
Pivotal
A NEW PLATFORM FOR A NEW ERA

Query Tuning & Rewriting

Pivotal® Greenplum Database

Agenda

- Introduction
- Ways to view plans
- Features in plans: operators, rows, cost
- Mitigating sub-optimal query plans
- Work the lab

Pivotal

Query Tuning & Rewriting

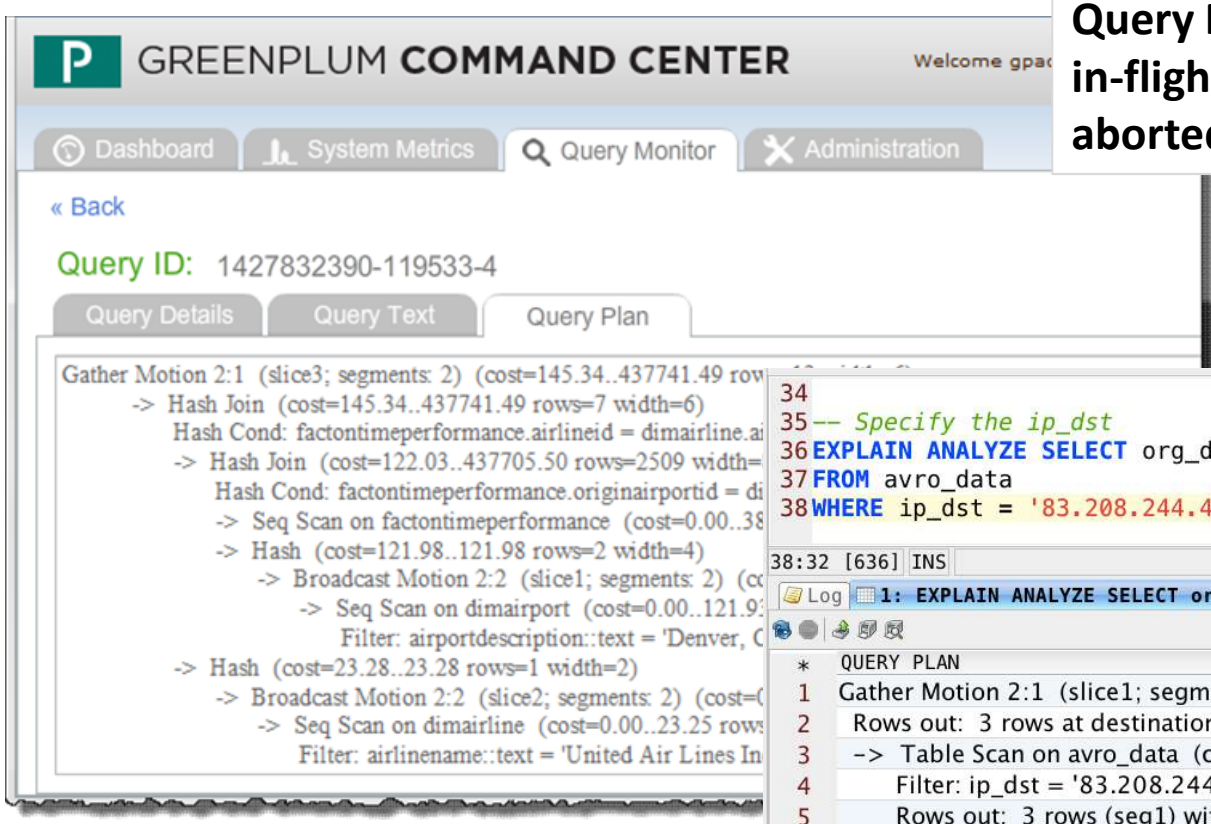
```
Timing is on.
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Gather Motion 128:1  (slice9; segments: 128)  (cost=0.00..5808.96 rows=100 width=17)
   Merge Key: customer.c_customer_id
   Rows out:  100 rows at destination with 21478 ms to end, start offset by 3605 ms.
   ->  Sort  (cost=0.00..5808.95 rows=1 width=17)
         Sort Key: customer.c_customer_id
         Rows out:  Avg 1.3 rows x 79 workers.  Max 2 rows (seg1) with 21469 ms to end, start offset by 3612 ms.
         Executor memory:  58K bytes avg, 58K bytes max (seg0).
         Work_mem used:  58K bytes avg, 58K bytes max (seg0). Workfile: (0 spilling, 0 reused)
         ->  Sequence  (cost=0.00..5808.95 rows=1 width=17)
               Rows out:  Avg 1.3 rows x 79 workers.  Max 2 rows (seg1) with 21468 ms to end, start offset by 3613 ms.
               ->  Shared Scan (share slice:id 9:0)  (cost=0.00..3519.39 rows=1067214 width=1)
                     Rows out:  Avg 1267950.0 rows x 128 workers.  Max 1270874 rows (seg79) with 11594 ms to first row, 11815 ms to end, start offset by 3606 ms.
                     ->  Materialize  (cost=0.00..3519.39 rows=1067214 width=1)
                           Rows out:  0 rows (seg0) with 13949 ms to end, start offset by 3605 ms.
                           Work_mem used:  6368K bytes avg, 6368K bytes max (seg0). Workfile: (0 spilling, 0 reused)
                           Work_mem wanted: 34637K bytes avg, 34688K bytes max (seg1) to lessen workfile I/O affecting 128 workers.
                           ->  HashAggregate  (cost=0.00..3518.32 rows=1067214 width=16)
                                 Group By: store_returns.sr_customer_sk, store_returns.sr_store_sk
                                 Rows out:  Avg 1267950.0 rows x 128 workers.  Max 1270874 rows (seg79) with 10568 ms to first row, 11306 ms to end, start offset by 3606 ms.
                                 Executor memory:  20689K bytes avg, 20689K bytes max (seg0).
                                 Work_mem used:  18158K bytes avg, 18158K bytes max (seg1). Workfile: (128 spilling, 0 reused)
                                 Work_mem wanted: 90868K bytes avg, 91325K bytes max (seg57) to lessen workfile I/O affecting 128 workers.
                                 (seg57)  1269544 groups total in 32 batches; 1 overflows; 1278580 spill groups.
                                 (seg57)  Hash chain length 1.5 avg, 14 max, using 1015454 of 2162688 buckets.
                                 ->  Redistribute Motion 128:128  (slice8; segments: 128)  (cost=0.00..3248.79 rows=1067214 width=16)
                                       Hash Key: store_returns.sr_customer_sk, store_returns.sr_store_sk
                                       Rows out:  Avg 1282816.8 rows x 128 workers at destination.  Max 1286257 rows (seg79) with 577 ms to first row, 9089 ms to end, start offset by 3607 ms.
                                       ->  Result  (cost=0.00..3195.34 rows=1067214 width=16)
                                             Rows out:  Avg 1282816.8 rows x 128 workers.  Max 1285917 rows (seg125) with 645 ms to first row, 3061 ms to end, start offset by 3683 ms.
                                             ->  HashAggregate  (cost=0.00..3195.34 rows=1067214 width=16)
                                                   Group By: store_returns.sr_customer_sk, store_returns.sr_store_sk
                                                   Rows out:  Avg 1282816.8 rows x 128 workers.  Max 1285917 rows (seg125) with 645 ms to first row, 2798 ms to end, start offset by 3683 ms.
                                                   Executor memory:  20689K bytes avg, 20689K bytes max (seg0).
                                                   ->  Hash Join  (cost=0.00..2925.80 rows=1067214 width=16)
                                                         Hash Cond: store_returns.sr_returned_date_sk = date_dim.d_date_sk
                                                         Rows out:  Avg 1302840.3 rows x 128 workers.  Max 1306167 rows (seg105) with 45 ms to first row, 1771 ms to end, start offset by 3678 ms.
                                                         Executor memory:  12K bytes avg, 12K bytes max (seg0).
                                                         Work_mem used:  12K bytes avg, 12K bytes max (seg0). Workfile: (0 spilling, 0 reused)
                                                         (seg105) Hash chain length 1.0 avg, 1 max, using 365 of 65539 buckets.
                                                         ->  Dynamic Table Scan on store_returns (dynamic scan id: 1)  (cost=0.00..928.47 rows=6749920 width=20)
                                                               Rows out:  Avg 1919046.7 rows x 128 workers.  Max 1922514 rows (seg105) with 23 ms to first row, 975 ms to end, start offset by 3700 ms.
                                                               Partitions scanned:  Avg 5.0 (out of 23) x 128 workers.  Max 5 parts (seg0).
                                                         ->  Hash  (cost=100.00..100.00 rows=1 width=4)
                                                               Rows in:  Avg 365.0 rows x 128 workers.  Max 365 rows (seg0) with 18 ms to end, start offset by 3683 ms.
                                                               ->  Partition Selector for store_returns (dynamic scan id: 1)  (cost=10.00..100.00 rows=1 width=4)
                                                                     Filter: store_returns.sr_returned_date_sk = date_dim.d_date_sk
                                                                     Rows out:  Avg 365.0 rows x 128 workers.  Max 365 rows (seg0) with 0.581 ms to first row, 18 ms to end, start offset by 3683 ms.
                                                                     ->  Broadcast Motion 128:128  (slice7; segments: 128)  (cost=0.00..431.06 rows=315 width=4)
                                                                           Rows out:  Avg 365.0 rows x 128 workers at destination.  Max 365 rows (seg0) with 0.035 ms to first row, 17 ms to end, start offset by 3683 ms.
                                                                           ->  Table Scan on date_dim  (cost=0.00..431.06 rows=3 width=4)
                                                                                 Filter: d_year = 1999
                                                                                 Rows out:  Avg 2.9 rows x 128 workers.  Max 4 rows (seg5) with 1.053 ms to first row, 1.114 ms to end, start offset by 3656 ms.
               ->  Redistribute Motion 1:128  (slice6)  (cost=0.00..2289.56 rows=100 width=17)
                     Rows out:  Avg 1.3 rows x 79 workers at destination.  Max 2 rows (seg1) with 9559 ms to end, start offset by 15522 ms.
                     ->  Limit  (cost=0.00..2289.56 rows=1 width=17)
                           Rows out:  100 rows with 21449 ms to first row, 21450 ms to end, start offset by 3629 ms.
                           ->  Gather Motion 128:1  (slice5; segments: 128)  (cost=0.00..2289.56 rows=100 width=17)
                                 Merge Key: customer.c_customer_id
                                 Rows out:  100 rows at destination with 21449 ms to first row, 21450 ms to end, start offset by 3629 ms.
                                 ->  Limit  (cost=0.00..2289.56 rows=1 width=17)
                                       Rows out:  Avg 100.0 rows x 128 workers.  Max 100 rows (seg0) with 20661 ms to end, start offset by 3618 ms.
                                       ->  Sort  (cost=0.00..2289.56 rows=39411 width=17)
                                             Sort Key: customer.c_customer_id
                                             Rows out:  Avg 100.0 rows x 128 workers.  Max 100 rows (seg0) with 20661 ms to end, start offset by 3618 ms.
                                             Executor memory:  2233K bytes avg, 2233K bytes max (seg0).
                                             Work_mem used:  2233K bytes avg, 2233K bytes max (seg0). Workfile: (0 spilling, 0 reused)
                                             ->  Hash Join  (cost=0.00..2231.56 rows=39411 width=17)
                                                   Hash Cond: customer.c_customer_sk = "inner".sr_customer_sk
                                                   Rows out:  Avg 5742.8 rows x 128 workers.  Max 5931 rows (seg111) with 20560 ms to first row, 20651 ms to end, start offset by 3623 ms.
                                                   Executor memory:  135K bytes avg, 140K bytes max (seg111).
                                                   Work_mem used:  135K bytes avg, 140K bytes max (seg111). Workfile: (0 spilling, 0 reused)
                                                   (seg111) Hash chain length 1.1 avg, 4 max, using 5438 of 65539 buckets.
                                                   ->  Table Scan on customer  (cost=0.00..457.17 rows=234375 width=21)
                                                         Rows out:  Avg 234375.0 rows x 128 workers.  Max 234376 rows (seg120) with 1.673 ms to first row, 54 ms to end, start offset by 24181 ms.
                                                   ->  Hash  (cost=1711.64..1711.64 rows=39411 width=4)
                                                         Rows in:  Avg 5742.8 rows x 128 workers.  Max 5931 rows (seg111) with 20558 ms to end, start offset by 3624 ms.
                                                         ->  Redistribute Motion 128:128  (slice4; segments: 128)  (cost=0.00..1711.64 rows=39411 width=4)
                                                               Hash Key: sr_customer_sk
```

Query Tuning & Rewriting

Pivotal

Explain Output Tools

Command Center
Query Plan output for in-flight, completed, or aborted queries



DbVisualizer **display of Explain plan**

```
34
35 -- Specify the ip_dst
36 EXPLAIN ANALYZE SELECT org_dst, gp_segment_id
37 FROM avro_data
38 WHERE ip_dst = '83.208.244.45';
```

38:32 [636] INS Auto Commit: ON

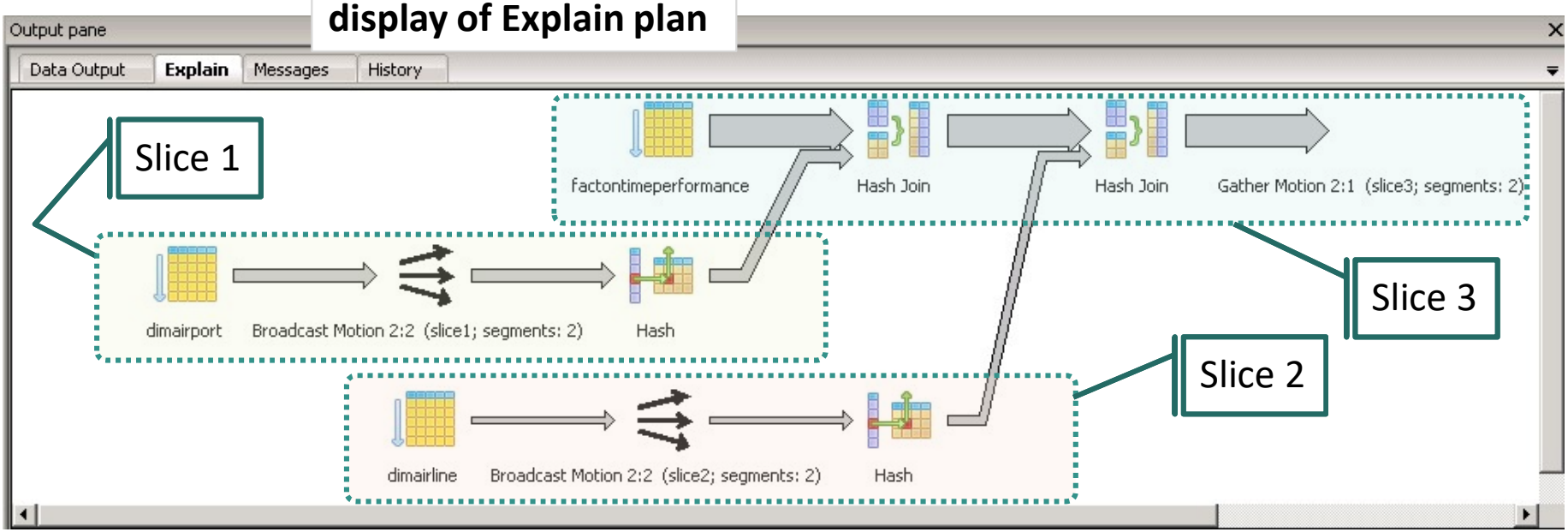
Log | 1: EXPLAIN ANALYZE SELECT org_dst... [13] ×

*	QUERY PLAN
1	Gather Motion 2:1 (slice1; segments: 2) (cost=0.00..431.07 rows=3 width=20)
2	Rows out: 3 rows at destination with 144 ms to end, start offset by 70 ms.
3	-> Table Scan on avro_data (cost=0.00..431.07 rows=2 width=20)
4	Filter: ip_dst = '83.208.244.45'::text
5	Rows out: 3 rows (seg1) with 53 ms to end, start offset by 160 ms.
6	Slice statistics:
7	(slice0) Executor memory: 159K bytes.
8	(slice1) Executor memory: 365K bytes avg x 2 workers, 365K bytes max (seg0).
9	Statement statistics:
10	Memory used: 131072K bytes
11	Settings: optimizer=on
12	Optimizer status: PQO version 1.597
13	Total runtime: 213.831 ms

Pivotal

Graphical Display of Explain Plans

pgAdmin III graphical display of Explain plan



EXPLAIN Example – Partition Elimination, Sorts, and Filters

Unique values are selected with the result grouped together

Example: Query executed on a partitioned table

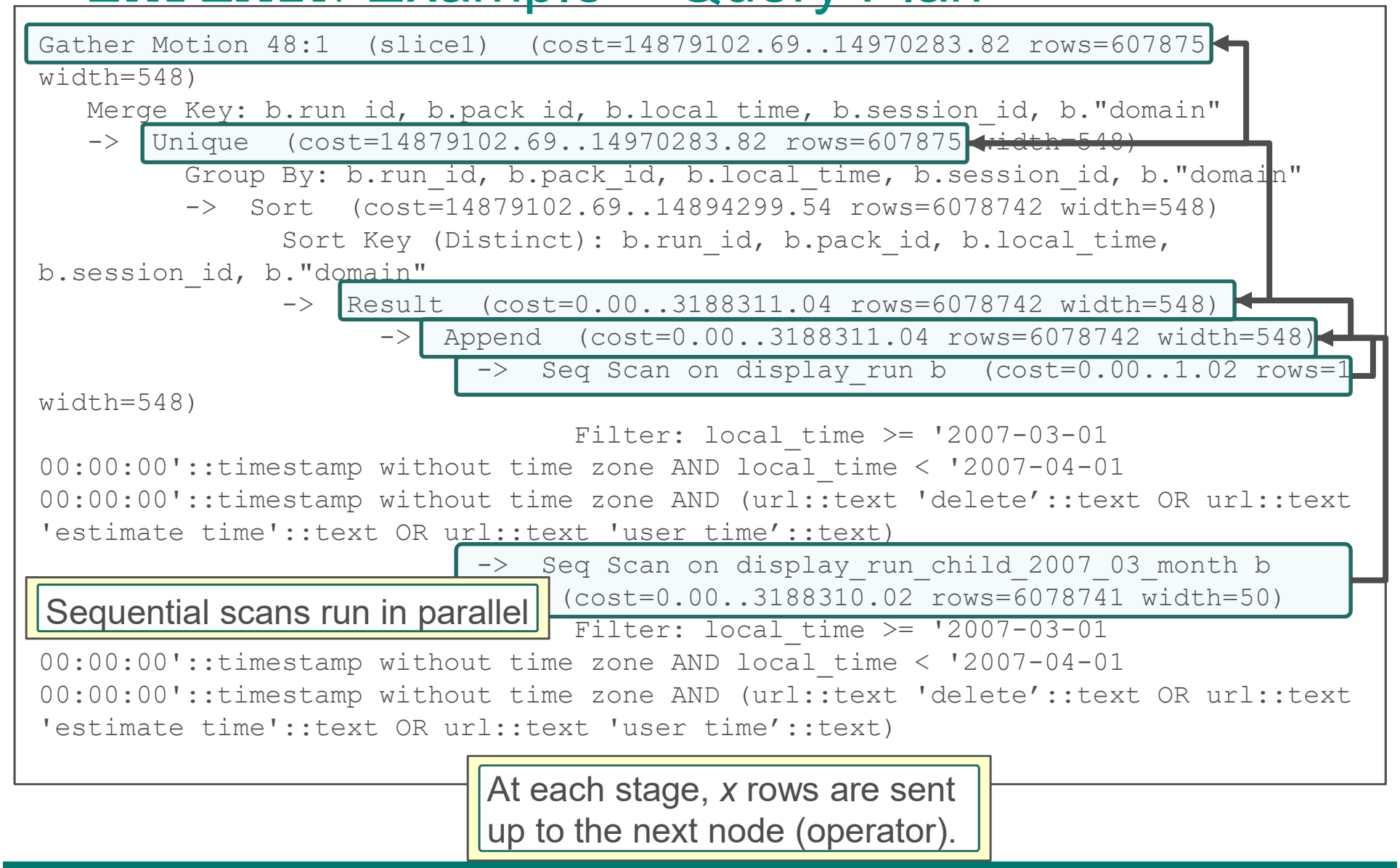
```
EXPLAIN SELECT DISTINCT (b.run_id), b.pack_id,
      b.local_time, b.session_id, b.domain
  FROM display_run b
  WHERE local_time >= '2007-03-01 00:00:00' AND
      local_time < '2007-04-01 00:00:00' AND (
        url='delete' OR url='estimate time' OR
        url='user time')
```

A sort is applied for the `DISTINCT` clause to guarantee a unique ordering of the rows

Filters are applied on columns on a partitioned table. Each partition will be scanned.

Pivotal

EXPLAIN Example – Query Plan



JOIN Order and Aggregation – Query

Example: JOIN Operation on multiple tables

```
SELECT COUNT(*) FROM partsupp ps, supplier, part
    WHERE ps.ps_suppkey = supplier.s_suppkey
    AND part.p_partkey = ps.ps_partkey;
```

`COUNT` applies an aggregate over all of the columns.

`JOIN` operation is applied over two tables

Begin by Examining Rows and Cost

```
Aggregate (cost=16141.02..16141.03 rows=1 width=0)
-> Gather Motion 2:1 (slice2) (cost=16140.97..16141.00 rows=1 width=0)
             -> Aggregate (cost=16140.97..16140.98 rows=1 width=0)
                    -> Hash Join (cost=2999.46..15647.43 rows=197414 width=0)
                        Hash Cond: ps.ps_partkey = part.p_partkey
                        -> Hash Join (cost=240.46..9920.71 rows=200020 width=4)
                            Hash Cond: ps.ps_suppkey = supplier.s_suppkey
                             -> Seq Scan on partsupp ps (cost=0.00..6180.00
rows=400000 width=8)
                             -> Hash (cost=177.97..177.97 rows=4999 width=4)
                                    -> Broadcast Motion 2:2 (slice1)
(cost=0.00..177.97 rows=4999
                                                          width=4)
                                       -> Seq Scan on supplier
(cost=0.00..77.99 rows=4999 width=4)
                        -> Hash (cost=1509.00..1509.00 rows=100000 width=4)
                             -> Seq Scan on part (cost=0.00..1509.00 rows=100000
width=4)
```

Note: Identify plan nodes where the estimated cost is very high and the number of rows is very large. This is where the majority of time is spent.

Pivotal

Validate Partition Elimination

```
Gather Motion 48:1  (slice1)  (cost=174933650.92..176041040.58 rows=7382598
width=548)
   Merge Key: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
   ->  Unique  (cost=174933650.92..176041040.58 rows=7382598 width=548)
         Group By: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
         ->  Sort  (cost=174933650.92..175118215.86 rows=73825977 width=548)
               Sort Key (Distinct): b.run_id, b.pack_id, b.local_time,
b.session_id, b."domain"
               ->  Result  (cost=0.00..31620003.26 rows=73825977 width=548)
                     ->  Append  (cost=0.00..31620003.26 rows=73825977 width=548)
                           ->  Seq Scan on display_run b  (cost=0.00..1.02 rows=1
width=548)
                                 Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
                           ->  Seq Scan on display_run_child_2007_03_month
b  (cost=0.00..2635000.02 rows=6079950 width=50)
                                 Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
                           ->  Seq Scan on display_run_child_2007_04_month
b  (cost=0.00..2635000.02 rows=6182099 width=50)
                                 Filter: url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text
                     …
```

All child partitions are scanned

Pivotal

Partition Elimination

```
Gather Motion 48:1  (slice1)  (cost=14879102.69..14970283.82 rows=607875
width=548)
   Merge Key: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
   ->  Unique  (cost=14879102.69..14970283.82 rows=607875 width=548)
         Group By: b.run_id, b.pack_id, b.local_time, b.session_id, b."domain"
         ->  Sort  (cost=14879102.69..14894299.54 rows=6078742 width=548)
               Sort Key (Distinct): b.run_id, b.pack_id, b.local_time,
b.session_id, b."domain"
               ->  Result  (cost=0.00..3188311.04 rows=6078742 width=548)
                     ->  Append  (cost=0.00..3188311.04 rows=6078742 width=548)
                           ->  Seq Scan on display_run b  (cost=0.00..1.02 rows=1
width=548)
                                 Filter: local_time >= '2007-03-01
00:00:00'::timestamp without time zone AND local_time < '2007-04-01
00:00:00'::timestamp without time zone AND (url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text)
                           ->  Seq Scan on display_run_child_2007_03_month b
                              (cost=0.00..3188310.02 rows=6078741 width=50)
                                 Filter: local_time >= '2007-03-01
00:00:00'::timestamp without time zone AND local_time < '2007-04-01
00:00:00'::timestamp without time zone AND (url::text 'delete'::text OR url::text
'estimate time'::text OR url::text 'user time'::text)
```

Partition is on `local_time`. `WHERE` clause is on `local_time`. Partition elimination is achieved.

Pivotal

Optimal Plan Heuristics

When analyzing query plans, try to design queries that select from the operators shown on the left:

Faster Operators	Slower Operators
Sequential Scan	
Hash JOIN	Nested Loop JOIN Merge JOIN
Hash Aggregate	Sort
Redistribute Motion	Broadcast Motion

Pivotal

Nested Loops and Broadcasts

Example: Nested loop query

```
SELECT * FROM factontimeperformance f1,
  factontimeperformance f2
WHERE f1.originairportid ~* 'JAX' AND
   f2.quarterid = 1;
```

Outer table is scanned and the filter is applied over each row

Broadcast is performed on the table determined by Greenplum to be smaller

Eliminate Nested Loop Joins

```
*  QUERY PLAN
1  Gather Motion 2:1  (slice2; segments: 2)  (cost=0.00..2913665.50 rows=402032 width=568)
2   ->  Nested Loop  (cost=0.00..2912640.19 rows=201016 width=568)
3       Join Filter: true
4       ->  Broadcast Motion 2:2  (slice1; segments: 2)  (cost=0.00..938.09 rows=1 width=284)
5           ->  Sequence  (cost=0.00..938.08 rows=1 width=284)
6               ->  Partition Selector for factontimeperformance (dynamic scan id: 2)  (cost=10.00..100.00 rows=50 width=4)
7                   Partitions selected:  46 (out of 46)
8               ->  Dynamic Table Scan on factontimeperformance (dynamic scan id: 2)  (cost=0.00..938.08 rows=1 width=2...
9                   Filter: quarterid = 2
10      ->  Sequence  (cost=0.00..938.08 rows=201016 width=284)
11          ->  Partition Selector for factontimeperformance (dynamic scan id: 1)  (cost=10.00..100.00 rows=50 width=4)
12              Partitions selected:  46 (out of 46)
13          ->  Dynamic Table Scan on factontimeperformance (dynamic scan id: 1)  (cost=0.00..938.08 rows=201016 width...
14              Filter: originairportid ~* 'JAX'::text
15 Settings:  optimizer=on
16 Optimizer status: PQO version 1.597
```

Note: Question the use of nested loop joins.

Pivotal

Eliminate Large Table Broadcast Motion

```
->  Hash  (cost=18039.92..18039.92 rows=20 width=66)
                              ->  Redistribute Motion
24:24  (slice3)  (cost=0.55..18039.92 rows=20 width=66)
                                    Hash Key:
cust_contact_activity.src_system_id::text
                                    ->  Hash Join  (cost=0.55..18039.52 rows=20
width=66)
                                          Hash Cond:
cust_contact_activity.contact_id::text = cust_contact.contact_id::text
                                          ->  Seq Scan on
cust_contact_activity  (cost=0.00..15953.63 rows=833663
                                           width=39)
                                    Hash  (cost=0.25..0.25 rows=24
width=84)
                                                ->  Broadcast Motion 24:24 (slice2)
(cost=0.00..0.25 rows=24 width=84)
                                                      ->  Seq Scan on
cust_contact  (cost=0.00..0.00 rows=1 width=84)
```

A small table broadcast is acceptable.

Note: adjust the value of gp_segments_for_planner to increase the cost of the motion to favor a redistribute motion over a broadcast.

Pivotal

work_mem (or, statement_mem)

```
 -> HashAggregate (cost=74852.40..84739.94 rows=791003 width=45)
          Group By: l_orderkey, l_partkey, l_comment
          Rows out: 2999671 rows (seg1) with 13345 ms to first row, 71558 ms to
end, start offset by 3.533 ms.
          Executor memory: 2645K bytes avg, 5019K bytes max (seg1).
          Work_mem used: 2321K bytes avg, 4062K bytes max (seg1).
          Work_mem wanted: 237859K bytes avg, 237859K bytes max (seg1) to lessen
workfile I/O
          affecting 1 workers.
          . . .
          -> Seq Scan on lineitem (cost=0.00..44855.70 rows=2999670 width=45)
          Rows out: 2999671 rows (seg1) with 0.571 ms to first row, 4167 ms to
end, start offset by 4.105
Slice statistics:
(slice0) Executor memory: 211K bytes.
(slice1) * Executor memory: 2840K bytes avg x 2 workers, 5209K bytes max (seg1).
Work_mem: 4062K bytes max, 237859K bytes wanted.
Settings: work_mem=4MB
Total runtime: 73326.082 ms
(24 rows)
```

Workfiles (Spill Files)

Consider the following:

- Operations performed in memory are optimal
- Insufficient memory means rows will be written out to disk as spill files
- There is a certain amount of overhead with any disk I/O operation

Spill files:

- Are located within the `pgsql_tmp` directory for the database
- Are named to indicate the node operation

```
[gpadmin:/gpdata/segments/gpseg1/base/17144/pgsql_tmp] ls -l
      total 334464
      -rw------- 1 gpadmin gpadmin 163M Jan 9 23:41
pgsql_tmp_SortTape_Slice1_14022.205
```

Pivotal

Workfile Improvements – Management

- Improvements have been made in the 4.2.5 and above release to workfiles (also known as spill files).
- There has historically been no supported way to manage the size of these workfiles or to understand what workfiles exist in previous versions.
- There are new parameter options to limit the size of workfiles created:

`gp_workfile_limit_per_query`	Limits the amount of workfile space that any particular query can use; protects against excessive workfile sizes
`gp_workfile_limit_per_segment`	Limits the amount of workfile space that can be used on any particular segment server; Protects against “out of disk space” errors
`gp_workfile_compress_algorithm`	Compression algorithm to use on spill files generated by hash aggregation or hash join operations

Workfile Improvements – Management Views

View	Description
`gp_workfile_entries`	Lists individual workfiles. The view contains one row for each operator using disk space for workfiles on a segment at the current time
`gp_workfile_usage_per_query`	Rollup of workfiles per query. The view contains one row for each query using disk space for workfiles on a segment at the current time.
`gp_workfile_usage_per_segment`	Rollup of workfiles per segment. The view contains one row for each segment. Each row displays the total amount of disk space used for workfiles on the segment at the current time.

Pivotal

Identify Re-spill in Hash Agg Operations

```
. . .
       -> HashAggregate (cost=74852.40..84739.94 rows=791003 width=45)
     Group By: l_orderkey, l_partkey, l_comment
     Rows out: 2999671 rows (seg1) with 13345 ms to first row, 71558 ms to end . .
.
     Executor memory: 2645K bytes avg, 5019K bytes max (seg1).
     Work_mem used: 2321K bytes avg, 4062K bytes max (seg1).
     Work_mem wanted: 237859K bytes avg, 237859K bytes max (seg1) to lessen
workfile I/O
     affecting 1 workers.
     (seg1) 2999671 groups total in 5 batches; 64 respill passes; 23343536 respill
rows.
     (seg1) Initial pass: 44020 groups made from 44020 rows; 2955651 rows spilled
to workfile.
     (seg1) Hash chain length 5.0 avg, 18 max, using 602986 of 607476 buckets.
     -> Seq Scan on lineitem (cost=0.00..44855.70 rows=2999670 width=45)
     Rows out: 2999671 rows (seg1) with 0.571 ms to first row, 4167 ms to end,
start offset by 4.105
Slice statistics:
(slice0) Executor memory: 211K bytes.
(slice1) * Executor memory: 2840K bytes avg x 2 workers, 5209K bytes max (seg1).
Work_mem: 4062K bytes max, 237859K bytes wanted.
Settings: work_mem=4MB
```

Pivotal

Review JOIN Order – Query Plan

```
Aggregate (cost=16141.02..16141.03 rows=1 width=0)
-> Gather Motion 2:1 (slice2) (cost=16140.97..16141.00 rows=1 width=0)
             -> Aggregate (cost=16140.97..16140.98 rows=1 width=0)
                    -> Hash Join (cost=2999.46..15647.43 rows=197414 width=0)
                        Hash Cond: ps.ps_partkey = part.p_partkey
                        -> Hash Join (cost=240.46..9920.71 rows=200020 width=4)
                            Hash Cond: ps.ps_suppkey = supplier.s_suppkey
                             -> Seq Scan on partsupp ps (cost=0.00..6180.00
rows=400000 width=8)
                             -> Hash (cost=177.97..177.97 rows=4999 width=4)
                                    -> Broadcast Motion 2:2 (slice1)
(cost=0.00..177.97 rows=4999
                                                          width=4)
                                       -> Seq Scan on supplier
(cost=0.00..77.99 rows=4999 width=4)
                        -> Hash (cost=1509.00..1509.00 rows=100000 width=4)
                                -> Seq Scan on part (cost=0.00..1509.00
rows=100000 width=4)
```

Use `join_collapse_limit` to Specify Join Order

To *specify* the join order:

- Set the parameter, `join_collapse_limit=1`
- Use ANSI style join syntax, as in the following example:

```
SELECT COUNT(*) FROM partsupp ps
    JOIN supplier ON ps_suppkey = s_suppkey
    JOIN part ON p_partkey = ps_partkey;
```

The following is an example of a non-ANSI style join:

```
SELECT COUNT(*) FROM partsupp, supplier, part
    WHERE ps_suppkey = s_suppkey
    AND p_partkey = ps_partkey;
```

Pivotal

Review

- Identify plan nodes with a large number of rows and high cost
- Validate partitions are being eliminated
- Eliminate large table broadcast motion
- Prefer the fast/efficient operators
- Identify spill files and increase memory
- Identify respill in HashAggregate
- Review join order

Pivotal


Pivotal
A NEW PLATFORM FOR A NEW ERA