WHY ISN'T MY QUERY USING AN INDEX?

TIPS ON SQL PERFORMANCE TO KEEP ON YOUR FINGER TIPS!

DENISH PATEL

SENIOR DATABASE ARCHITECT

WHO AM I?

- Denish Patel
  - Senior Database Architect
  - Data Engineering – Hadoop, NiFi , Spark
  - DBA – Postgres, Oracle, MySQL and SQL Server
  - DevOps – Ansible, CI/CD, Git, Database Reliability Engineering
- Blog: http://www.pateldenish.com/
- Twitter: https://twitter.com/DenishPatel
- Slack: https://postgres-slack.herokuapp.com/
- Email: Denish.j.patel@gmail.com

Denish Patel
@DenishPatel

Lead Database engineer 'Use documentation as Map, not GPS.' - Denish

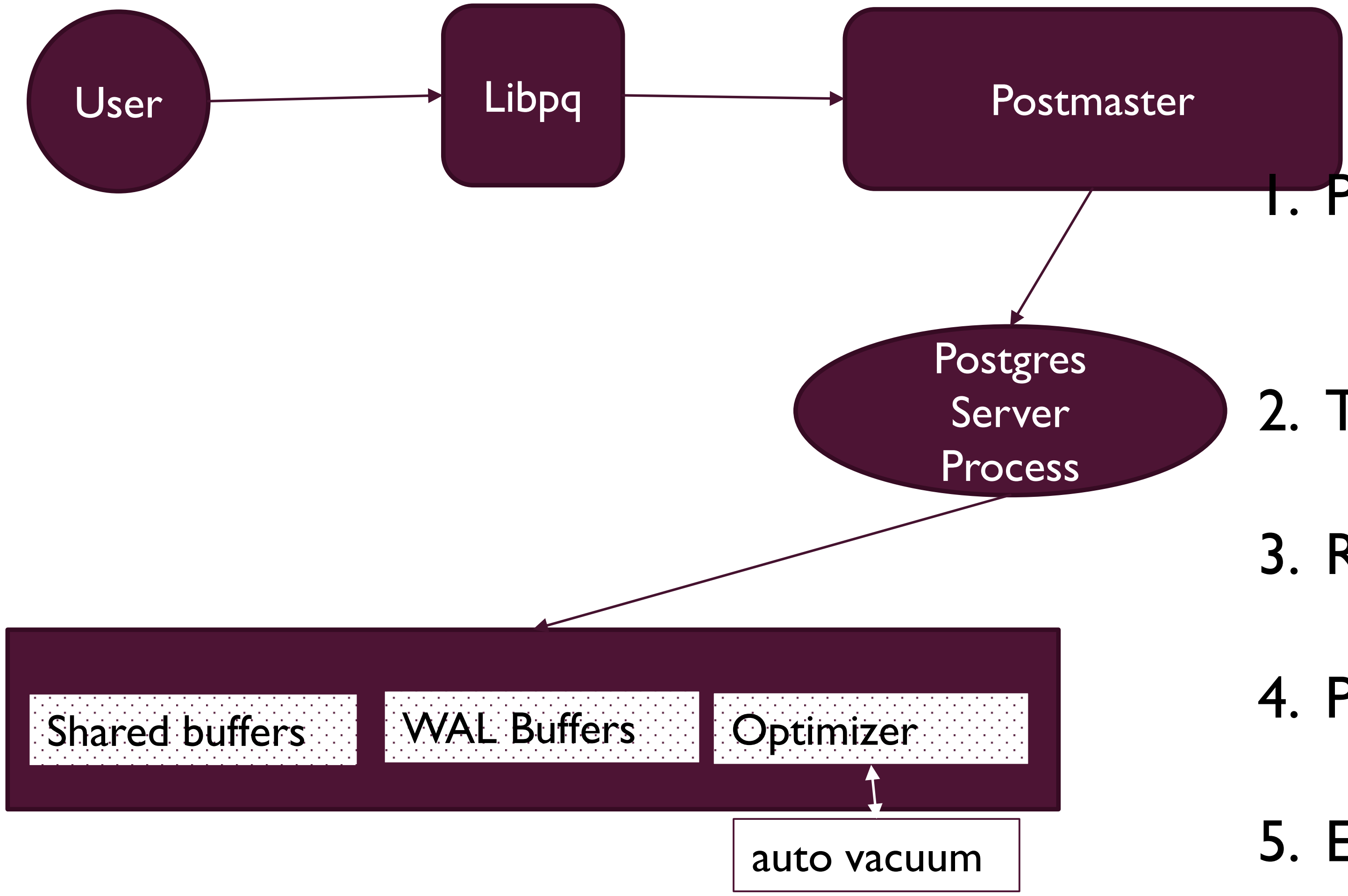
Baltimore-Washington DC

about.me/denish.patel

AGENDA

- Postgres Query Execution Architecture
- How optimizer decides execution plan from choices?
- How to read query plans?
- Q/A

POSTGRES QUERY EXECUTION



1. Parser
   - Semantic & privilege checking
   - Spelling checking
2. Traffic Cop
   - Simple and complex parsing
3. Rewriter
   - apply rules
4. Planner/Optimizer
   - Generate plans
5. Execute plan generated by optimizer

PLANNER/OPTIMIZER

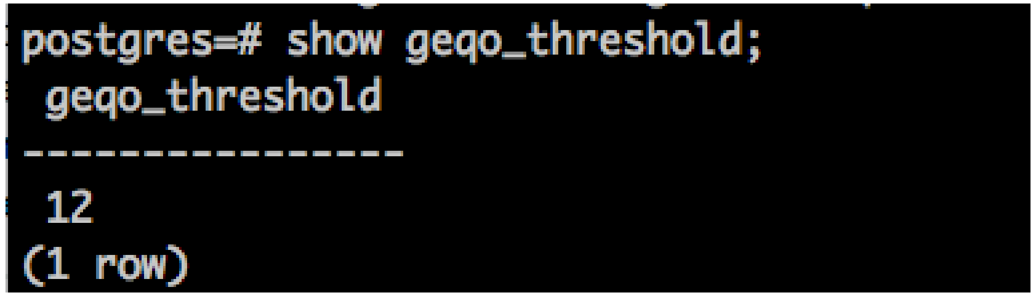
- The task of the planner/optimizer is to create an optimal execution plan.
  - Brain!
- The planner/optimizer starts by generating plans for scanning each individual relation (table) used in the query
  - Available Indexes
  - Sequential scan vs Index Scan
- Query required joining <u>two</u> or more tables
  - Nested loop join
  - Merge join
  - Hash join

QUERY OPTIMIZATION

- Heuristic/Rules
  - Rewrite the query to remove stupid/inefficient things
  - Does not require a cost model
- Cost-Based Search
  - Use a cost model to evaluate multiple plans and pick the one with the lowest cost

POSTGRES PLANNER/OPTIMIZER

- If the query uses less than **geqo_threshold** relations, a near-exhaustive search algorithm conducted to find the best join sequence. The default value is for this parameter in 12.



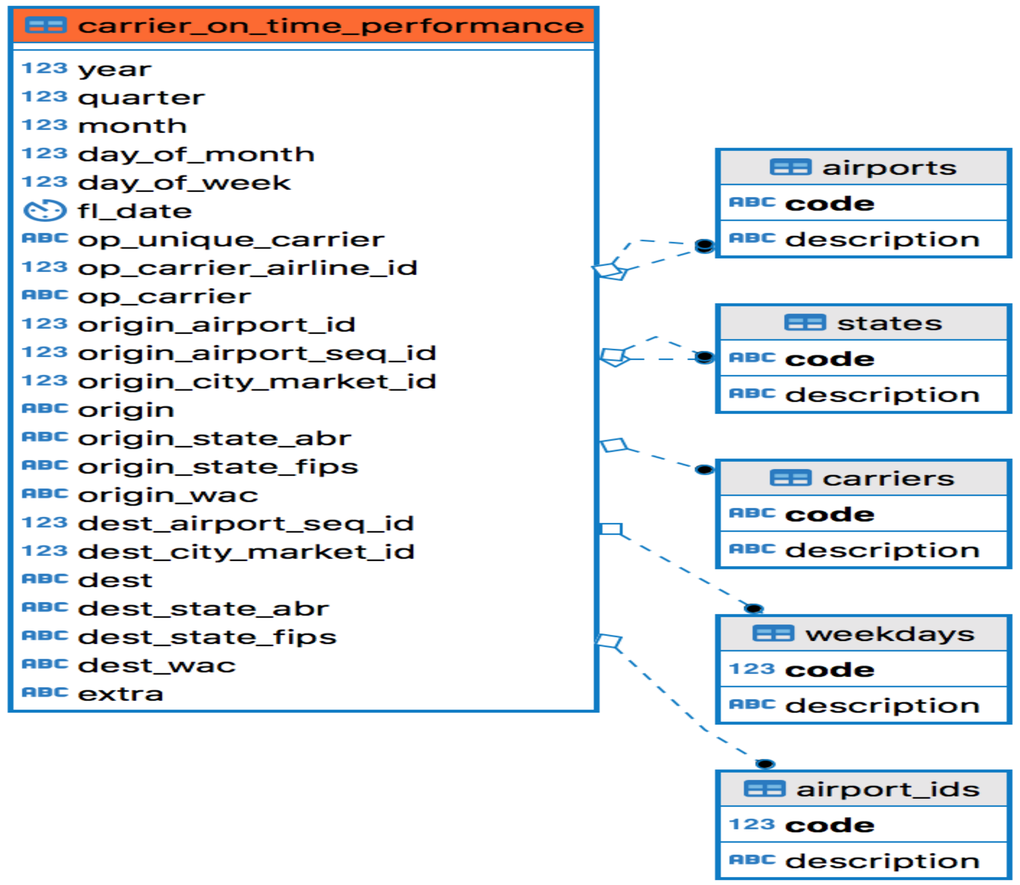
```
postgres=# show geqo_threshold;
 geqo_threshold
----------------
 12
(1 row)
```

- When geqo_threshold is exceeded, the join sequences considered are determined by heuristics search method– Genetic Algorithms (GA)

COST ESTIMATION

- How long will a query take?
  - CPU : small cost; tough to estimate
  - Disk : # of block transfers
  - Memory : amount of DRAM used
  - Network: # of messages
- How many tables will be read/written?
- What statistics to keep?

SAMPLE DATABASE – TRANSPORT STATS AIRLINES



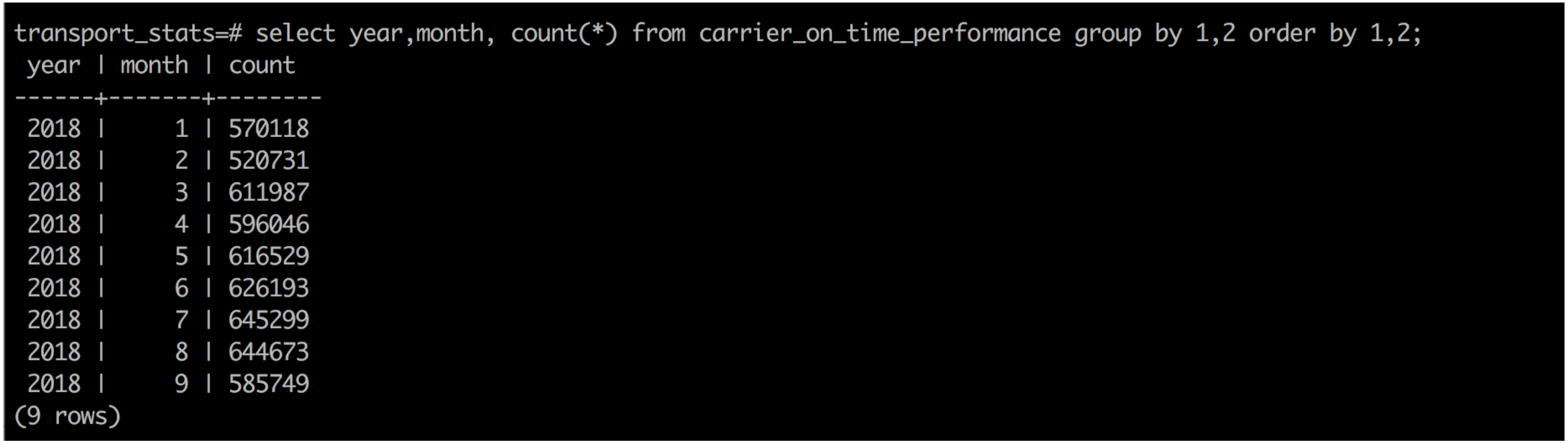
Source: https://www.transtats.bts.gov/Tables.asp?DB_ID=120&DB_Name=Airline%20On-Time%20Performance%20Data&DB_Short_Name=On-Time

SAMPLE DATABASE

- transport_stats=# select count(*) from carrier_on_time_performance;

Count

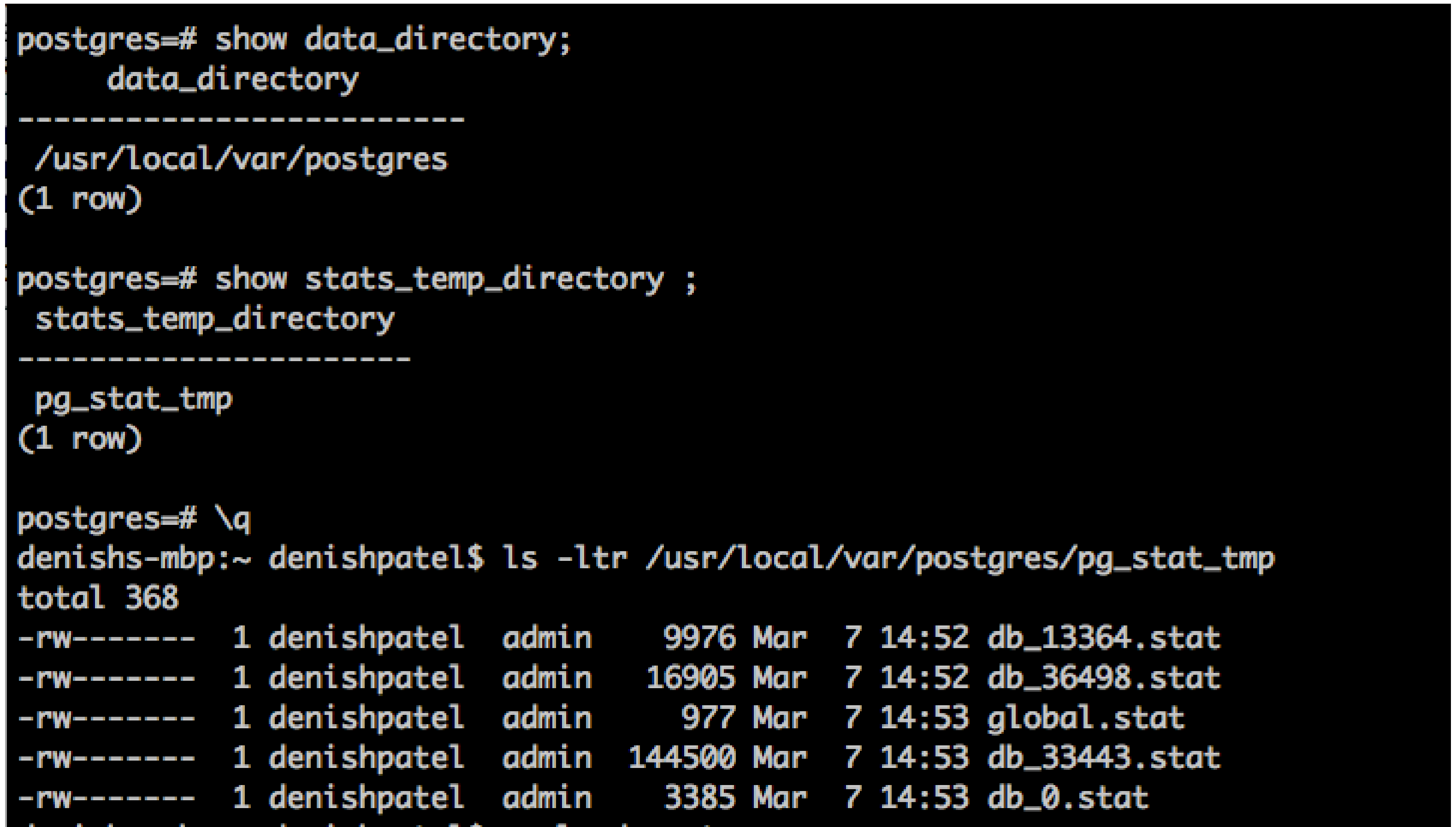
5417325



```
transport_stats=# select year,month, count(*) from carrier_on_time_performance group by 1,2 order by 1,2;
 year | month | count
------+-------+--------
 2018 |     1 | 570118
 2018 |     2 | 520731
 2018 |     3 | 611987
 2018 |     4 | 596046
 2018 |     5 | 616529
 2018 |     6 | 626193
 2018 |     7 | 645299
 2018 |     8 | 644673
 2018 |     9 | 585749
(9 rows)
```

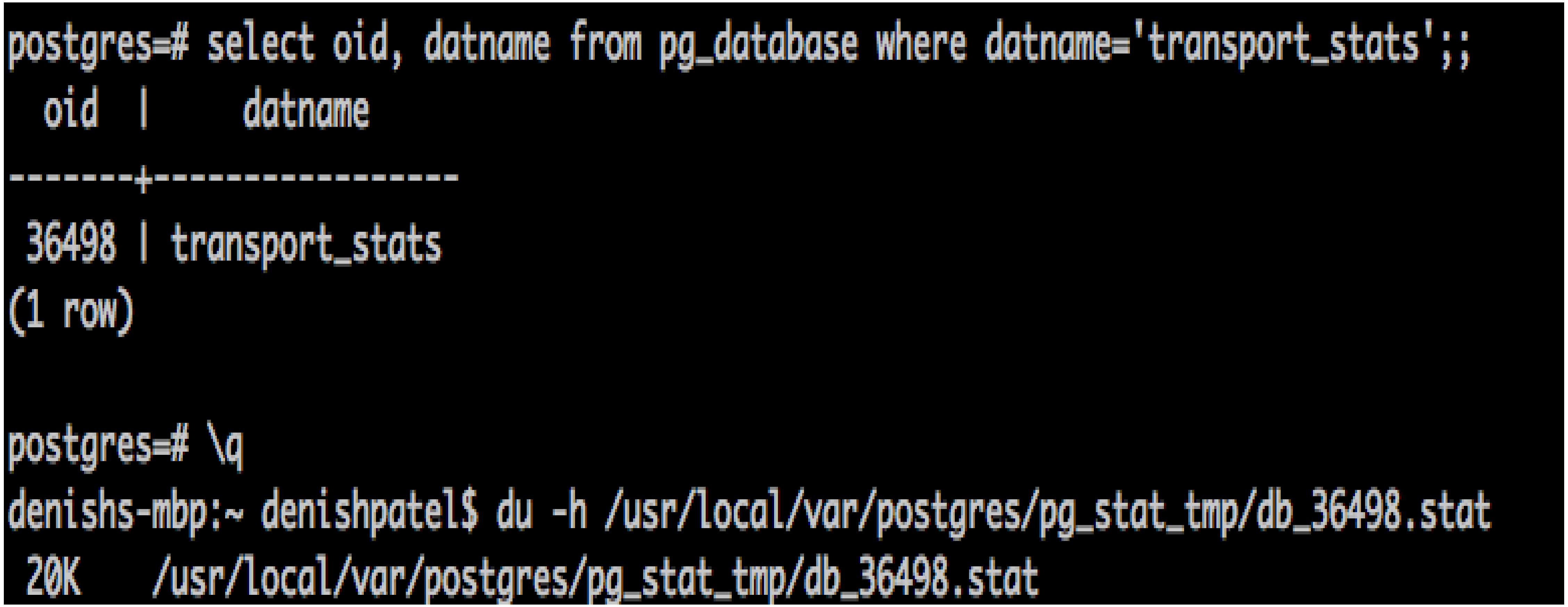
STATISTICS

- Postgres stores internal statistics about tables, attributes and indices in internal catalog
  - ANALYZE
  - VACUUM ANALYZE
  - Auto-vacuum analyze



```
postgres=# show data_directory;
     data_directory
-------------------------
 /usr/local/var/postgres
(1 row)

postgres=# show stats_temp_directory ;
 stats_temp_directory
----------------------
 pg_stat_tmp
(1 row)

postgres=# \q
denishs-mbp:~ denishpatel$ ls -ltr /usr/local/var/postgres/pg_stat_tmp
total 368
-rw-------  1 denishpatel  admin    9976 Mar  7 14:52 db_13364.stat
-rw-------  1 denishpatel  admin   16905 Mar  7 14:52 db_36498.stat
-rw-------  1 denishpatel  admin     977 Mar  7 14:53 global.stat
-rw-------  1 denishpatel  admin  144500 Mar  7 14:53 db_33443.stat
-rw-------  1 denishpatel  admin    3385 Mar  7 14:53 db_0.stat
```

STATISTICS

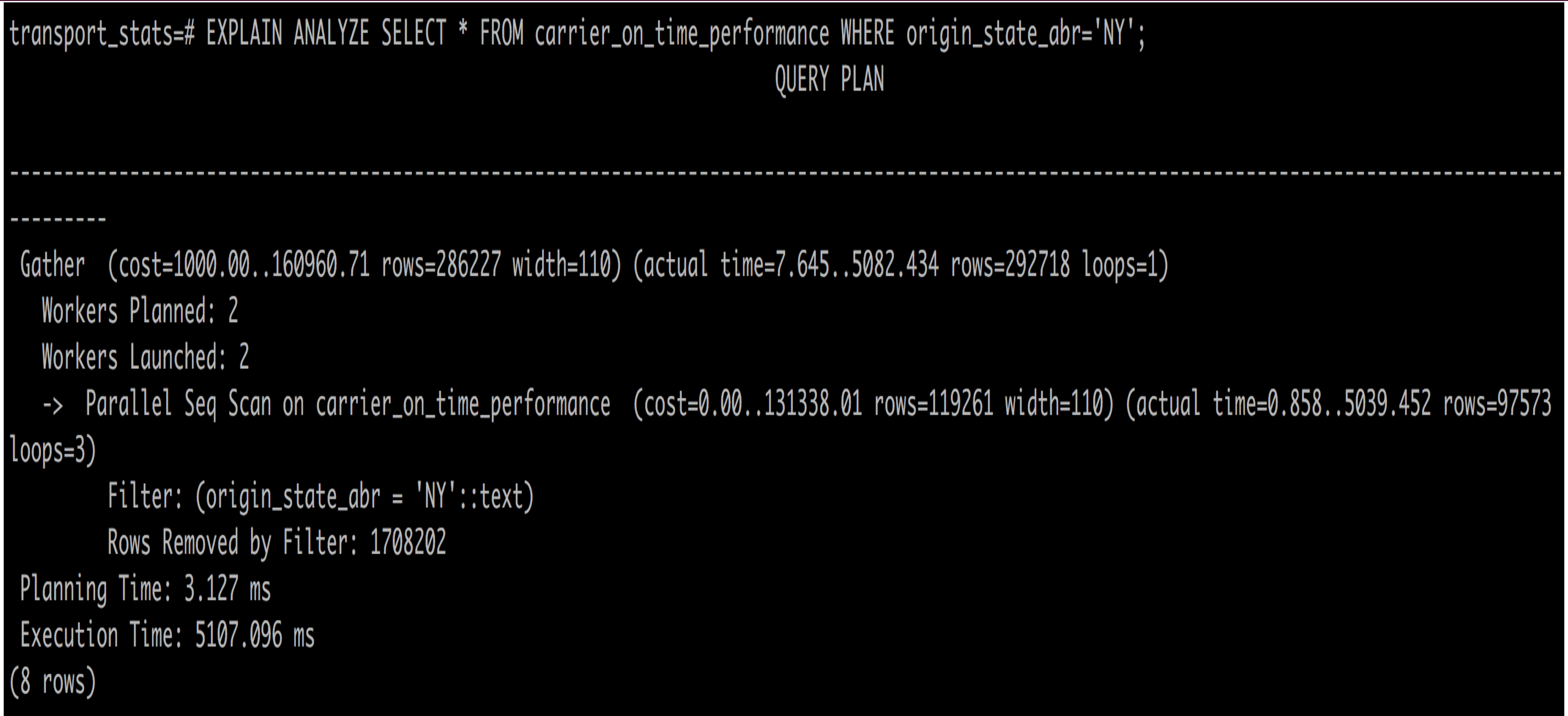


```
postgres=# select oid, datname from pg_database where datname='transport_stats';;
  oid  |     datname
-------+-----------------
 36498 | transport_stats
(1 row)

postgres=# \q
denishs-mbp:~ denishpatel$ du -h /usr/local/var/postgres/pg_stat_tmp/db_36498.stat
 20K    /usr/local/var/postgres/pg_stat_tmp/db_36498.stat
```

POSTGRES QUERY PLANS

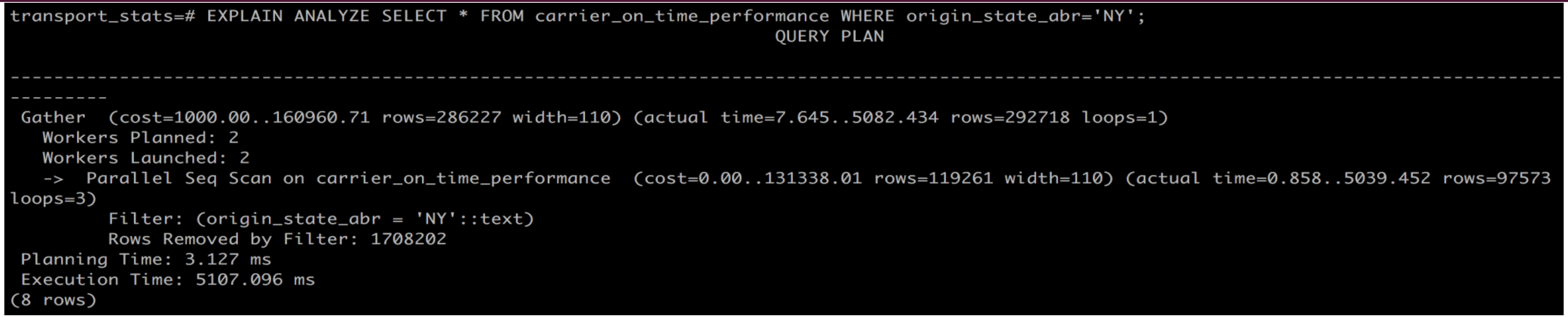
- Each query requires a Plan
- EXPLAIN is your friend!
- EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM X;
- Using ANALYZE will actually execute the query. Don't worry you can rollback
  - *BEGIN;*
  - *EXPLAIN ANALYZE UPDATE tablename WHERE X=y;*
  - *ROLLBACK;*

EXPLAIN ANALYZE



```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance WHERE origin_state_abr='NY';
                                                                      QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------
---------
 Gather  (cost=1000.00..160960.71 rows=286227 width=110) (actual time=7.645..5082.434 rows=292718 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on carrier_on_time_performance  (cost=0.00..131338.01 rows=119261 width=110) (actual time=0.858..5039.452 rows=97573 loops=3)
         Filter: (origin_state_abr = 'NY'::text)
         Rows Removed by Filter: 1708202
 Planning Time: 3.127 ms
 Execution Time: 5107.096 ms
(8 rows)
```

EXPLAINING EXPLAIN



```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance WHERE origin_state_abr='NY';
                                                                      QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------------
---------
 Gather  (cost=1000.00..160960.71 rows=286227 width=110) (actual time=7.645..5082.434 rows=292718 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on carrier_on_time_performance  (cost=0.00..131338.01 rows=119261 width=110) (actual time=0.858..5039.452 rows=97573
loops=3)
         Filter: (origin_state_abr = 'NY'::text)
         Rows Removed by Filter: 1708202
 Planning Time: 3.127 ms
 Execution Time: 5107.096 ms
(8 rows)
```

Parallel Seq Scan on carrier_on_time_performance

(cost=0.00..131338.01 rows=119261 width=110) (actual time=0.858..5039.452 rows=97573 loops=3)

Cost of retrieving first row

Cost of retrieving all rows

Number of rows returned

Avg. width of rows (bytes)

Number of times executed

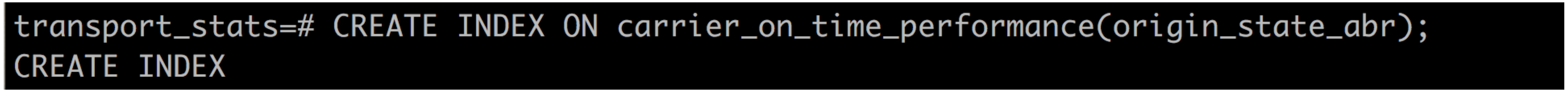
EXPLAINING THE EXPLAIN

- The costs are measured in arbitrary units determined by the planner's cost parameters
  - seq_page_cost - units of disk page fetches . Default is 1.0
  - Random_page_cost
  - cpu_tuple_cost (and others)
- Upper-level node includes the cost of all its child nodes
- Cost **does <u>not</u>** consider the time spent **transmitting result**
- **Planning time** does not include parsing or rewriting.
- **Execution time - Time spent executing AFTER triggers is <u>not</u> counted**

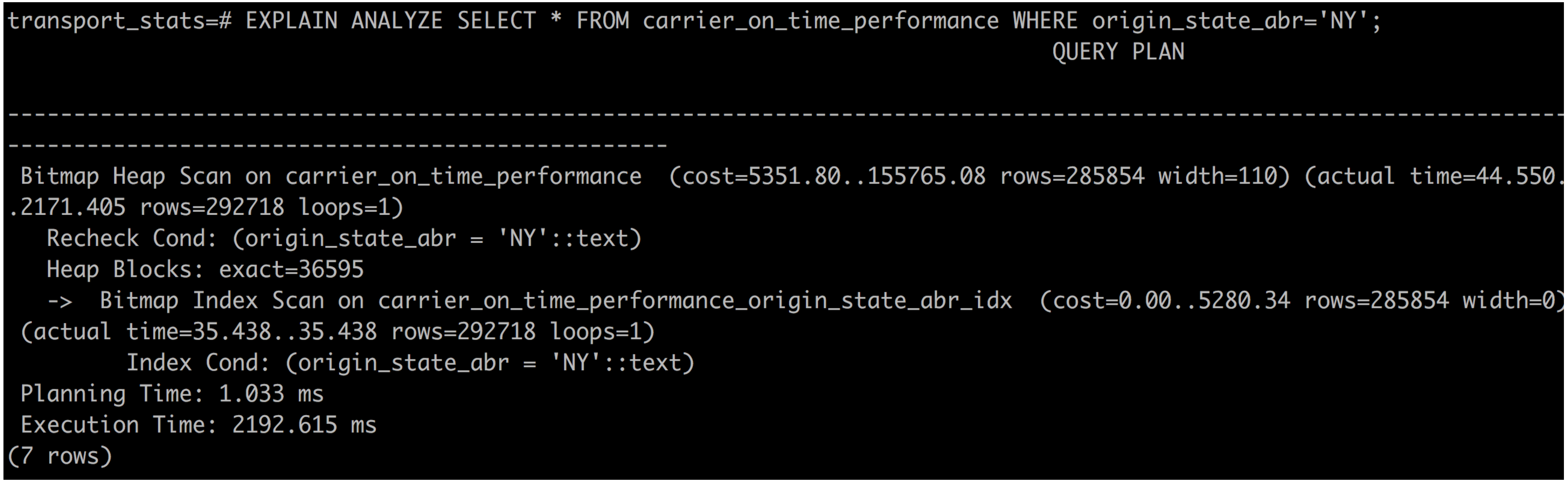
EXPLAINING EXPLAIN – LIMIT?

```
transport_stats=# explain analyze select * from public.carrier_on_time_performance where origin='BWI' and origin_state_abr='MD' limit 1;

                                                QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------
--

 Limit  (cost=0.00..156.62 rows=1 width=110) (actual time=0.038..0.039 rows=1 loops=1)

   ->  Seq Scan on carrier_on_time_performance  (cost=0.00..184345.88 rows=1177 width=110) (actual time=0.036..0.037 rows=1 loops=1)

         Filter: ((origin = 'BWI'::text) AND (origin_state_abr = 'MD'::text))

         Rows Removed by Filter: 59

 Planning Time: 1.790 ms

 Execution Time: 0.074 ms
```

LET’S CREATE INDEX



```
transport_stats=# CREATE INDEX ON carrier_on_time_performance(origin_state_abr);
CREATE INDEX
```



```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance WHERE origin_state_abr='NY';
                                                                                QUERY PLAN


----------------------------------------------------------------------------------------------------------------------
--------------------------------------------------
 Bitmap Heap Scan on carrier_on_time_performance  (cost=5351.80..155765.08 rows=285854 width=110) (actual time=44.550.
.2171.405 rows=292718 loops=1)
   Recheck Cond: (origin_state_abr = 'NY'::text)
   Heap Blocks: exact=36595
   ->  Bitmap Index Scan on carrier_on_time_performance_origin_state_abr_idx  (cost=0.00..5280.34 rows=285854 width=0)
 (actual time=35.438..35.438 rows=292718 loops=1)
         Index Cond: (origin_state_abr = 'NY'::text)
 Planning Time: 1.033 ms
 Execution Time: 2192.615 ms
(7 rows)
```

SCANS

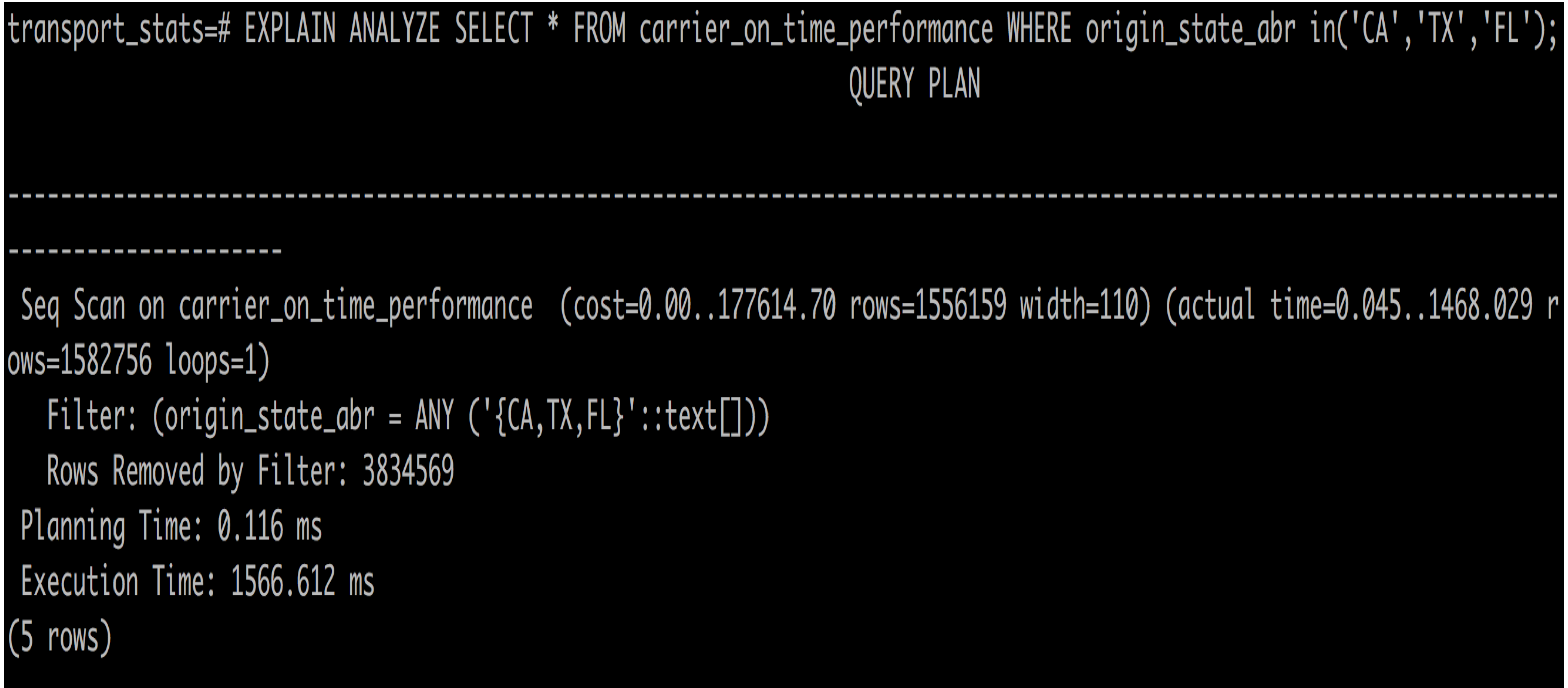
- Sequential Scan
- Bitmap Scan
- Index Scan
- What is Re-check condition?

CARDINALITY

- Uniqueness of data values contained in a column
  - **High** - percentage of totally unique values
  - **Low** - repeat data
- Index on low cardinality does not help

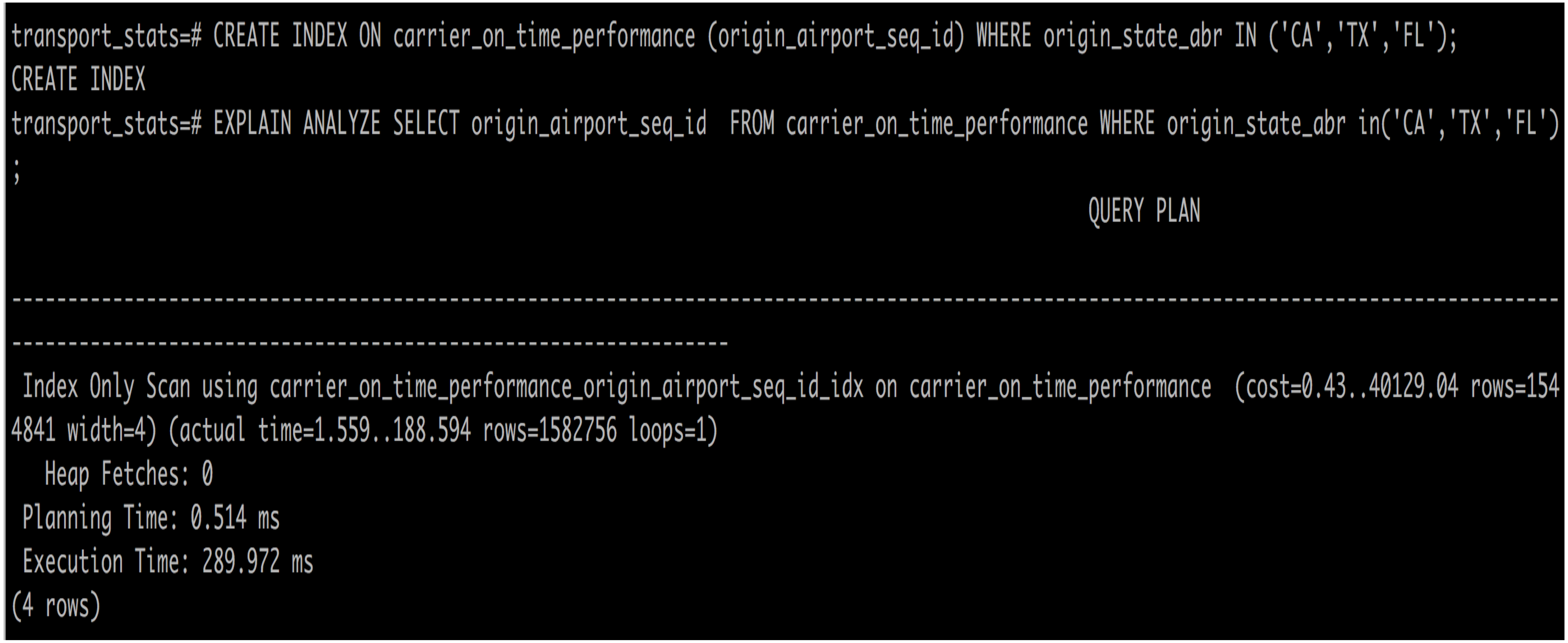
```
transport_stats=# select origin_state_abr,count(*) from carrier_on_time_performance group by 1 order by 2 desc limit 3;
 origin_state_abr | count
------------------+--------
 CA               | 595402
 TX               | 565602
 FL               | 421752
(3 rows)
```

CARDINALITY



```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance WHERE origin_state_abr in('CA','TX','FL');
                                                            QUERY PLAN

---------------------------------------------------------------------------------------------------------------------
---------------------
 Seq Scan on carrier_on_time_performance  (cost=0.00..177614.70 rows=1556159 width=110) (actual time=0.045..1468.029 r
ows=1582756 loops=1)
   Filter: (origin_state_abr = ANY ('{CA,TX,FL}'::text[]))
   Rows Removed by Filter: 3834569
 Planning Time: 0.116 ms
 Execution Time: 1566.612 ms
(5 rows)
```

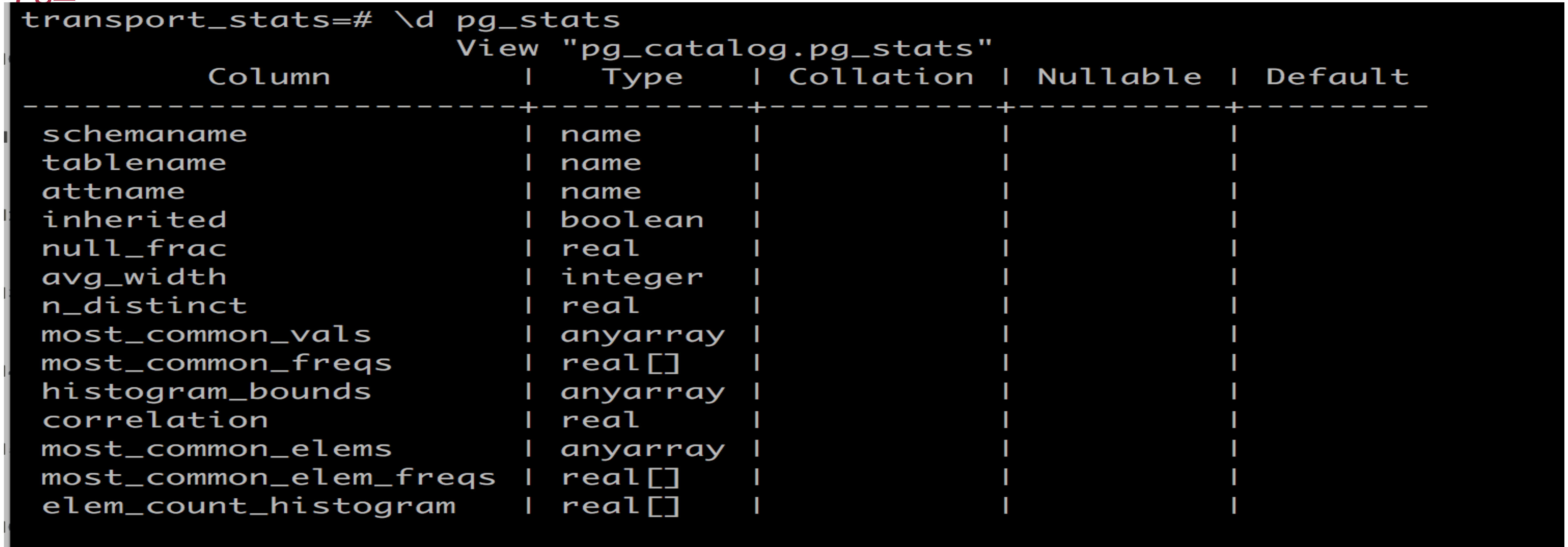
PARTIAL INDEX



```
transport_stats=# CREATE INDEX ON carrier_on_time_performance (origin_airport_seq_id) WHERE origin_state_abr IN ('CA','TX','FL');
CREATE INDEX
transport_stats=# EXPLAIN ANALYZE SELECT origin_airport_seq_id  FROM carrier_on_time_performance WHERE origin_state_abr in('CA','TX','FL')
;
                                                                                          QUERY PLAN


-----------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
 Index Only Scan using carrier_on_time_performance_origin_airport_seq_id_idx on carrier_on_time_performance  (cost=0.43..40129.04 rows=154
4841 width=4) (actual time=1.559..188.594 rows=1582756 loops=1)
   Heap Fetches: 0
 Planning Time: 0.514 ms
 Execution Time: 289.972 ms
(4 rows)
```

ROW ESTIMATION

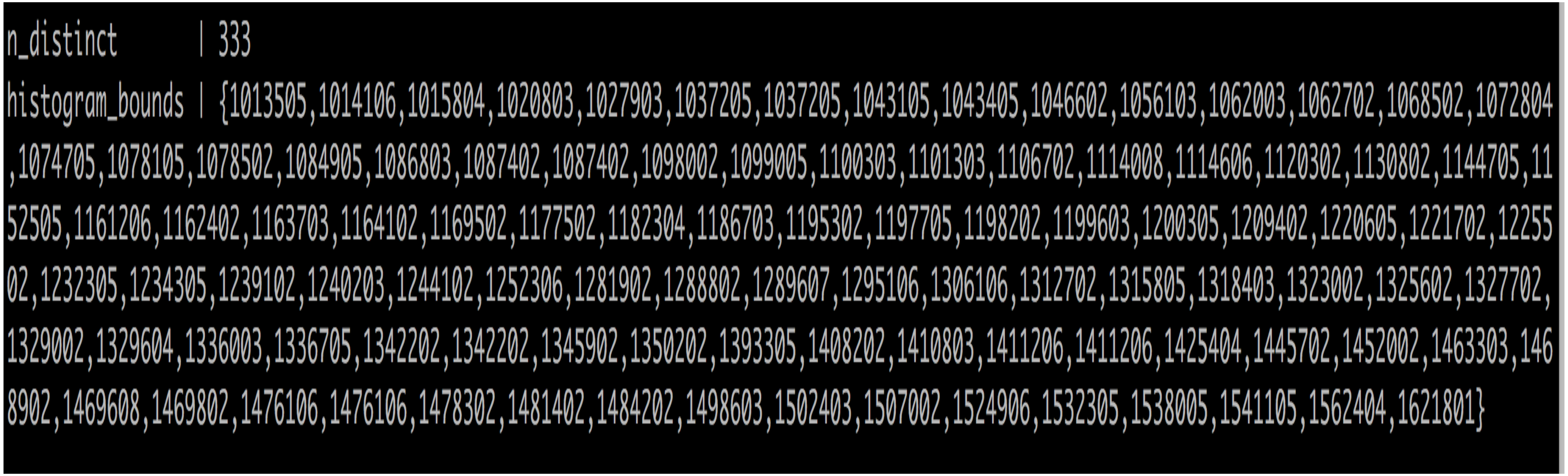
- Postgres keep tracks of histogram values for row estimation in pg_statistics table
- *pg_stats* view



```
transport_stats=# \d pg_stats
                     View "pg_catalog.pg_stats"
         Column          |   Type    | Collation | Nullable | Default
-------------------------+-----------+-----------+----------+---------
 schemaname              | name      |           |          |
 tablename               | name      |           |          |
 attname                 | name      |           |          |
 inherited               | boolean   |           |          |
 null_frac               | real      |           |          |
 avg_width               | integer   |           |          |
 n_distinct              | real      |           |          |
 most_common_vals        | anyarray  |           |          |
 most_common_freqs       | real[]    |           |          |
 histogram_bounds        | anyarray  |           |          |
 correlation             | real      |           |          |
 most_common_elems       | anyarray  |           |          |
 most_common_elem_freqs  | real[]    |           |          |
 elem_count_histogram    | real[]    |           |          |
```

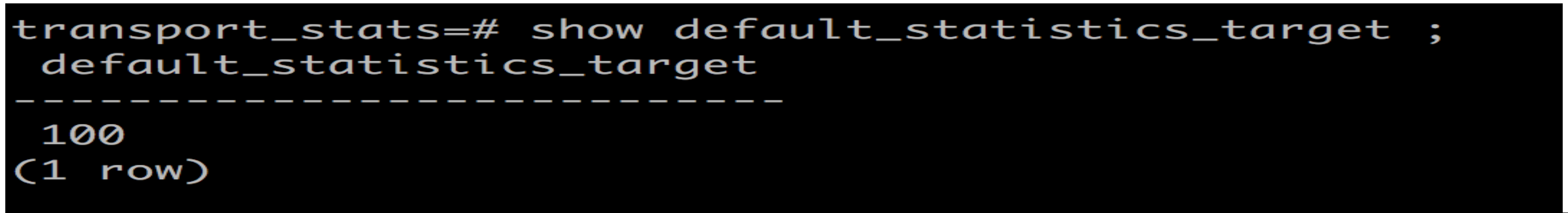
HISTOGRAMS

- SELECT n_distinct, histogram_bounds FROM pg_stats WHERE tablename ='carrier_on_time_performance' AND attname='origin_airport_seq_id’;

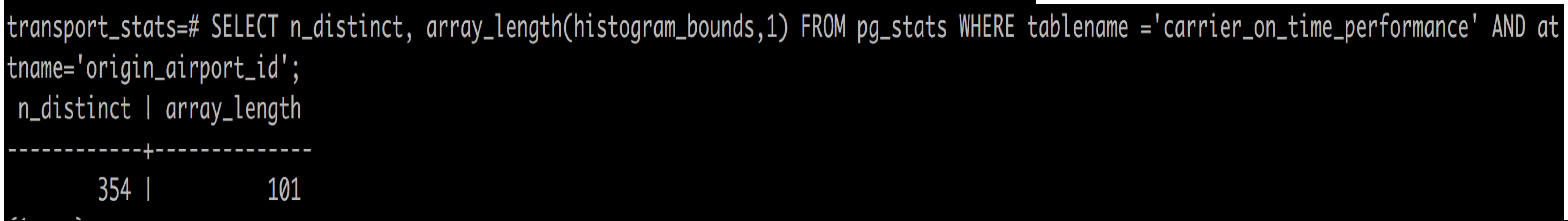


```
n_distinct       | 333
histogram_bounds | {1013505,1014106,1015804,1020803,1027903,1037205,1037205,1043105,1043405,1046602,1056103,1062003,1062702,1068502,1072804
,1074705,1078105,1078502,1084905,1086803,1087402,1087402,1098002,1099005,1100303,1101303,1106702,1114008,1114606,1120302,1130802,1144705,11
52505,1161206,1162402,1163703,1164102,1169502,1177502,1182304,1186703,1195302,1197705,1198202,1199603,1200305,1209402,1220605,1221702,12255
02,1232305,1234305,1239102,1240203,1244102,1252306,1281902,1288802,1289607,1295106,1306106,1312702,1315805,1318403,1323002,1325602,1327702,
1329002,1329604,1336003,1336705,1342202,1342202,1345902,1350202,1393305,1408202,1410803,1411206,1411206,1425404,1445702,1452002,1463303,146
8902,1469608,1469802,1476106,1476106,1478302,1481402,1484202,1498603,1502403,1507002,1524906,1532305,1538005,1541105,1562404,1621801}
```

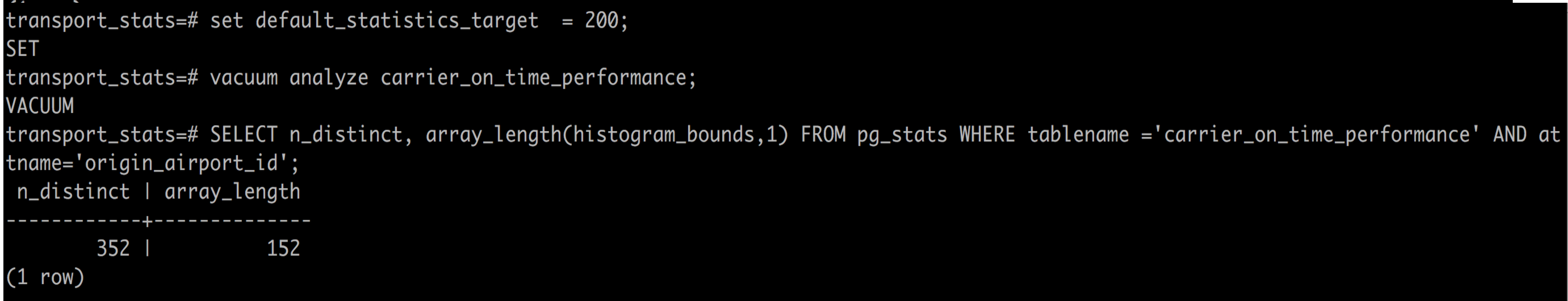
DEFAULT_STATISTICS_TARGET



```
transport_stats=# show default_statistics_target ;
 default_statistics_target
---------------------------
 100
(1 row)
```



```
transport_stats=# SELECT n_distinct, array_length(histogram_bounds,1) FROM pg_stats WHERE tablename ='carrier_on_time_performance' AND at
tname='origin_airport_id';
 n_distinct | array_length
------------+--------------
        354 |          101
```



```
transport_stats=# set default_statistics_target  = 200;
SET
transport_stats=# vacuum analyze carrier_on_time_performance;
VACUUM
transport_stats=# SELECT n_distinct, array_length(histogram_bounds,1) FROM pg_stats WHERE tablename ='carrier_on_time_performance' AND at
tname='origin_airport_id';
 n_distinct | array_length
------------+--------------
        352 |          152
(1 row)
```

DEFAULT_STATISTICS_TARGET

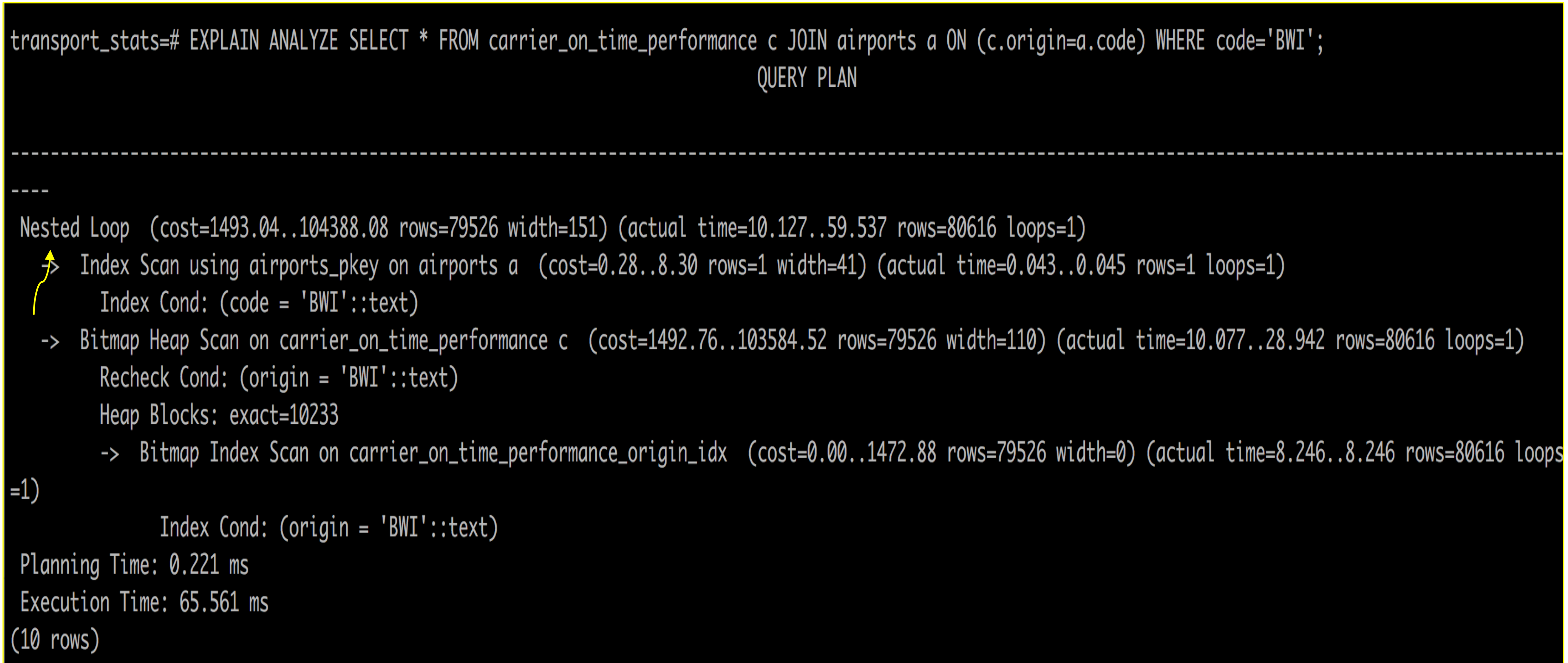
- transport_stats=# alter table carrier_on_time_performance alter COLUMN origin_airport_id set statistics 1000;
- ALTER TABLE

- transport_stats=# alter table carrier_on_time_performance alter COLUMN origin_airport_id set statistics -1;
- ALTER TABLE

LET'S TALK ABOUT JOIN

- Nested Loop
- Hash Join
- Merge Join

NESTED LOOP

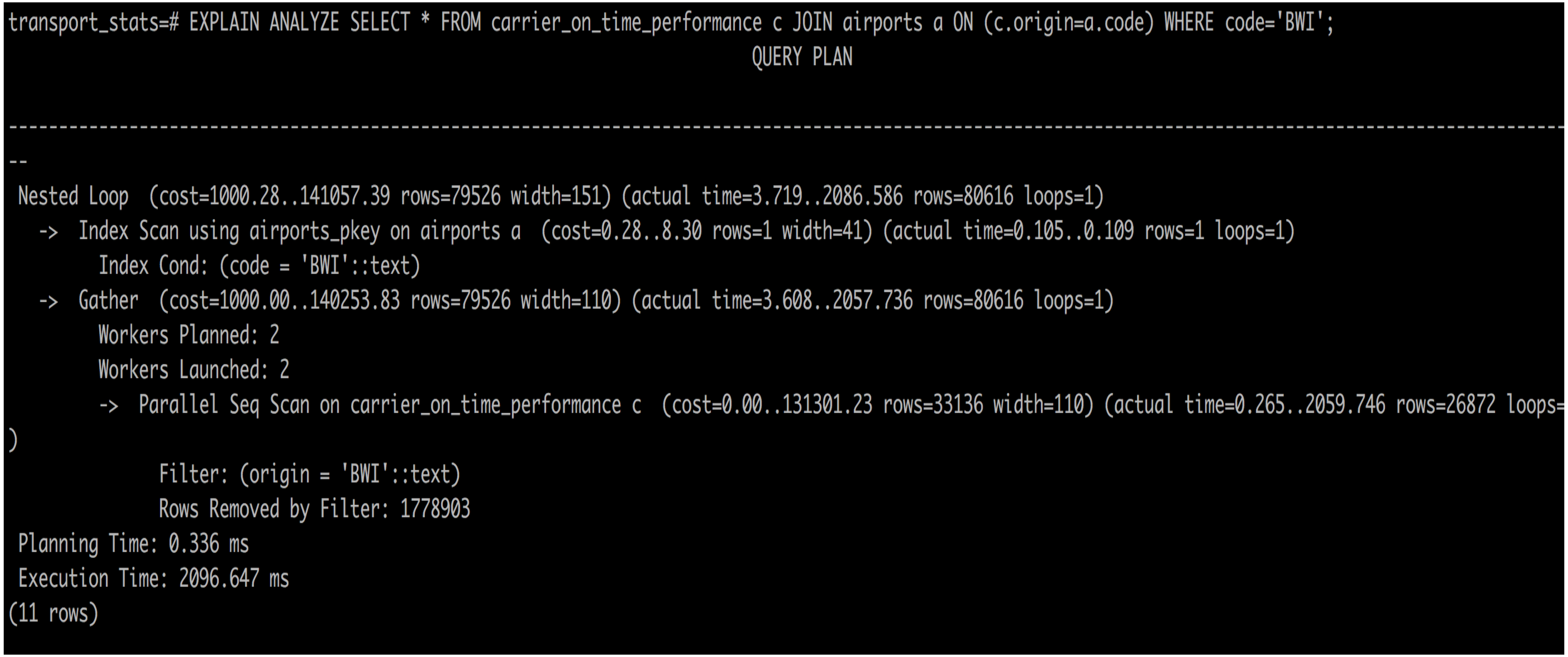


```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) WHERE code='BWI';
                                                                      QUERY PLAN

------------------------------------------------------------------------------------------------------------------------------------------
----
 Nested Loop  (cost=1493.04..104388.08 rows=79526 width=151) (actual time=10.127..59.537 rows=80616 loops=1)
   ->  Index Scan using airports_pkey on airports a  (cost=0.28..8.30 rows=1 width=41) (actual time=0.043..0.045 rows=1 loops=1)
         Index Cond: (code = 'BWI'::text)
   ->  Bitmap Heap Scan on carrier_on_time_performance c  (cost=1492.76..103584.52 rows=79526 width=110) (actual time=10.077..28.942 rows=80616 loops=1)
         Recheck Cond: (origin = 'BWI'::text)
         Heap Blocks: exact=10233
         ->  Bitmap Index Scan on carrier_on_time_performance_origin_idx  (cost=0.00..1472.88 rows=79526 width=0) (actual time=8.246..8.246 rows=80616 loops
=1)
               Index Cond: (origin = 'BWI'::text)
 Planning Time: 0.221 ms
 Execution Time: 65.561 ms
(10 rows)
```

NETSTED LOOP

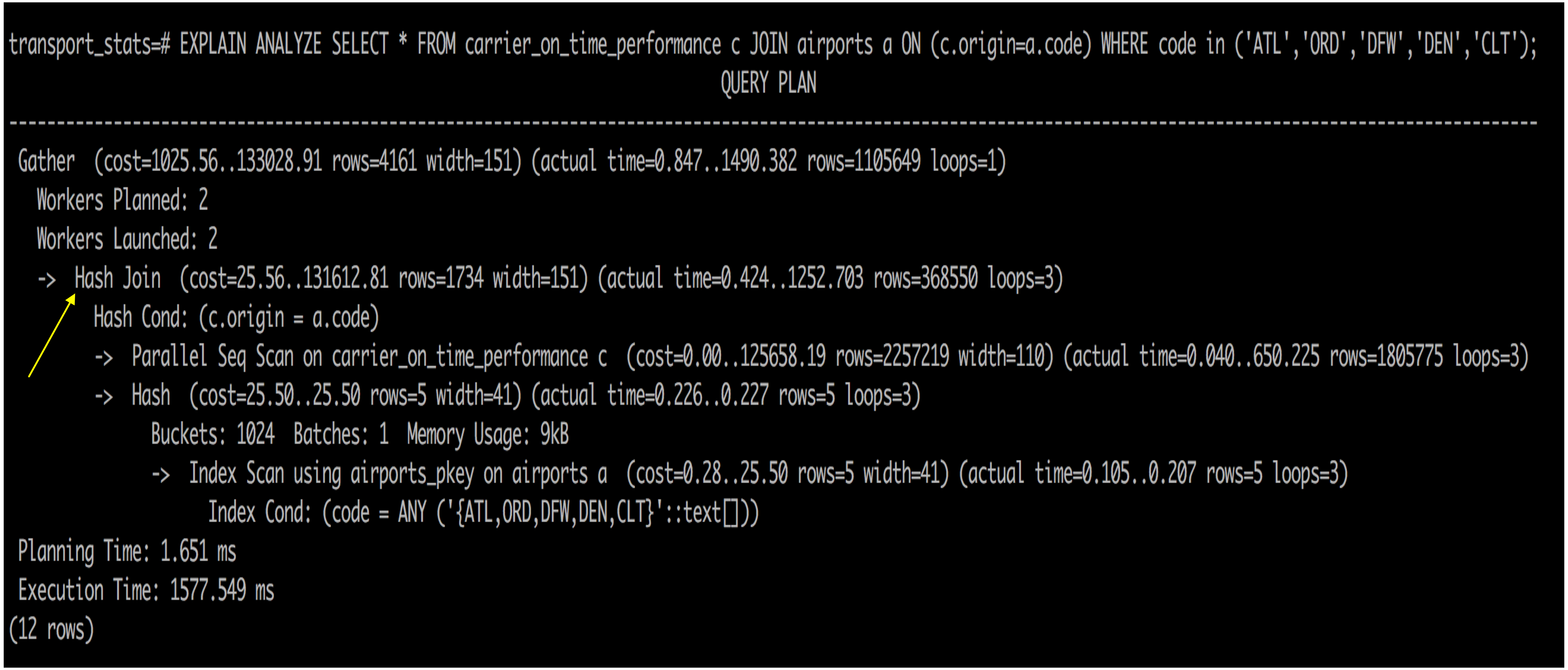
- Iterate all entries form “airports” and find relevant entries from “carrier_on_time_performance” table
- Emitting rows with WHERE clause (WHERE airport code='BWI’)
- Slower in performance (if index is not used)
- Make sure relevant index exist to match WHERE clause
- A nested loop is the only join algorithm Postgres has that can be used to process any join!

NETSTED LOOP – NO INDEX



```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) WHERE code='BWI';
                                                                          QUERY PLAN


-----------------------------------------------------------------------------------------------------------------------------------------------------------
--
 Nested Loop  (cost=1000.28..141057.39 rows=79526 width=151) (actual time=3.719..2086.586 rows=80616 loops=1)
   ->  Index Scan using airports_pkey on airports a  (cost=0.28..8.30 rows=1 width=41) (actual time=0.105..0.109 rows=1 loops=1)
         Index Cond: (code = 'BWI'::text)
   ->  Gather  (cost=1000.00..140253.83 rows=79526 width=110) (actual time=3.608..2057.736 rows=80616 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Parallel Seq Scan on carrier_on_time_performance c  (cost=0.00..131301.23 rows=33136 width=110) (actual time=0.265..2059.746 rows=26872 loops=
)
               Filter: (origin = 'BWI'::text)
               Rows Removed by Filter: 1778903
 Planning Time: 0.336 ms
 Execution Time: 2096.647 ms
(11 rows)
```

HASH JOIN

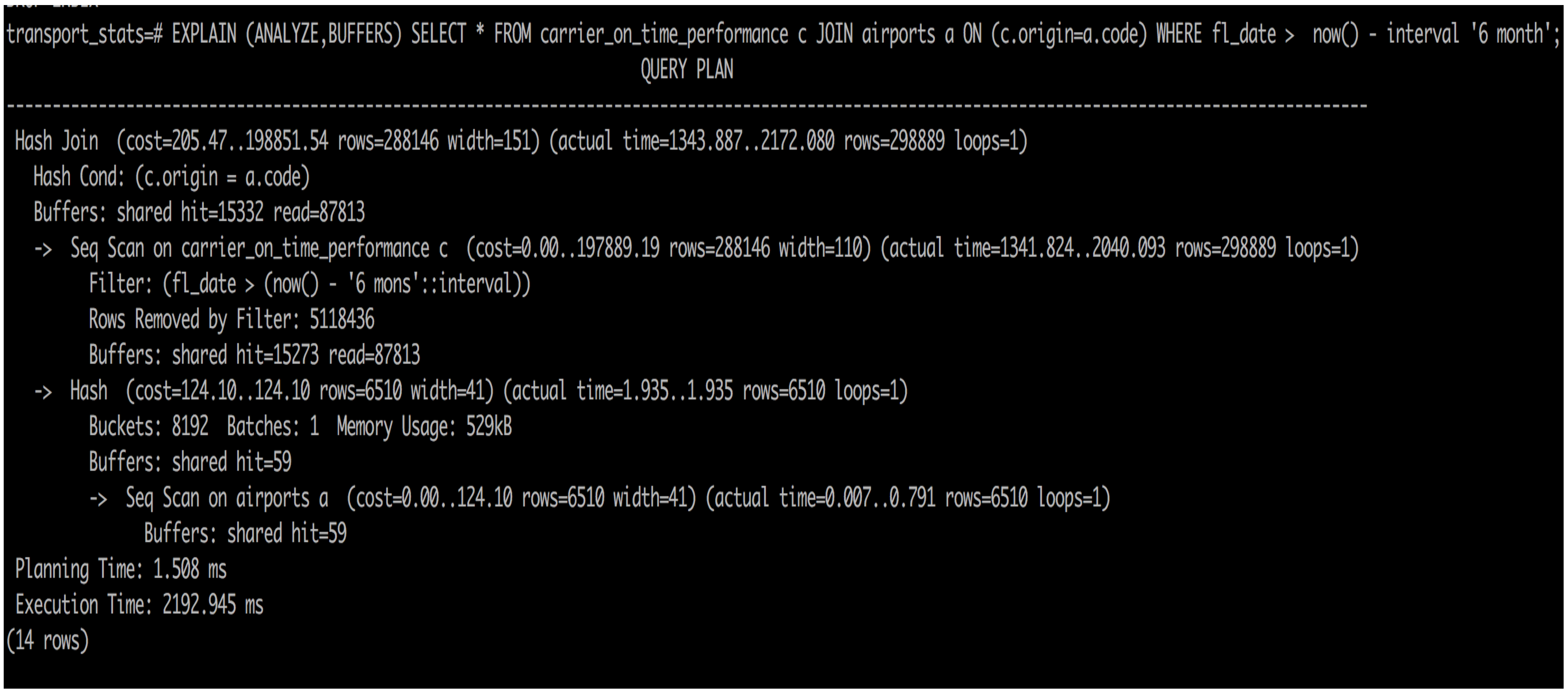


```
transport_stats=# EXPLAIN ANALYZE SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) WHERE code in ('ATL','ORD','DFW','DEN','CLT');
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1025.56..133028.91 rows=4161 width=151) (actual time=0.847..1490.382 rows=1105649 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Hash Join  (cost=25.56..131612.81 rows=1734 width=151) (actual time=0.424..1252.703 rows=368550 loops=3)
         Hash Cond: (c.origin = a.code)
         ->  Parallel Seq Scan on carrier_on_time_performance c  (cost=0.00..125658.19 rows=2257219 width=110) (actual time=0.040..650.225 rows=1805775 loops=3)
         ->  Hash  (cost=25.50..25.50 rows=5 width=41) (actual time=0.226..0.227 rows=5 loops=3)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Index Scan using airports_pkey on airports a  (cost=0.28..25.50 rows=5 width=41) (actual time=0.105..0.207 rows=5 loops=3)
                     Index Cond: (code = ANY ('{ATL,ORD,DFW,DEN,CLT}'::text[]))
 Planning Time: 1.651 ms
 Execution Time: 1577.549 ms
(12 rows)
```

HASH JOIN

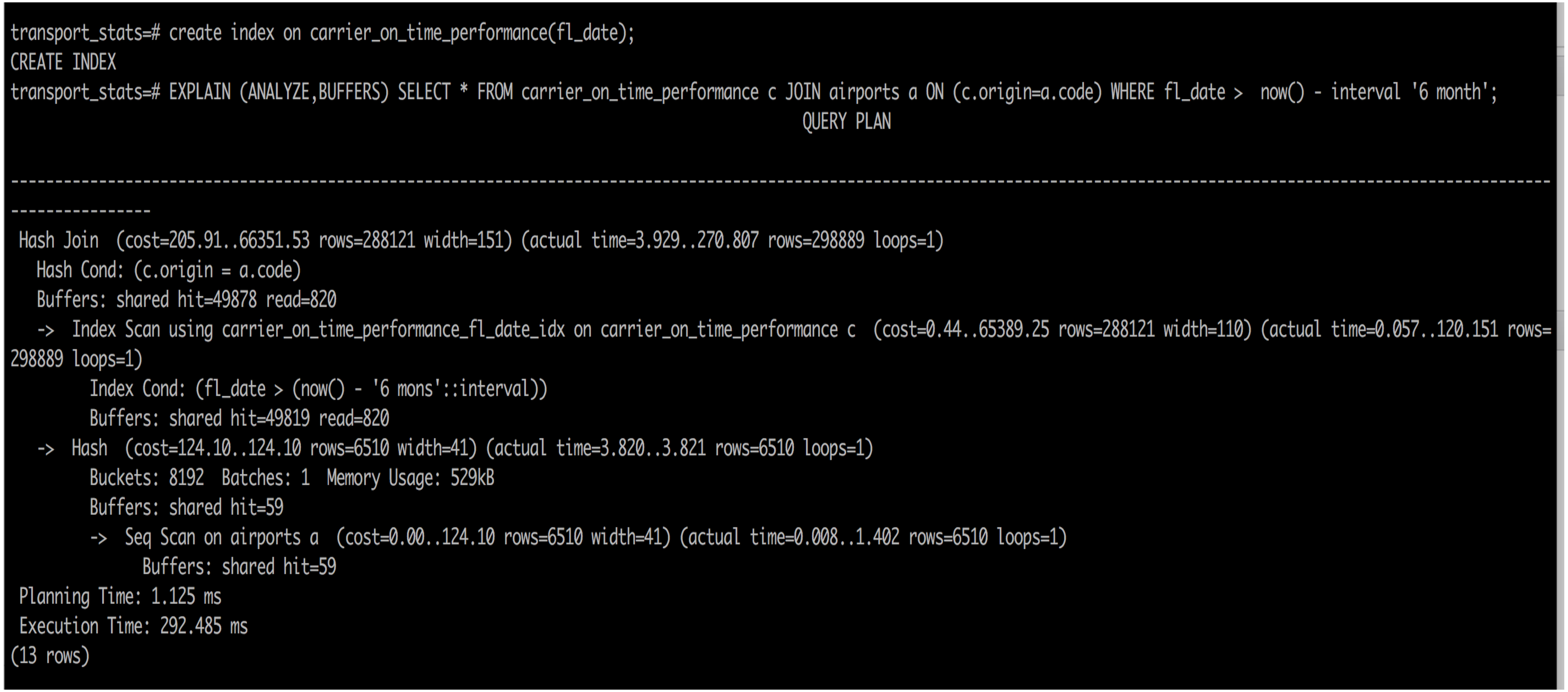
- Create a small hash table from large table
  - The resulting hash table has to fit in memory
  - If the table is really small, a nested loop is used
- Different index strategy:
  - Hash joins do not need indexes on the join predicates. They use the hash table instead.
- A hash join uses indexes only if the index supports the independent (any column but join column) predicates
- Reduce the hash table size to improve performance
  - Horizontally (less rows)
  - Vertically (less columns) – avoid SELECT * FROM table
- Hash joins cannot perform joins that have range conditions in the join predicates

HASH JOIN



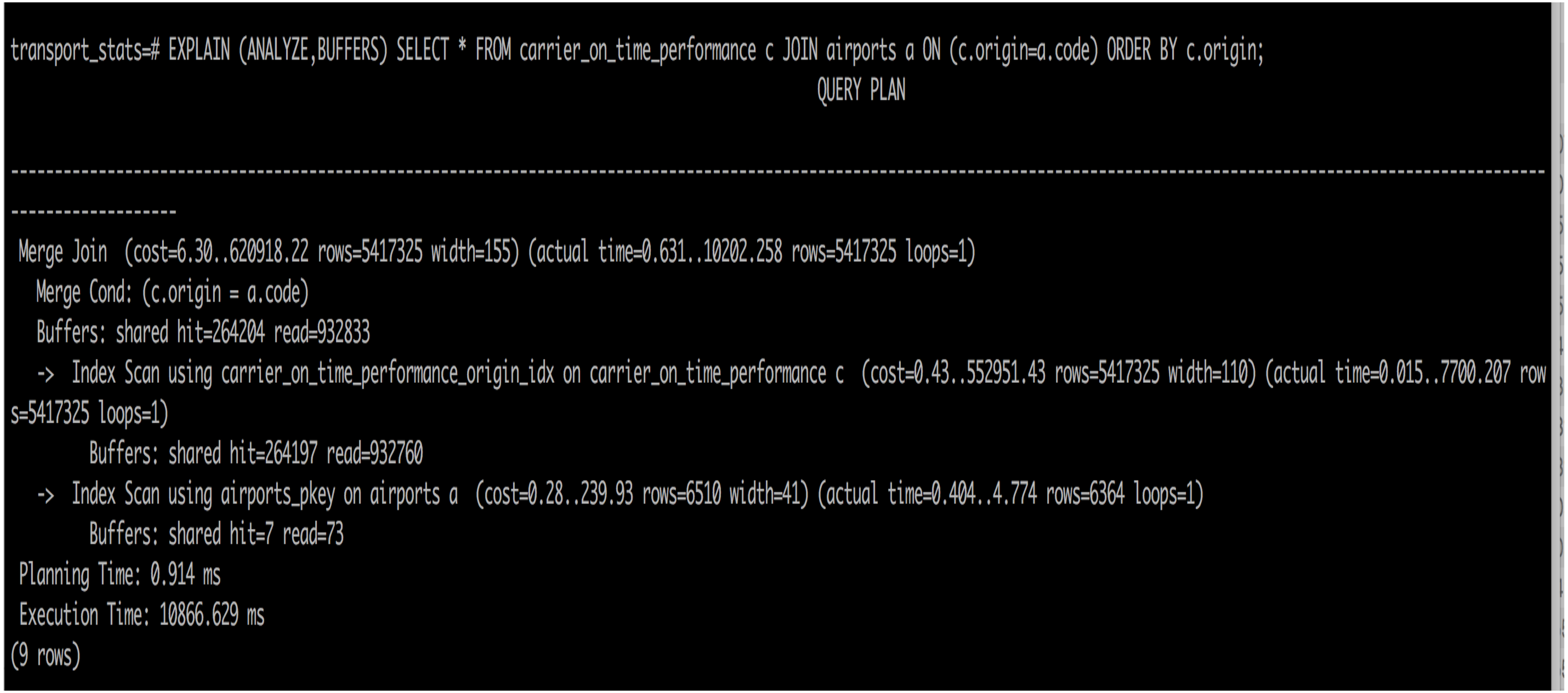
```
transport_stats=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) WHERE fl_date >  now() - interval '6 month';
                                                          QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=205.47..198851.54 rows=288146 width=151) (actual time=1343.887..2172.080 rows=298889 loops=1)
   Hash Cond: (c.origin = a.code)
   Buffers: shared hit=15332 read=87813
   ->  Seq Scan on carrier_on_time_performance c  (cost=0.00..197889.19 rows=288146 width=110) (actual time=1341.824..2040.093 rows=298889 loops=1)
         Filter: (fl_date > (now() - '6 mons'::interval))
         Rows Removed by Filter: 5118436
         Buffers: shared hit=15273 read=87813
   ->  Hash  (cost=124.10..124.10 rows=6510 width=41) (actual time=1.935..1.935 rows=6510 loops=1)
         Buckets: 8192  Batches: 1  Memory Usage: 529kB
         Buffers: shared hit=59
         ->  Seq Scan on airports a  (cost=0.00..124.10 rows=6510 width=41) (actual time=0.007..0.791 rows=6510 loops=1)
               Buffers: shared hit=59
 Planning Time: 1.508 ms
 Execution Time: 2192.945 ms
(14 rows)
```

HASH JOIN



```
transport_stats=# create index on carrier_on_time_performance(fl_date);
CREATE INDEX
transport_stats=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) WHERE fl_date >  now() - interval '6 month';
                                                                                                   QUERY PLAN

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------
 Hash Join  (cost=205.91..66351.53 rows=288121 width=151) (actual time=3.929..270.807 rows=298889 loops=1)
   Hash Cond: (c.origin = a.code)
   Buffers: shared hit=49878 read=820
   ->  Index Scan using carrier_on_time_performance_fl_date_idx on carrier_on_time_performance c  (cost=0.44..65389.25 rows=288121 width=110) (actual time=0.057..120.151 rows=
298889 loops=1)
         Index Cond: (fl_date > (now() - '6 mons'::interval))
         Buffers: shared hit=49819 read=820
   ->  Hash  (cost=124.10..124.10 rows=6510 width=41) (actual time=3.820..3.821 rows=6510 loops=1)
         Buckets: 8192  Batches: 1  Memory Usage: 529kB
         Buffers: shared hit=59
         ->  Seq Scan on airports a  (cost=0.00..124.10 rows=6510 width=41) (actual time=0.008..1.402 rows=6510 loops=1)
               Buffers: shared hit=59
 Planning Time: 1.125 ms
 Execution Time: 292.485 ms
(13 rows)
```

MERGE JOIN



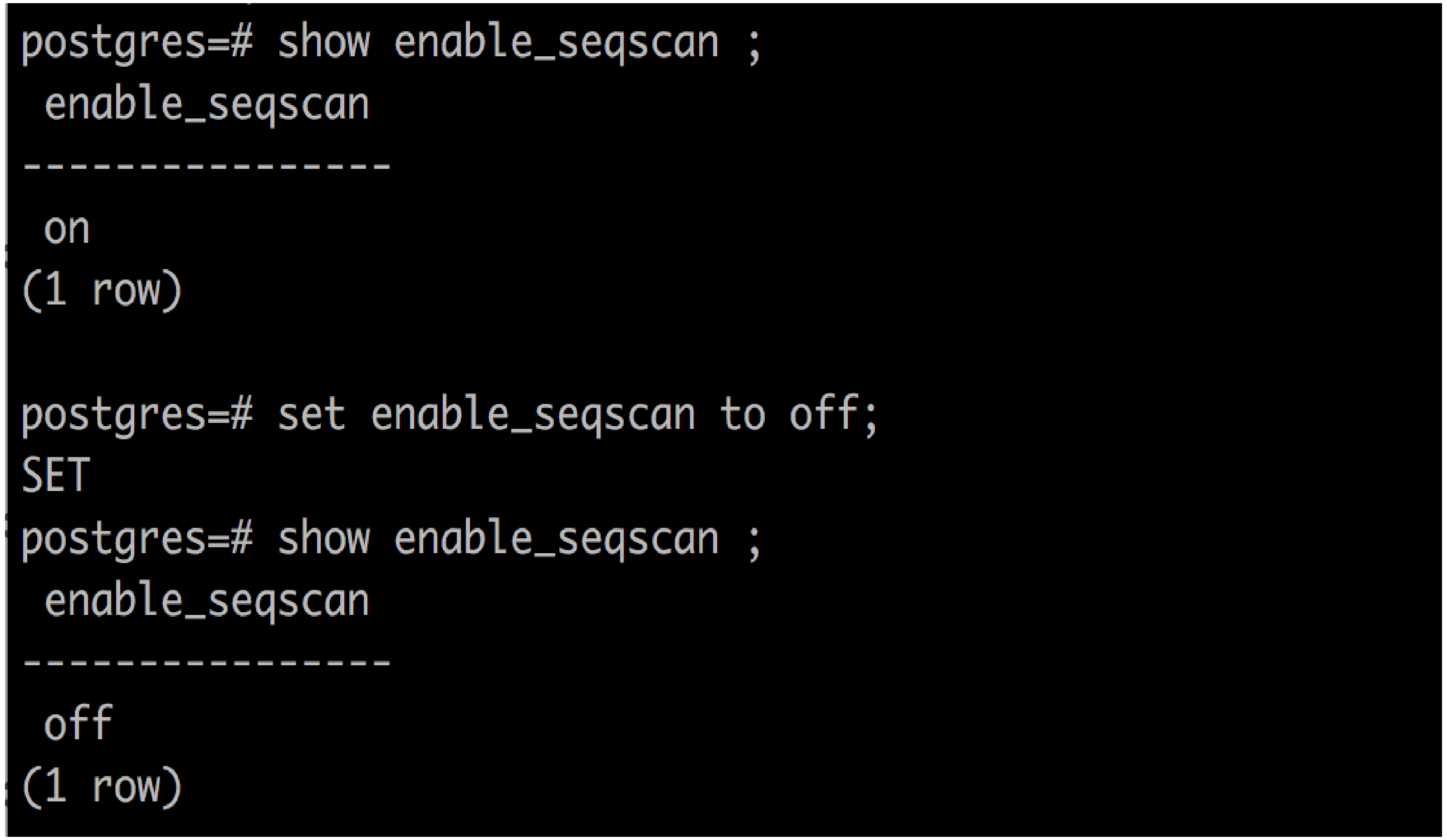
```
transport_stats=# EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM carrier_on_time_performance c JOIN airports a ON (c.origin=a.code) ORDER BY c.origin;
                                                                          QUERY PLAN


----------------------------------------------------------------------------------------------------------------------------------------
--------------
 Merge Join  (cost=6.30..620918.22 rows=5417325 width=155) (actual time=0.631..10202.258 rows=5417325 loops=1)
   Merge Cond: (c.origin = a.code)
   Buffers: shared hit=264204 read=932833
   ->  Index Scan using carrier_on_time_performance_origin_idx on carrier_on_time_performance c  (cost=0.43..552951.43 rows=5417325 width=110) (actual time=0.015..7700.207 row
s=5417325 loops=1)
         Buffers: shared hit=264197 read=932760
   ->  Index Scan using airports_pkey on airports a  (cost=0.28..239.93 rows=6510 width=41) (actual time=0.404..4.774 rows=6364 loops=1)
         Buffers: shared hit=7 read=73
 Planning Time: 0.914 ms
 Execution Time: 10866.629 ms
(9 rows)
```

MERGE JOIN

- The MERGE join combines two sorted lists.
- Both sides of the join must be sorted by the JOIN PREDICATES.
- Similar index strategy like HASH JOIN
- Make sure the index is sorted list

HINTS? - POSTGRESQL CONF PARAMETERS

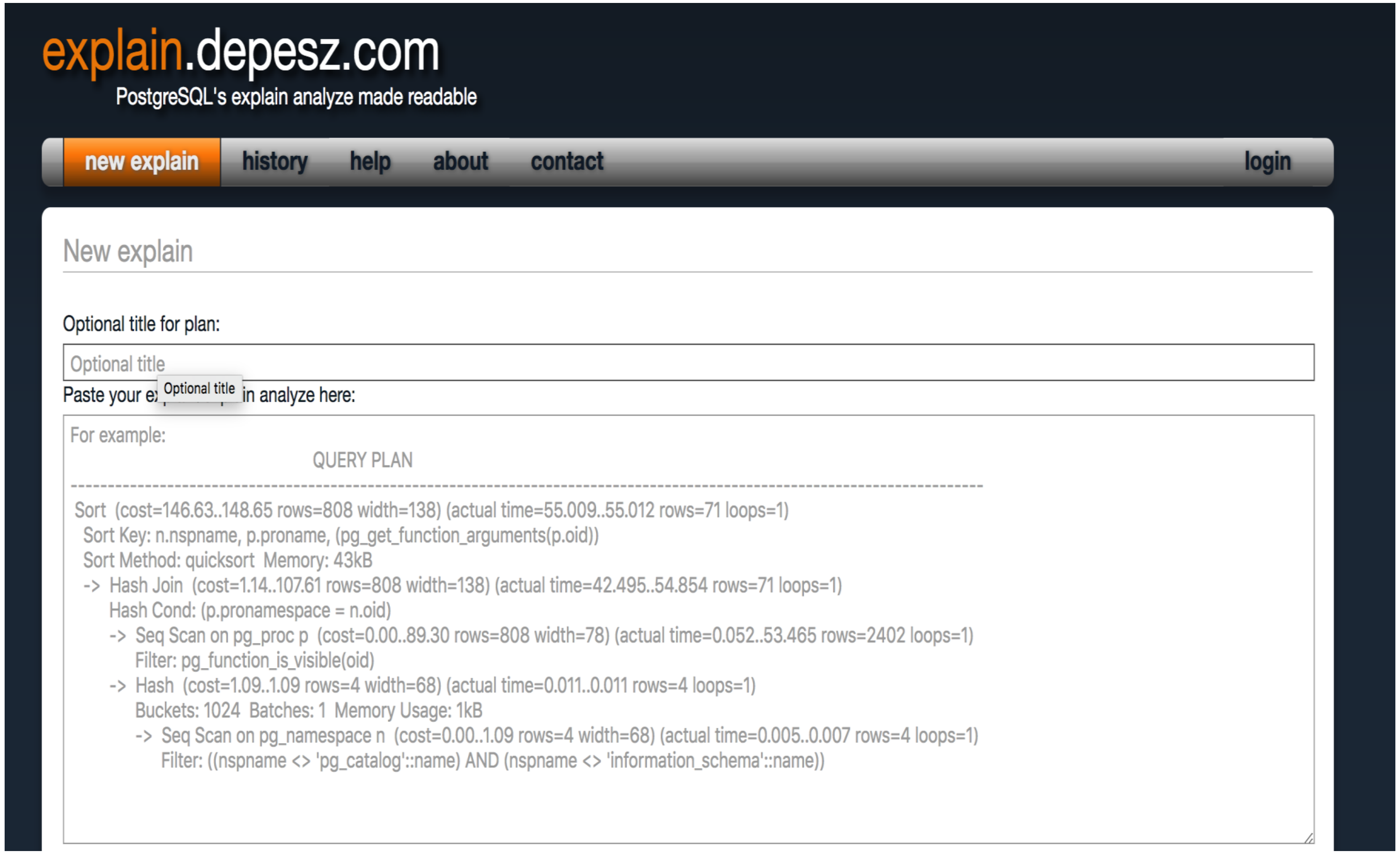
- #enable_bitmapscan = on
- #enable_hashagg = on
- #enable_hashjoin = on
- #enable_indexscan = on
- #enable_indexonlyscan = on
- #enable_material = on
- #enable_mergejoin = on
- #enable_nestloop = on
- #enable_parallel_append = on
- #enable_seqscan = on
- #enable_sort = on



```
postgres=# show enable_seqscan ;
 enable_seqscan
----------------
 on
(1 row)

postgres=# set enable_seqscan to off;
SET
postgres=# show enable_seqscan ;
 enable_seqscan
----------------
 off
(1 row)
```

HTTPS://EXPLAIN.DEPESZ.COM/

- https://explain.depesz.com/



New explain

Optional title for plan:

Optional title

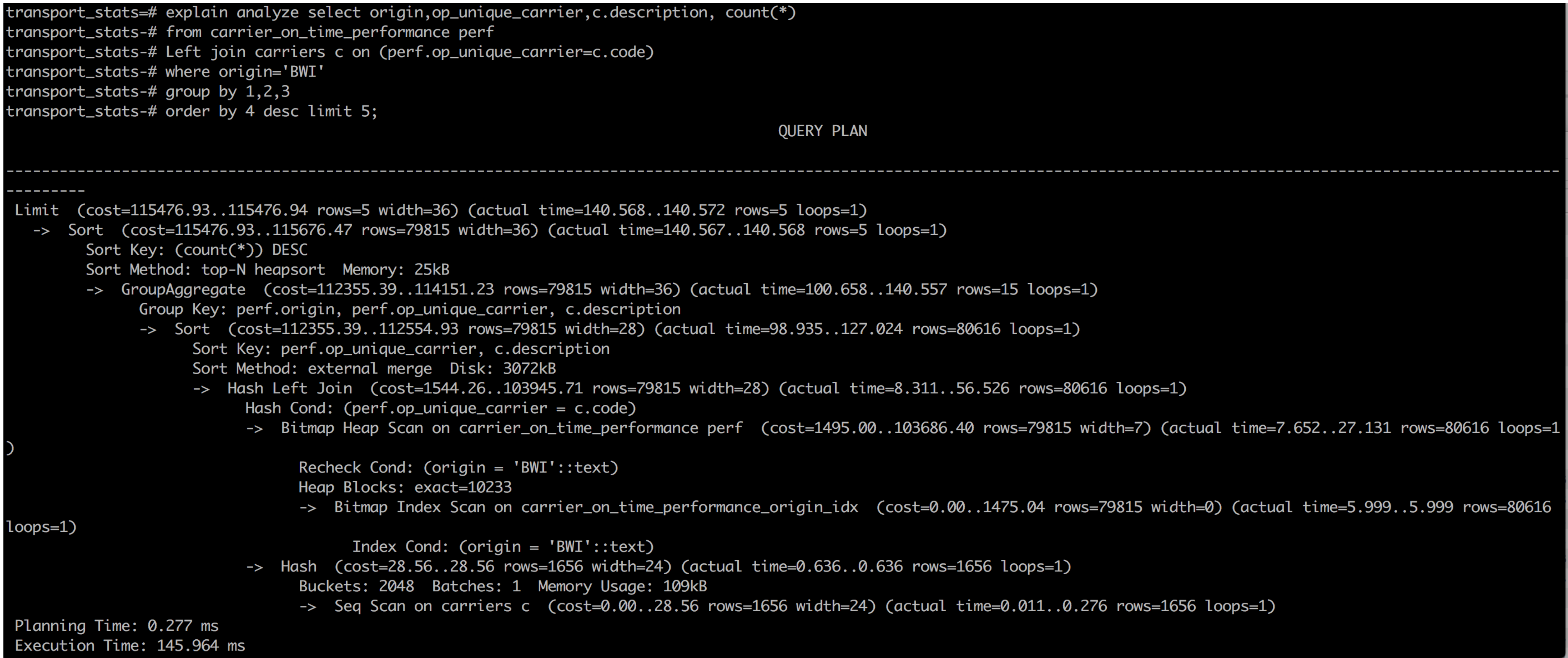
Paste your ex Optional title in analyze here:

```
For example:
                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=146.63..148.65 rows=808 width=138) (actual time=55.009..55.012 rows=71 loops=1)
   Sort Key: n.nspname, p.proname, (pg_get_function_arguments(p.oid))
   Sort Method: quicksort  Memory: 43kB
   ->  Hash Join  (cost=1.14..107.61 rows=808 width=138) (actual time=42.495..54.854 rows=71 loops=1)
         Hash Cond: (p.pronamespace = n.oid)
         ->  Seq Scan on pg_proc p  (cost=0.00..89.30 rows=808 width=78) (actual time=0.052..53.465 rows=2402 loops=1)
               Filter: pg_function_is_visible(oid)
         ->  Hash  (cost=1.09..1.09 rows=4 width=68) (actual time=0.011..0.011 rows=4 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 1kB
               ->  Seq Scan on pg_namespace n  (cost=0.00..1.09 rows=4 width=68) (actual time=0.005..0.007 rows=4 loops=1)
                     Filter: ((nspname <> 'pg_catalog'::name) AND (nspname <> 'information_schema'::name))
```

LET’S PRACTICE

- Find top 5 best performant carriers departing from BWI airport
- Find top 5 best performance carries arriving to BWI

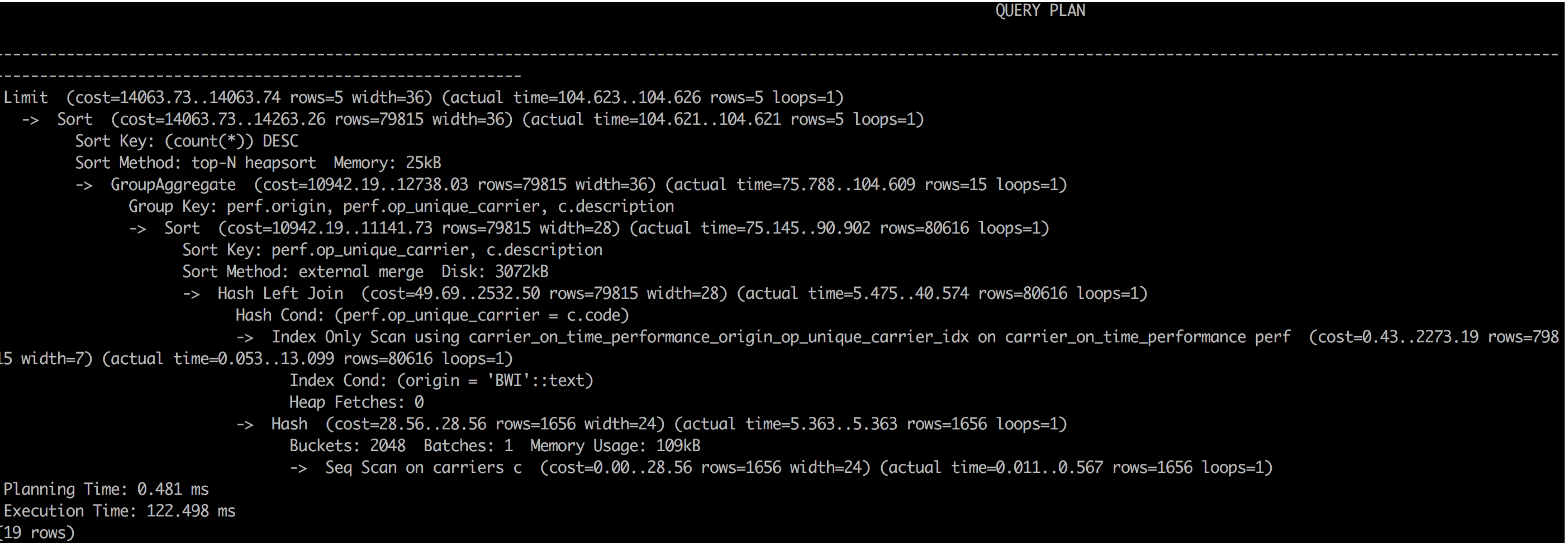
DEPARTING FROM BWI



```
transport_stats=# explain analyze select origin,op_unique_carrier,c.description, count(*)
transport_stats-# from carrier_on_time_performance perf
transport_stats-# Left join carriers c on (perf.op_unique_carrier=c.code)
transport_stats-# where origin='BWI'
transport_stats-# group by 1,2,3
transport_stats-# order by 4 desc limit 5;
                                                                              QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------
 Limit  (cost=115476.93..115476.94 rows=5 width=36) (actual time=140.568..140.572 rows=5 loops=1)
   ->  Sort  (cost=115476.93..115676.47 rows=79815 width=36) (actual time=140.567..140.568 rows=5 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  GroupAggregate  (cost=112355.39..114151.23 rows=79815 width=36) (actual time=100.658..140.557 rows=15 loops=1)
               Group Key: perf.origin, perf.op_unique_carrier, c.description
               ->  Sort  (cost=112355.39..112554.93 rows=79815 width=28) (actual time=98.935..127.024 rows=80616 loops=1)
                     Sort Key: perf.op_unique_carrier, c.description
                     Sort Method: external merge  Disk: 3072kB
                     ->  Hash Left Join  (cost=1544.26..103945.71 rows=79815 width=28) (actual time=8.311..56.526 rows=80616 loops=1)
                           Hash Cond: (perf.op_unique_carrier = c.code)
                           ->  Bitmap Heap Scan on carrier_on_time_performance perf  (cost=1495.00..103686.40 rows=79815 width=7) (actual time=7.652..27.131 rows=80616 loops=1
)
                                 Recheck Cond: (origin = 'BWI'::text)
                                 Heap Blocks: exact=10233
                                 ->  Bitmap Index Scan on carrier_on_time_performance_origin_idx  (cost=0.00..1475.04 rows=79815 width=0) (actual time=5.999..5.999 rows=80616
loops=1)
                                       Index Cond: (origin = 'BWI'::text)
                           ->  Hash  (cost=28.56..28.56 rows=1656 width=24) (actual time=0.636..0.636 rows=1656 loops=1)
                                 Buckets: 2048  Batches: 1  Memory Usage: 109kB
                                 ->  Seq Scan on carriers c  (cost=0.00..28.56 rows=1656 width=24) (actual time=0.011..0.276 rows=1656 loops=1)
 Planning Time: 0.277 ms
 Execution Time: 145.964 ms
```

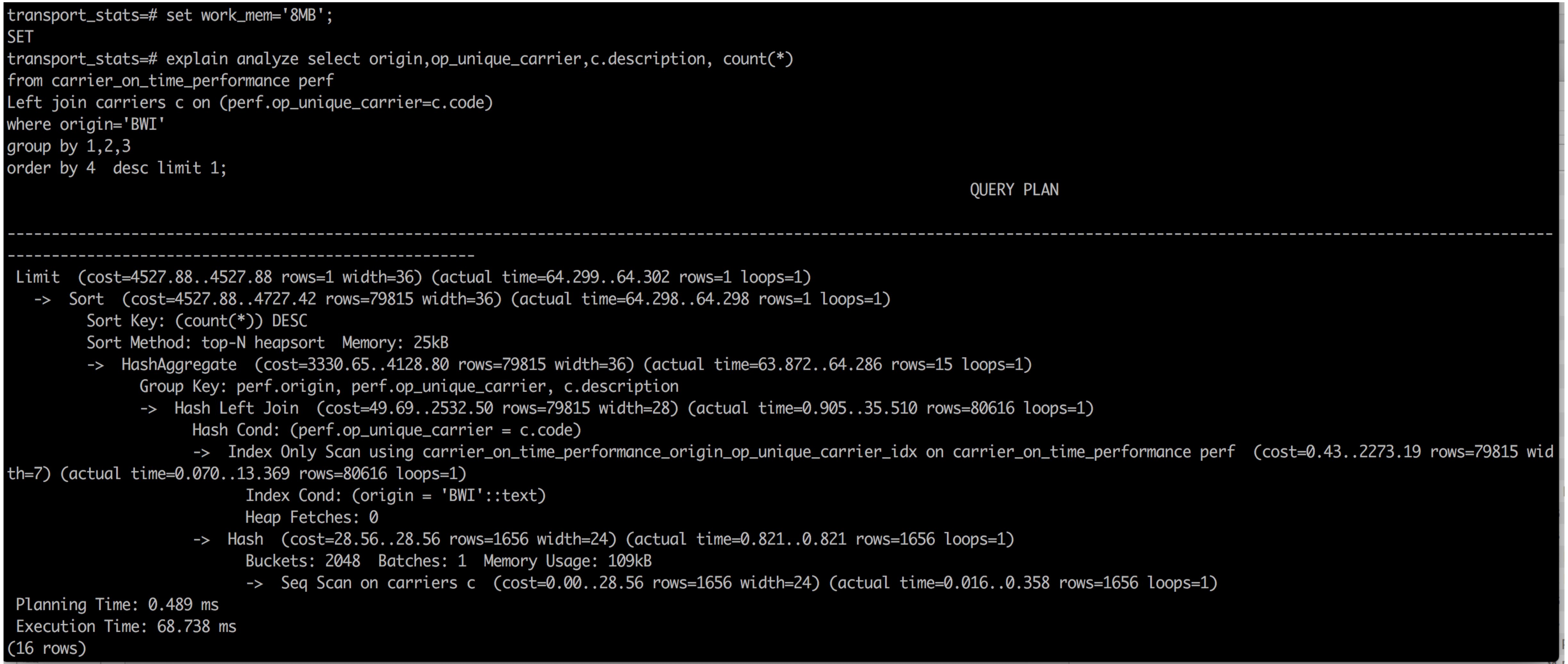
QUERY STATS AFTER ADDING INDEX

- transport_stats=# create index on carrier_on_time_performance(origin,op_unique_carrier);



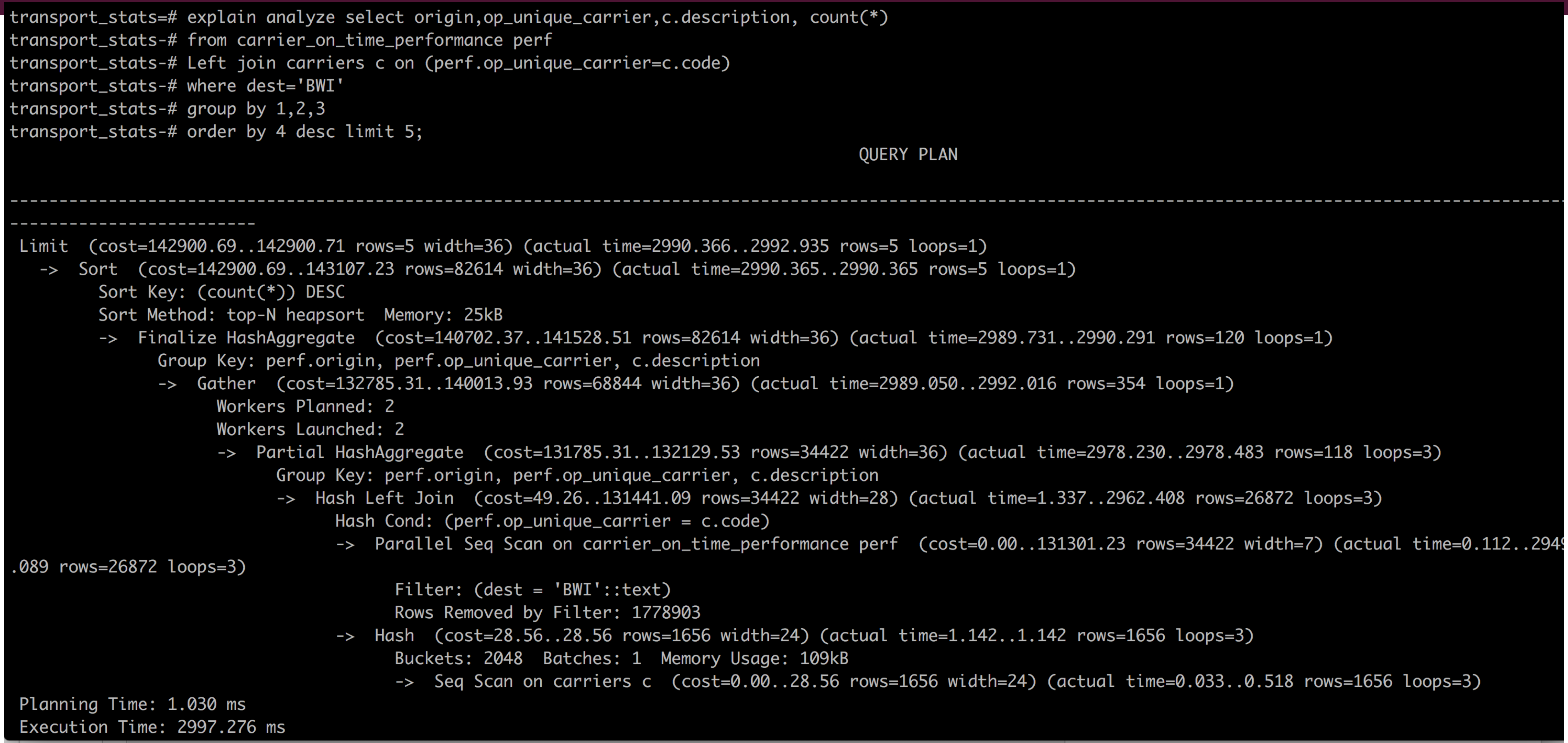
```
                                                                                                                QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------
 Limit  (cost=14063.73..14063.74 rows=5 width=36) (actual time=104.623..104.626 rows=5 loops=1)
   ->  Sort  (cost=14063.73..14263.26 rows=79815 width=36) (actual time=104.621..104.621 rows=5 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  GroupAggregate  (cost=10942.19..12738.03 rows=79815 width=36) (actual time=75.788..104.609 rows=15 loops=1)
               Group Key: perf.origin, perf.op_unique_carrier, c.description
               ->  Sort  (cost=10942.19..11141.73 rows=79815 width=28) (actual time=75.145..90.902 rows=80616 loops=1)
                     Sort Key: perf.op_unique_carrier, c.description
                     Sort Method: external merge  Disk: 3072kB
                     ->  Hash Left Join  (cost=49.69..2532.50 rows=79815 width=28) (actual time=5.475..40.574 rows=80616 loops=1)
                           Hash Cond: (perf.op_unique_carrier = c.code)
                           ->  Index Only Scan using carrier_on_time_performance_origin_op_unique_carrier_idx on carrier_on_time_performance perf  (cost=0.43..2273.19 rows=798
15 width=7) (actual time=0.053..13.099 rows=80616 loops=1)
                                 Index Cond: (origin = 'BWI'::text)
                                 Heap Fetches: 0
                           ->  Hash  (cost=28.56..28.56 rows=1656 width=24) (actual time=5.363..5.363 rows=1656 loops=1)
                                 Buckets: 2048  Batches: 1  Memory Usage: 109kB
                                 ->  Seq Scan on carriers c  (cost=0.00..28.56 rows=1656 width=24) (actual time=0.011..0.567 rows=1656 loops=1)
 Planning Time: 0.481 ms
 Execution Time: 122.498 ms
(19 rows)
```

REMOVE DISK SORT?



```
transport_stats=# set work_mem='8MB';
SET
transport_stats=# explain analyze select origin,op_unique_carrier,c.description, count(*)
from carrier_on_time_performance perf
Left join carriers c on (perf.op_unique_carrier=c.code)
where origin='BWI'
group by 1,2,3
order by 4  desc limit 1;
                                                                                                        QUERY PLAN

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------
 Limit  (cost=4527.88..4527.88 rows=1 width=36) (actual time=64.299..64.302 rows=1 loops=1)
   ->  Sort  (cost=4527.88..4727.42 rows=79815 width=36) (actual time=64.298..64.298 rows=1 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=3330.65..4128.80 rows=79815 width=36) (actual time=63.872..64.286 rows=15 loops=1)
               Group Key: perf.origin, perf.op_unique_carrier, c.description
               ->  Hash Left Join  (cost=49.69..2532.50 rows=79815 width=28) (actual time=0.905..35.510 rows=80616 loops=1)
                     Hash Cond: (perf.op_unique_carrier = c.code)
                     ->  Index Only Scan using carrier_on_time_performance_origin_op_unique_carrier_idx on carrier_on_time_performance perf  (cost=0.43..2273.19 rows=79815 wid
th=7) (actual time=0.070..13.369 rows=80616 loops=1)
                           Index Cond: (origin = 'BWI'::text)
                           Heap Fetches: 0
                     ->  Hash  (cost=28.56..28.56 rows=1656 width=24) (actual time=0.821..0.821 rows=1656 loops=1)
                           Buckets: 2048  Batches: 1  Memory Usage: 109kB
                           ->  Seq Scan on carriers c  (cost=0.00..28.56 rows=1656 width=24) (actual time=0.016..0.358 rows=1656 loops=1)
 Planning Time: 0.489 ms
 Execution Time: 68.738 ms
(16 rows)
```

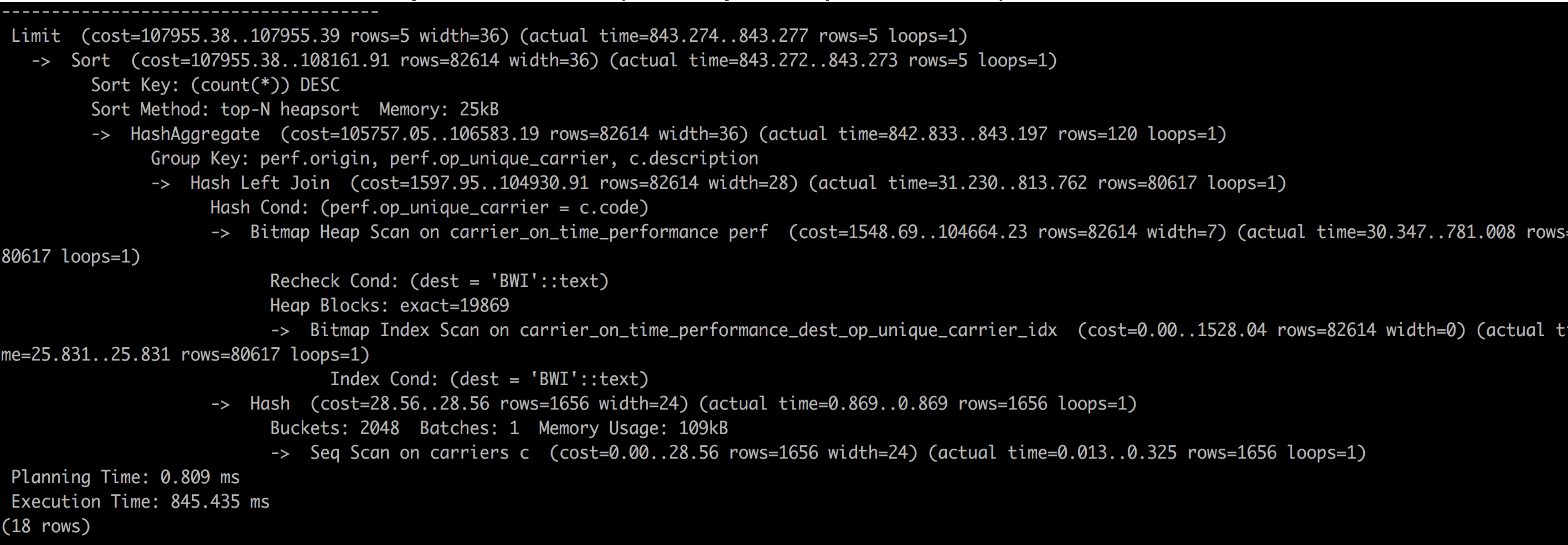
FIND TOP 5 BEST PERFORMANCE CARRIES ARRIVING TO BWI



```
transport_stats=# explain analyze select origin,op_unique_carrier,c.description, count(*)
transport_stats-# from carrier_on_time_performance perf
transport_stats-# Left join carriers c on (perf.op_unique_carrier=c.code)
transport_stats-# where dest='BWI'
transport_stats-# group by 1,2,3
transport_stats-# order by 4 desc limit 5;
                                                                                      QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------
 Limit  (cost=142900.69..142900.71 rows=5 width=36) (actual time=2990.366..2992.935 rows=5 loops=1)
   ->  Sort  (cost=142900.69..143107.23 rows=82614 width=36) (actual time=2990.365..2990.365 rows=5 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Finalize HashAggregate  (cost=140702.37..141528.51 rows=82614 width=36) (actual time=2989.731..2990.291 rows=120 loops=1)
               Group Key: perf.origin, perf.op_unique_carrier, c.description
               ->  Gather  (cost=132785.31..140013.93 rows=68844 width=36) (actual time=2989.050..2992.016 rows=354 loops=1)
                     Workers Planned: 2
                     Workers Launched: 2
                     ->  Partial HashAggregate  (cost=131785.31..132129.53 rows=34422 width=36) (actual time=2978.230..2978.483 rows=118 loops=3)
                           Group Key: perf.origin, perf.op_unique_carrier, c.description
                           ->  Hash Left Join  (cost=49.26..131441.09 rows=34422 width=28) (actual time=1.337..2962.408 rows=26872 loops=3)
                                 Hash Cond: (perf.op_unique_carrier = c.code)
                                 ->  Parallel Seq Scan on carrier_on_time_performance perf  (cost=0.00..131301.23 rows=34422 width=7) (actual time=0.112..294
.089 rows=26872 loops=3)
                                       Filter: (dest = 'BWI'::text)
                                       Rows Removed by Filter: 1778903
                                 ->  Hash  (cost=28.56..28.56 rows=1656 width=24) (actual time=1.142..1.142 rows=1656 loops=3)
                                       Buckets: 2048  Batches: 1  Memory Usage: 109kB
                                       ->  Seq Scan on carriers c  (cost=0.00..28.56 rows=1656 width=24) (actual time=0.033..0.518 rows=1656 loops=3)
 Planning Time: 1.030 ms
 Execution Time: 2997.276 ms
```

ADD INDEX

- transport_stats=# create index on carrier_on_time_performance(dest,op_unique_carrier);



```
-------------------------------------
 Limit  (cost=107955.38..107955.39 rows=5 width=36) (actual time=843.274..843.277 rows=5 loops=1)
   ->  Sort  (cost=107955.38..108161.91 rows=82614 width=36) (actual time=843.272..843.273 rows=5 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=105757.05..106583.19 rows=82614 width=36) (actual time=842.833..843.197 rows=120 loops=1)
               Group Key: perf.origin, perf.op_unique_carrier, c.description
               ->  Hash Left Join  (cost=1597.95..104930.91 rows=82614 width=28) (actual time=31.230..813.762 rows=80617 loops=1)
                     Hash Cond: (perf.op_unique_carrier = c.code)
                     ->  Bitmap Heap Scan on carrier_on_time_performance perf  (cost=1548.69..104664.23 rows=82614 width=7) (actual time=30.347..781.008 rows=
80617 loops=1)
                           Recheck Cond: (dest = 'BWI'::text)
                           Heap Blocks: exact=19869
                           ->  Bitmap Index Scan on carrier_on_time_performance_dest_op_unique_carrier_idx  (cost=0.00..1528.04 rows=82614 width=0) (actual t
me=25.831..25.831 rows=80617 loops=1)
                                 Index Cond: (dest = 'BWI'::text)
                     ->  Hash  (cost=28.56..28.56 rows=1656 width=24) (actual time=0.869..0.869 rows=1656 loops=1)
                           Buckets: 2048  Batches: 1  Memory Usage: 109kB
                           ->  Seq Scan on carriers c  (cost=0.00..28.56 rows=1656 width=24) (actual time=0.013..0.325 rows=1656 loops=1)
 Planning Time: 0.809 ms
 Execution Time: 845.435 ms
(18 rows)
```

RESULTS

```
transport_stats=#  select origin,op_unique_carrier,c.description, count(*)
from carrier_on_time_performance perf
Left join carriers c on (perf.op_unique_carrier=c.code)
where dest='BWI'
group by 1,2,3
order by 4 desc limit 5;
 origin | op_unique_carrier |      description       | count
--------+-------------------+------------------------+-------
 ATL    | DL                | Delta Air Lines Inc.   |  2885
 MCO    | WN                | Southwest Airlines Co. |  2522
 FLL    | WN                | Southwest Airlines Co. |  2494
 BOS    | WN                | Southwest Airlines Co. |  2457
 TPA    | WN                | Southwest Airlines Co. |  1998
(5 rows)
```

TAKE AWAY …

- Keep the statistics updated
  - Keep auto-vacuum turned ON
- Identify slow query postgres logs or pg_stat_statements
- EXPLAIN ANALYZE
- Review cardinality , histograms
- Try out different indices based on JOIN conditions
  - B-tree
  - Partial index
  - Functional index

TAKE AWAY ..

- If possible, try to avoid selecting all columns
- If query is not using index
    - default_statistics_target
    - Play with different session level settings to understand optimizer behavior
        - i.e set enable_seqscan on;

THANK YOU!

- Thanks for attending!
- Looking forward to chat over Slack channel !
- https://postgres-slack.herokuapp.com/

QUESTIONS?

- Any question?
- Future questions:
  - Denish.j.patel@gmail.com
  - Slack channel