェクションは同じ式を使用してパーティションされます。パーティションの定義は論理データベース設計の一部であり、ユーザーが定義する必要があります。Verticaデータベースデザイナー（DBD）ではパーティション式は指定されません。パーティションは、クエリの実行中に並列性を向上させることができます。

Verticaは、ファクトテーブルを次の理由でパーティション化することを推奨しています。

- パーティションキーに述語がある場合、Verticaは述語と一致するROSコンテナを簡単に見つけることができるため、スキャン時間が改善されます。パーティション化されたテーブルのROSコンテナが必要ない場合、Verticaはクエリの実行中にコンテナを処理しないようにすることができます。このプロセスはパーティションプルーニングと呼ばれます。ROSコンテナを排除するために、Verticaはクエリ述部をパーティション関連のメタデータと比較します。
- 異なるパーティションのデータはディスク上の別々のファイルに保存され、並列実行が向上します。
- データをパーティションすると、データのロード中に削除されないようになります。
- ファクトテーブルの場合、古いデータを削除するにはパーティションを使用するのが最も効率的です。
- 分割されたファイルにデータが分散されるため、パーティション化によって並列性が向上する可能性があります。
- パーティション化は、システムからデータを削除するときのパフォーマンスを助けることもできます。

TABLES システムテーブル

TABLESシステムテーブルには、テーブルがパーティション化されているかどうか、もしそうであればどのようにパーティション化されているかどうかなど、データベース内のすべてのテーブルに関する情報が含まれています。

クエリ例

テーブルがパーティション化されているかどうかを確認し、パーティション式を識別するには、TABLESシステムテーブルを照会します。パーティション式は、パーティションのソート方法を定義します。

```
=> SELECT table_schema, table_name, owner_name, partition_expression, create_time FROM
tables WHERE table_name = :t_name and table_schema = :t_
schema;
table_schema | table_name | owner_name | partition_expression | create_time
--------------+-----------------+------------+------------------------------+------------
--------------------------------
store | store_sales_fact | dbadmin | store_sales_fact.store_key | 2016-08-02
17:48:43.60541+00
(1 row)
```

結果分析

テーブルが大きいあるいはファクトテーブルの場合、Verticaがクエリパフォーマンスのためにディスク上のデータを整理できるようにパーティション化します Verticaは必要なパーティションだけを読み込みます。パーティショニングの詳細については、以下を参照ください。

- Vertica documentationのUsing Table Partitions
- Vertica Knowledge BaseのVertica Partitions: The FAQs

リソース割り当てのレビュー

実行される各クエリは、システムリソースを使用します。同時に実行される複数のクエリは、システムリソースを求めて競合する可能性があります。適切なクエリのパフォーマンスを得るには、データベースワークロードに適したリソースを割り当ててください。

RESOURCE_ACQUISITIONS システムテーブル

RESOURCE_ACQUISITIONSシステムテーブルには、特定のクエリによって取得されたリソースが含まれています。

クエリ例

次のクエリは、特定のtransaction_idおよびstatement_idのリソース要求に関する情報を返します。

```
=> SELECT node_name, request_type,pool_name, memory_kb, filehandles,
threads,succeeded,result, failing_resource FROM resource_acquisitions WHERE
transaction_id = :t_id and statement_id = :s_id order by node_name;
node_name | request_type | pool_name | memory_kb | filehandles | threads | succeeded |
result | failing_resource
-------------------+--------------+-----------+-----------+-------------+---------+-------
----+---------+------------------
v_vmartdb_node0001 | Acquire | general | 102400 | 0 | 0 | t | Granted |
v_vmartdb_node0001 | Reserve | general | 1548488 | 54 | 56 | t | Granted |
v_vmartdb_node0002 | Reserve | general | 1548488 | 51 | 53 | t | Granted
|v_vmartdb_node0003 | Reserve | general
| 1548488 | 51 | 53 | t | Granted |
```

```
(4 rows)
```

取得要求（特定のリソースを取得する要求）は、イニシエーターノードでのみ発生します。この要求は、オプティマイザがクエリを計画するために使用するリソースを示します。デフォルト値は100 MBです。
ただし、MEMORY_LIMIT_HITイベントがQUERY_EVENTSシステムテーブルに表示されている場合、100 MBではクエリを計画するには不十分であることが分かります。クエリが複雑すぎるか、100 MBを使い切る前に最適なプランを作成するために、クエリのテーブルにプロジェクションが多すぎる可能性があります。
この問題を解決するには、クエリを簡略化し、未使用の不要なプロジェクションを削除するか、オプティマイザがクエリを計画するために使用できるメモリを増やします。使用可能なメモリを増やすには、MaxOptMemMB構成パラメーターを変更します。

```
=> ALTER DATABASE dbname SET MaxOptMemMB = 150
```

実行ノードでは、ReserveRequest はクエリを実行するためにリソースを予約するように求めます。予約メモリがクエリを実行するのに十分でない場合、AcquireAdditional 要求タイプが表示されます。オペレータのニーズに応じて、AcquireAdditional 要求が成功する場合と成功しない場合があります。後続の列は、Vertica が追加のリソースを許可したかどうかを示します。要求が成功しなかった場合、結果列には次の例のように理由が示されます。

```
v_vmartdb_node0001 | AcquireAdditional | t | | Granted
| 151.03
v_vmartdb_node0001 | 45035996273723096 | 1 | AcquireAdditional | t | |
Granted
| 228.03
v_vmartdb_node0001 | 45035996273723096 | 1 | AcquireAdditional | f |
Memory(KB) | Request exceeds limits: Memory(KB)
Exceeded: Requested = 161480705, Free = 7744915 (Limit = 246861296, Used =
239116381) | 382.03
2015-10-26 17:16:35 | v_vmartdb_node0001 | 45035996273723096 | 1 |
AcquireAdditional | f Request exceeds limits: Memory(KB)
Exceeded: Requested = 161480705, Free = 7744915 (Limit = 246861296, Used =
239116381) | 382.03
```

RequestAdditional 要求は、ハッシュ結合または Group by Hash で最も頻繁に発生します。

結果分析

クエリがVerticaが該当のクエリに対して予約したリソースよりも多くのリソースを要求した場合は、さらに分析を実行して、オプティマイザがクエリを適切に計画しなかった理由を調べます。この問題は、テーブルの統計情報に問題がある可能性があります。統計が10%のサンプルに基づいている場合、ANALYZE_HISTOGRAMを実行して統計サンプルを増やし、統計の精度を向上させます。

本ドキュメントの推奨事項

各クエリのパフォーマンスのユースケースは異なるため、本ドキュメントの内容は参考情報として使用してください。Verticaは、本書で提案されているように変数を使用し、変数をここに記載されたステートメントにカットアンドペーストすることを推奨します。

If you have other useful queries that help you with performance tuning, we'd love to hear about them. Add them to the Comments section of the Big Data and Analytics Knowledge Base so that the Vertica user community can add to their performance-tuning expertise.

パフォーマンスのチューニングに役立つその他の便利なクエリがある場合、その内容についておきかせいただけると幸いです。Big Data and Analytics Knowledge Baseのコメントセクションに追加して、Vertica user communityがパフォーマンスチューニングの専門知識に追加できるようにします。

詳細情報

詳細情報	...See
Vertica Community Edition	https://my.vertica.com/community/
Vertica Documentation	http://my.vertica.com/docs/latest/HTML/index.htm
Big Data and Analytics Community	https://forum.vertica.com/