

Explaining EXPLAIN:

A Deep Dive into EXPLAIN Plans

SF PgDay | 2020.01.21 | Richard Yen

About Me

- Support Engineer at EnterpriseDB
- Previously a DBA and Web Developer
- Been using PostgreSQL since v. 7.4 (I'll let you do the math)

Why is my query slow?

Let me **EXPLAIN!**

What does **EXPLAIN** do?

What does **EXPLAIN** do?

- Tells you what the query planner chose to do

What does **EXPLAIN** do?

- Tells you what the query planner chose to do
- Tells you which step(s) took the most time

What does **EXPLAIN** do?

- Tells you what the query planner chose to do
- Tells you which step(s) took the most time
- Gives you statistics on how the query was executed

Not a Query Fairy

Not a Query Fairy

- Won't rewrite queries for you

Not a Query Fairy

- Won't rewrite queries for you
- Won't tell you why the query planner made some choice

Not a Query Fairy

- Won't rewrite queries for you
- Won't tell you why the query planner made some choice
- Won't tell you if query performance is being affected by another session

Not a Query Fairy

- Won't rewrite queries for you
- Won't tell you why the query planner made some choice
- Won't tell you if query performance is being affected by another session
- Won't take into account stuff happening outside the database

Not a Query Fairy



- Won't rewrite queries for you
- Won't tell you why the query planner made some choice
- Won't tell you if query performance is being affected by another session
- Won't take into account stuff happening outside the database
- Won't account for external environmental factors (i.e., network latency)

How does it work?

- Table/Index Statistics
 - Stored in `pg_statistic` (don't look there)
 - Can be viewable by looking `pg_stats` (for the adventurous)
 - Refreshed with `ANALYZE` (not to be confused with `EXPLAIN ANALYZE`)
- Configuration
 - `enable_*` parameters
 - `*_cost` parameters

Config parameters

- Cost Parameters

- `cpu_index_tuple_cost`
- `cpu_operator_cost`
- `cpu_tuple_cost`
- `jit_above_cost` *
- `jit_inline_above_cost` *
- `jit_optimize_above_cost` *
- `parallel_setup_cost` *
- `parallel_tuple_cost` *
- `random_page_cost`
- `seq_page_cost`

- Join Parameters

- `enable_bitmapscan`
- `enable_gathermerge`
- `enable_hashjoin`
- `enable_mergejoin`
- `enable_nestloop`
- `enable_partitionwise_join`

- Scan Parameters

- `enable_indexonlyscan`
- `enable_indexscan`
- `enable_seqscan`
- `enable_tidscan`

- Other Parameters

- `enable_hashagg`
- `enable_parallel_append`
- `enable_parallel_hash`
- `enable_partition_pruning`
- `enable_partitionwise_aggregate`
- `enable_material`
- `enable_sort`

Config parameters

- Cost Parameters

- `cpu_index_tuple_cost`
- `cpu_operator_cost`
- `cpu_tuple_cost`
- `jit_above_cost` *
- `jit_inline_above_cost` *
- `jit_optimize_above_cost` *
- `parallel_setup_cost` *
- `parallel_tuple_cost` *
- `random_page_cost`
- `seq_page_cost`

- Join Parameters

- `enable_bitmapscan`
- `enable_gathermerge`
- `enable_hashjoin`
- `enable_mergejoin`
- `enable_nestloop`
- `enable_partitionwise_join`

- Scan Parameters

- `enable_indexonlyscan`
- `enable_indexscan`
- `enable_seqscan`
- `enable_tidscan`

- Other Parameters

- `enable_hashagg`
- `enable_parallel_append`
- `enable_parallel_hash`
- `enable_partition_pruning`
- `enable_partitionwise_aggregate`
- `enable_material`
- `enable_sort`

```
SELECT *  
FROM pg_settings  
WHERE name LIKE '%cost'  
OR name LIKE 'enable%';
```

Config parameters

- Cost Parameters

- cpu_index_tuple_cost
- cpu_operator_cost
- cpu_tuple_cost
- jit_above_cost *
- jit_inline_above_cost *
- jit_optimize_above_cost *
- parallel_setup_cost *
- parallel_tuple_cost *
- random_page_cost
- seq_page_cost

- Join Parameters

- enable_bitmapscan
- enable_gathermerge
- enable_hashjoin
- enable_mergejoin
- enable_nestloop
- enable_partitionwise_join

- Scan Parameters

- enable_indexonlyscan
- enable_indexscan
- enable_seqscan
- enable_tidscan

- Other Parameters

- enable_hashagg
- enable_parallel_append
- enable_parallel_hash
- enable_partition_pruning
- enable_partitionwise_aggregate
- enable_material
- enable_sort

```
SELECT *  
FROM pg_settings  
WHERE name LIKE '%cost'  
OR name LIKE 'enable%';
```

A Simple Example

```
bash $ pgbench -i && psql
```

```
<...>
```

```
postgres=# EXPLAIN SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.00..4141.00 rows=99999 width=461)
```

```
  Join Filter: (a.bid = b.bid)
```

```
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364)
```

```
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97)
```

```
          Filter: (aid < 100000)
```

```
(5 rows)
```

A Simple Example

```
bash $ pgbench -i && psql
```

```
<...>
```

```
postgres=# EXPLAIN SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.00..4141.00 rows=99999 width=461)
```

```
  Join Filter: (a.bid = b.bid)
```

```
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364)
```

```
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97)
```

```
          Filter: (aid < 100000)
```

???

```
(5 rows)
```

Cost Calculation

```
Nested Loop (cost=0.00..4141.00 rows=99999 width=461)
```

```
Join Filter: (a.bid = b.bid)
```

```
-> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364)
```

```
-> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97)
```

Cost Calculation

```
Nested Loop (cost=0.00..4141.00 rows=99999 width=461)
  Join Filter: (a.bid = b.bid)
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364)
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97)
```

$$\text{cost} = (\# \text{blocks} * \text{seq_page_cost}) + (\# \text{records} * \text{cpu_tuple_cost}) + (\# \text{records} * \text{cpu_filter_cost})$$

```
postgres=# select pg_relation_size('pgbench_accounts');
 pg_relation_size
```

```
-----
      13434880
```

```
block_size      = 8192    (8kB, typical OS)
#blocks         = 1640    (relation_size / block_size)
#records        = 100000
seq_page_cost   = 1       (default)
cpu_tuple_cost  = 0.01    (default)
cpu_filter_cost = 0.0025  (default)
```

$$\text{cost} = (1640 * 1) + (100000 * 0.01) + (100000 * 0.0025) = 2890$$

Cost Calculation

```
Nested Loop (cost=0.00..4141.00 rows=99999 width=461)
  Join Filter: (a.bid = b.bid)
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364)
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97)
```

$$\text{cost} = (\# \text{blocks} * \text{seq_page_cost}) + (\# \text{records} * \text{cpu_tuple_cost}) + (\# \text{records} * \text{cpu_filter_cost})$$

```
postgres=# select pg_relation_size('pgbench_accounts');
 pg_relation_size
```

```
-----
      13434880
```

```
block_size      = 8192    (8kB, typical OS)
#blocks         = 1640    (relation_size / block_size)
#records        = 100000
seq_page_cost   = 1       (default)
cpu_tuple_cost  = 0.01    (default)
cpu_filter_cost = 0.0025  (default)
```

$$\text{cost} = (1640 * 1) + (100000 * 0.01) + (100000 * 0.0025) = 2890$$

A Simple Example (w/ ANALYZE)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE  
a.aid < 100000;
```

QUERY PLAN

```
Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)  
  Join Filter: (a.bid = b.bid)  
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1  
loops=1)  
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752  
rows=99999 loops=1)  
      Filter: (aid < 100000)  
      Rows Removed by Filter: 1  
Planning Time: 0.306 ms  
Execution Time: 61.031 ms  
(8 rows)
```


A Simple Example (w/ ANALYZE)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE  
a.aid < 100000;
```

QUERY PLAN

```
Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)  
  Join Filter: (a.bid = b.bid)  
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1  
loops=1)  
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752  
rows=99999 loops=1)  
      Filter: (aid < 100000)  
      Rows Removed by Filter: 1  
Planning Time: 0.306 ms  
Execution Time: 61.031 ms  
(8 rows)
```

A Simple Example (w/ ANALYZE)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE  
a.aid < 100000;
```

QUERY PLAN

Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)

Join Filter: (a.bid = b.bid)

-> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1 loops=1)

-> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752 rows=99999 loops=1)

Filter: (aid < 100000)

Rows Removed by Filter: 1

Planning Time: 0.306 ms

Execution Time: 61.031 ms

(8 rows)

A Simple Example (w/ ANALYZE & BUFFERS)

```
postgres=# EXPLAIN (BUFFERS, ANALYZE) SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)  
  Join Filter: (a.bid = b.bid)  
  Buffers: shared hit=3 read=1638  
    -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1 loops=1)  
        Buffers: shared hit=1  
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752 rows=99999  
loops=1)  
        Filter: (aid < 100000)  
        Rows Removed by Filter: 1  
        Buffers: shared hit=2 read=1638  
Planning Time: 0.306 ms  
Execution Time: 61.031 ms  
(8 rows)
```

A Simple Example (w/ ANALYZE & BUFFERS)

```
postgres=# EXPLAIN (BUFFERS, ANALYZE) SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

QUERY PLAN

```
-----
Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)
  Join Filter: (a.bid = b.bid)
  Buffers: shared hit=3 read=1638
  -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1 loops=1)
    Buffers: shared hit=1
  -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752 rows=99999 loops=1)
    Filter: (aid < 100000)
    Rows Removed by Filter: 1
    Buffers: shared hit=2 read=1638
Planning Time: 0.306 ms
Execution Time: 61.031 ms
(8 rows)
```

A Simple Example (w/ ANALYZE & BUFFERS)

```
postgres=# EXPLAIN (BUFFERS, ANALYZE) SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

QUERY PLAN

```
-----
Nested Loop (cost=0.00..4141.00 rows=99999 width=461) (actual time=0.039..56.582 rows=99999 loops=1)
  Join Filter: (a.bid = b.bid)
  Buffers: shared hit=3 read=1638
  -> Seq Scan on pgbench_branches b (cost=0.00..1.01 rows=1 width=364) (actual time=0.025..0.026 rows=1 loops=1)
    Buffers: shared hit=1
  -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.008..25.752 rows=99999 loops=1)
    Filter: (aid < 100000)
    Rows Removed by Filter: 1
    Buffers: shared hit=2 read=1638
Planning Time: 0.306 ms
Execution Time: 61.031 ms
(8 rows)
```

A Simple Example (with more rows)

```
postgres=# INSERT INTO pgbench_branches (bid, bbalance, filler) VALUES (generate_series(2,100000),1,'');  
INSERT 0 99999  
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
```

A Simple Example (with more rows)

```
postgres=# INSERT INTO pgbench_branches (bid, bbalance, filler) VALUES (generate_series(2,100000),1,'');
INSERT 0 99999
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
               QUERY PLAN
-----
Hash Join  (cost=1676.90..4830.08 rows=99999 width=461) (actual time=147.289..229.678 rows=99999 loops=1)
  Hash Cond: (a.bid = b.bid)
    -> Seq Scan on pgbench_accounts a  (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.020..26.903 rows=99999 loops=1)
      Filter: (aid < 100000)
      Rows Removed by Filter: 1
    -> Hash  (cost=1656.40..1656.40 rows=1640 width=364) (actual time=63.742..63.743 rows=100000 loops=1)
      Buckets: 32768 (originally 2048)  Batches: 4 (originally 1)  Memory Usage: 3841kB
      -> Seq Scan on pgbench_branches b  (cost=0.00..1656.40 rows=1640 width=364) (actual time=0.014..22.897 rows=100000 loops=1)
Planning Time: 0.278 ms
Execution Time: 234.480 ms
(10 rows)
```

A Simple Example (with more rows)

```
postgres=# INSERT INTO pgbench_branches (bid, bbalance, filler) VALUES (generate_series(2,100000),1,'');
INSERT 0 99999
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100000;
QUERY PLAN
```

```
-----
Hash Join (cost=1676.90..4830.08 rows=99999 width=461) (actual time=147.289..229.678 rows=99999 loops=1)
  Hash Cond: (a.bid = b.bid)
    -> Seq Scan on pgbench_accounts a (cost=0.00..2890.00 rows=99999 width=97) (actual time=0.020..26.903 rows=99999 loops=1)
      Filter: (aid < 100000)
      Rows Removed by Filter: 1
    -> Hash (cost=1656.40..1656.40 rows=1640 width=364) (actual time=63.742..63.743 rows=100000 loops=1)
      Buckets: 32768 (originally 2048) Batches: 4 (originally 1) Memory Usage: 3841kB
      -> Seq Scan on pgbench_branches b (cost=0.00..1656.40 rows=1640 width=364) (actual time=0.014..22.897 rows=100000 loops=1)
Planning Time: 0.278 ms
Execution Time: 234.480 ms
(10 rows)
```


A Simple Example (seeking less rows)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100;
```

A Simple Example (seeking less rows)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100;  
QUERY PLAN
```

```
-----  
Merge Join (cost=14.60..16.13 rows=99 width=194) (actual time=0.094..0.130 rows=99 loops=1)  
  Merge Cond: (b.bid = a.bid)  
    -> Index Scan using pgbench_branches_pkey on pgbench_branches b (cost=0.29..4247.29 rows=100000 width=97) (actual  
time=0.013..0.014 rows=2 loops=1)  
    -> Sort (cost=14.31..14.55 rows=99 width=97) (actual time=0.071..0.079 rows=99 loops=1)  
          Sort Key: a.bid  
          Sort Method: quicksort  Memory: 38kB  
          -> Index Scan using pgbench_accounts_pkey on pgbench_accounts a (cost=0.29..11.03 rows=99 width=97) (actual  
time=0.010..0.033 rows=99 loops=1)  
                Index Cond: (aid < 100)  
Planning Time: 0.931 ms  
Execution Time: 0.205 ms  
(10 rows)
```

A Simple Example (seeking less rows)

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts a JOIN pgbench_branches b ON (a.bid=b.bid) WHERE a.aid < 100;  
QUERY PLAN
```

```
-----  
Merge Join (cost=14.60..16.13 rows=99 width=194) (actual time=0.094..0.130 rows=99 loops=1)  
  Merge Cond: (b.bid = a.bid)  
    -> Index Scan using pgbench_branches_pkey on pgbench_branches b (cost=0.29..4247.29 rows=100000 width=97) (actual  
time=0.013..0.014 rows=2 loops=1)  
    -> Sort (cost=14.31..14.55 rows=99 width=97) (actual time=0.071..0.079 rows=99 loops=1)  
          Sort Key: a.bid  
          Sort Method: quicksort  Memory: 38kB  
    -> Index Scan using pgbench_accounts_pkey on pgbench_accounts a (cost=0.29..11.03 rows=99 width=97) (actual  
time=0.010..0.033 rows=99 loops=1)  
          Index Cond: (aid < 100)  
Planning Time: 0.931 ms  
Execution Time: 0.205 ms  
(10 rows)
```

A Word on Joins

A Word on Joins

- Nested Loops
 - For each row in outer table, scan for matching rows in the inner table
 - Fast to start, best for small tables

A Word on Joins

- Nested Loops
 - For each row in outer table, scan for matching rows in the inner table
 - Fast to start, best for small tables
- Merge Join
 - Zipper-operation on `_sorted_` data sets
 - Good for large tables
 - High startup cost if additional sort is required

A Word on Joins

- Nested Loops
 - For each row in outer table, scan for matching rows in the inner table
 - Fast to start, best for small tables
- Merge Join
 - Zipper-operation on `_sorted_` data sets
 - Good for large tables
 - High startup cost if additional sort is required
- Hash Join
 - Build hash of inner table values, scan outer table for matches
 - Only usable for equality conditions
 - High startup cost, but fast execution

Scan Improvements

```
postgres=# UPDATE pgbench_accounts SET bid = aid;  
UPDATE 100000
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts WHERE bid = 1;  
QUERY PLAN
```

```
Seq Scan on pgbench_accounts (cost=0.00..5778.24 rows=199939 width=97) (actual time=19.322..45.161 rows=1 loops=1)  
  Filter: (bid = 1)  
  Rows Removed by Filter: 99999  
Planning Time: 0.101 ms  
Execution Time: 45.191 ms  
(5 rows)
```

```
postgres=# CREATE INDEX pgba_bid_idx ON pgbench_accounts (bid);  
CREATE INDEX
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts WHERE bid = 1;  
QUERY PLAN
```

```
Index Scan using pgba_bid_idx on pgbench_accounts (cost=0.29..8.31 rows=1 width=97) (actual time=0.076..0.077 rows=1 loops=1)  
  Index Cond: (bid = 1)  
Planning Time: 0.312 ms  
Execution Time: 0.119 ms  
(4 rows)
```


Scan Improvements

```
postgres=# UPDATE pgbench_accounts SET bid = aid;  
UPDATE 100000
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts WHERE bid = 1;  
QUERY PLAN
```

```
Seq Scan on pgbench_accounts (cost=0.00..5778.24 rows=199939 width=97) (actual time=19.322..45.161 rows=1 loops=1)  
  Filter: (bid = 1)  
  Rows Removed by Filter: 99999  
Planning Time: 0.101 ms  
Execution Time: 45.191 ms  
(5 rows)
```

```
postgres=# CREATE INDEX pgba_bid_idx ON pgbench_accounts (bid);  
CREATE INDEX
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts WHERE bid = 1;  
QUERY PLAN
```

```
Index Scan using pgba_bid_idx on pgbench_accounts (cost=0.29..8.31 rows=1 width=97) (actual time=0.076..0.077 rows=1 loops=1)  
  Index Cond: (bid = 1)  
Planning Time: 0.212 ms  
Execution Time: 0.119 ms  
(4 rows)
```

The Fastest Scan

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts where aid < 1000;
```

```
QUERY PLAN
```

```
-----  
Index Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.43..47.87 rows=939 width=97) (actual time=0.371..0.721  
rows=999 loops=1)
```

```
Index Cond: (aid < 1000)
```

```
Planning Time: 0.226 ms
```

```
Execution Time: 0.815 ms
```

```
(4 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT aid FROM pgbench_accounts where aid < 1000;
```

```
QUERY PLAN
```

```
-----  
Index Only Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.43..28.87 rows=939 width=4) (actual time=0.022..0.169  
rows=999 loops=1)
```

```
Index Cond: (aid < 1000)
```

```
Heap Fetches: 0
```

```
Planning Time: 0.161 ms
```

```
Execution Time: 0.237 ms
```

```
(5 rows)
```

The Fastest Scan

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_accounts where aid < 1000;  
QUERY PLAN
```

```
Index Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.43..47.87 rows=939 width=97) (actual time=0.371..0.721  
rows=999 loops=1)  
  Index Cond: (aid < 1000)  
Planning Time: 0.226 ms  
Execution Time: 0.815 ms  
(4 rows)
```

```
postgres=# EXPLAIN ANALYZE SELECT aid FROM pgbench_accounts where aid < 1000;  
QUERY PLAN
```

```
Index Only Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.43..28.87 rows=939 width=4) (actual time=0.022..0.169  
rows=999 loops=1)  
  Index Cond: (aid < 1000)  
  Heap Fetches: 0  
Planning Time: 0.161 ms  
Execution Time: 0.237 ms  
(5 rows)
```

Index Scan Costs

```
postgres=# show random_page_cost;  
random_page_cost
```

```
-----
```

```
4  
(1 row)
```

```
postgres=# EXPLAIN SELECT * FROM pgbench_accounts WHERE aid < 1000;  
QUERY PLAN
```

```
-----
```

```
Index Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.29..50.30 rows=1029 width=97)  
Index Cond: (aid < 1000)
```

```
(2 rows)
```

```
postgres=# SET random_page_cost = 100;
```

```
postgres=# EXPLAIN SELECT * FROM pgbench_accounts WHERE aid < 1000;  
QUERY PLAN
```

```
-----
```

```
Index Scan using pgbench_accounts_pkey on pgbench_accounts (cost=0.29..434.30 rows=1029 width=97)  
Index Cond: (aid < 1000)
```

```
(2 rows)
```

```
postgres=# SET random_page_cost = 1000;
```

```
postgres=# EXPLAIN SELECT * FROM pgbench_accounts WHERE aid < 1000;  
QUERY PLAN
```

```
-----
```

```
Seq Scan on pgbench_accounts (cost=0.00..2890.00 rows=1029 width=97)  
Filter: (aid < 1000)
```

```
(2 rows)
```

Scan Types

- Sequential Scan
 - Scan the whole table
- Index Scan
 - Scan all/some rows in index; look up rows in heap
 - Causes random seek
- Index Only Scan
 - Scan all/some rows in index; no need to look up rows in heap
- Bitmap Heap Scan
 - Scan index, building a bitmap of pages to visit; look up only relevant pages in heap for rows

What Could Go Wrong?

What Could Go Wrong?

- Bad statistics

What Could Go Wrong?

- Bad statistics
- Insufficient Memory Allocation

What Could Go Wrong?

- Bad statistics
- Insufficient Memory Allocation
- Index Definition Mismatch

What Could Go Wrong?

- Bad statistics
- Insufficient Memory Allocation
- Index Definition Mismatch
- Other Maddening Manifestations 🤔 🤔 🤔

Bad Statistics

```
$ pgbench -T 300 && psql
postgres=# CREATE INDEX foo ON pgbench_history (aid);
CREATE INDEX
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE aid < 100;
```

QUERY PLAN

```
Seq Scan on pgbench_history (cost=0.00..2346.00 rows=35360 width=50) (actual time=0.221..22.912 rows=170 loops=1)
  Filter: (aid < 100)
  Rows Removed by Filter: 159911
Planning Time: 0.610 ms
Execution Time: 24.292 ms
```

(6 rows)

```
postgres=# ANALYZE;
ANALYZE
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE aid < 100;
```

QUERY PLAN

```
Index Scan using foo on pgbench_history (cost=0.42..579.09 rows=153 width=50) (actual time=0.017..1.918 rows=170 loops=1)
  Index Cond: (aid < 100)
Planning Time: 0.167 ms
Execution Time: 3.507 ms
```

(5 rows)

Bad Statistics

```
$ pgbench -T 300 && psql
postgres=# CREATE INDEX foo ON pgbench_history (aid);
CREATE INDEX
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE aid < 100;
```

QUERY PLAN

```
-----
Seq Scan on pgbench_history  (cost=0.00..2346.00 rows=35360 width=50) (actual time=0.221..22.912 rows=170 loops=1)
  Filter: (aid < 100)
  Rows Removed by Filter: 159911
Planning Time: 0.610 ms
Execution Time: 24.292 ms
```

(6 rows)

```
postgres=# ANALYZE;
```

```
ANALYZE
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE aid < 100;
```

QUERY PLAN

```
-----
Index Scan using foo on pgbench_history  (cost=0.42..579.09 rows=153 width=50) (actual time=0.017..1.918 rows=170 loops=1)
  Index Cond: (aid < 100)
Planning Time: 0.167 ms
Execution Time: 3.507 ms
```

(5 rows)

VACUUM and ANALYZE often!

autovacuum will help you with that 👍

CREATE STATISTICS (v. 10+)

```
CREATE STATISTICS [ IF NOT EXISTS ] statistics_name  
    [ ( statistics_kind [, ... ] ) ]  
    ON column_name, column_name [, ...]  
    FROM table_name
```

Insufficient Memory Allocation

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE delta < 0 ORDER BY delta;  
QUERY PLAN
```

```
-----  
Sort  (cost=12225.68..12424.74 rows=79623 width=50) (actual time=1187.391..1763.319 rows=79875 loops=1)  
  Sort Key: delta  
  Sort Method: external merge  Disk: 2664kB  
    -> Seq Scan on pgbench_history  (cost=0.00..3021.01 rows=79623 width=50) (actual time=0.023..593.128 rows=79875 loops=1)  
      Filter: (delta < 0)  
      Rows Removed by Filter: 80206  
Planning Time: 0.082 ms  
Execution Time: 2312.374 ms
```

(8 rows)

```
postgres=# SHOW work_mem ;  
work_mem
```

```
-----  
4MB
```

(1 row)

```
postgres=# SET work_mem = '16 MB';  
SET
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE delta < 0 ORDER BY delta;  
QUERY PLAN
```

```
-----  
Sort  (cost=9502.68..9701.74 rows=79623 width=50) (actual time=1128.871..1662.322 rows=79875 loops=1)  
  Sort Key: delta  
  Sort Method: quicksort  Memory: 9313kB  
    -> Seq Scan on pgbench_history  (cost=0.00..3021.01 rows=79623 width=50) (actual time=0.021..569.691 rows=79875 loops=1)  
      Filter: (delta < 0)  
      Rows Removed by Filter: 80206  
Planning Time: 0.083 ms  
Execution Time: 2187.715 ms
```

(8 rows)

Insufficient Memory Allocation

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE delta < 0 ORDER BY delta;  
QUERY PLAN
```

```
-----  
Sort  (cost=12225.68..12424.74 rows=79623 width=50) (actual time=1187.391..1763.319 rows=79875 loops=1)  
  Sort Key: delta  
  Sort Method: external merge  Disk: 2664kB  
-> Seq Scan on pgbench_history  (cost=0.00..3021.01 rows=79623 width=50) (actual time=0.023..593.128 rows=79875 loops=1)  
  Filter: (delta < 0)  
  Rows Removed by Filter: 80206  
Planning Time: 0.082 ms  
Execution Time: 2312.374 ms
```

(8 rows)

```
postgres=# SHOW work_mem ;  
work_mem
```

```
-----  
4MB
```

(1 row)

```
postgres=# SET work_mem = '16 MB';  
SET
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM pgbench_history WHERE delta < 0 ORDER BY delta;  
QUERY PLAN
```

```
-----  
Sort  (cost=9502.68..9701.74 rows=79623 width=50) (actual time=1128.871..1662.322 rows=79875 loops=1)  
  Sort Key: delta  
  Sort Method: quicksort  Memory: 9313kB  
-> Seq Scan on pgbench_history  (cost=0.00..3021.01 rows=79623 width=50) (actual time=0.021..569.691 rows=79875 loops=1)  
  Filter: (delta < 0)  
  Rows Removed by Filter: 80206  
Planning Time: 0.083 ms  
Execution Time: 2187.715 ms
```

(8 rows)

Index Definition Mismatch

```
postgres=# CREATE INDEX fillertext_idx ON pgbench_history (aid, substring(filler,1,1));
postgres=# EXPLAIN SELECT * FROM pgbench_history WHERE aid = 10000 AND left(filler,1) = 'b';
               QUERY PLAN
```

```
-----
Bitmap Heap Scan on pgbench_history  (cost=4.44..12.26 rows=1 width=47)
  Recheck Cond: (aid = 10000)
  Filter: ("left"((filler)::text, 1) = 'b'::text)
  Heap Blocks: exact=2
->  Bitmap Index Scan on fillertext_idx  (cost=0.00..4.43 rows=2 width=0)
      Index Cond: (aid = 10000)
```

(6 rows)

```
postgres=# EXPLAIN SELECT * FROM pgbench_history WHERE aid = 10000 AND substring(lower(filler),1,1) = 'b';
               QUERY PLAN
```

```
-----
Bitmap Heap Scan on pgbench_history  (cost=4.44..12.26 rows=1 width=47)
  Recheck Cond: (aid = 10000)
  Filter: ("substring"(lower((filler)::text), 1, 1) = 'b'::text)
  Heap Blocks: exact=2
->  Bitmap Index Scan on fillertext_idx  (cost=0.00..4.43 rows=2 width=0)
      Index Cond: (aid = 10000)
```

(6 rows)

Index Definition Mismatch

```
postgres=# EXPLAIN SELECT * FROM pgbench_history WHERE aid = 10000 AND substring(filler,1,1) = 'b';
```

```
QUERY PLAN
```

```
-----  
Index Scan using fillertext_idx on pgbench_history  (cost=0.42..8.44 rows=1 width=47)  
  Index Cond: ((aid = 10000) AND ("substring"((filler)::text, 1, 1) = 'b'::text))  
(2 rows)
```

Other Maddening Manifestations

Other Maddening Manifestations

- Join order

Other Maddening Manifestations

- Join order
- Prepared Statements

Other Maddening Manifestations

- Join order
- Prepared Statements
 - `PREPARE foo AS SELECT * FROM pgbench_accounts WHERE aid = $1;`

Other Maddening Manifestations

- Join order
- Prepared Statements
 - `PREPARE foo AS SELECT * FROM pgbench_accounts WHERE aid = $1;`
 - First 5 executions use a custom plan (taking into account `$1`)

Other Maddening Manifestations

- Join order
- Prepared Statements
 - `PREPARE foo AS SELECT * FROM pgbench_accounts WHERE aid = $1;`
 - First 5 executions use a custom plan (taking into account `$1`)
 - After that, a generic plan is used

Other Maddening Manifestations

- Join order
- Prepared Statements
 - `PREPARE foo AS SELECT * FROM pgbench_accounts WHERE aid = $1;`
 - First 5 executions use a custom plan (taking into account `$1`)
 - After that, a generic plan is used
- `from_collapse_limit / join_collapse_limit`

Other Maddening Manifestations

- Join order
- Prepared Statements
 - `PREPARE foo AS SELECT * FROM pgbench_accounts WHERE aid = $1;`
 - First 5 executions use a custom plan (taking into account `$1`)
 - After that, a generic plan is used
- `from_collapse_limit / join_collapse_limit`
- ORM!

Other aRRRgh Manifestations (ORM)

```
postgres=# \d mytable
               Table "public.mytable"
  Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
 col_a   | numeric                |           | not null |
 col_b   | numeric                |           | not null |
 col1    | character varying(128) |           |          |
 col2    | character varying(512) |           |          |
 col3    | character varying(128) |           |          |
 col4    | timestamp without time zone |         |          |
 col5    | character varying(128) |           |          |

postgres=# EXPLAIN (ANALYZE, BUFFERS) UPDATE mytable SET col1 = 'A', col2 = 'text', (...) WHERE col_a = '3443949' AND col_b = '2222696';
                                QUERY PLAN
-----
Update on mytable  (cost=0.43..8.45 rows=1 width=1364) (actual time=0.167..0.167 rows=0 loops=1)
  Buffers: shared hit=10
    ->  Index Scan using "mytable_idx" on mytable (cost=0.43..8.45 rows=1 width=1364) (actual time=0.074..0.074 rows=1 loops=1)
          Index Cond: ((mytable.col_a = '3443949'::numeric) AND (mytable.col_b = '2222696'::numeric))
          Buffers: shared hit=4
Planning time: 0.480 ms
Execution time: 0.252 ms
(8 rows)
```

Other aRRRgh Manifestations (ORM)

postgres=# \d mytable

Table "public.mytable"				
Column	Type	Collation	Nullable	Default
col_a	numeric		not null	
col_b	numeric		not null	
col1	character varying(128)			
col2	character varying(512)			
col3	character varying(128)			
col4	timestamp without time zone			
col5	character varying(128)			

postgres=# EXPLAIN (ANALYZE, BUFFERS) UPDATE mytable SET col1 = 'A', col2 = 'text', (...) WHERE col_a = '3443949' AND col_b = '2222696';

QUERY PLAN

Update on mytable (cost=0.43..8.45 rows=1 width=1364) (actual time=0.167..0.167 rows=0 loops=1)
 Buffers: shared hit=10
 -> Index Scan using "mytable_idx" on mytable (cost=0.43..8.45 rows=1 width=1364) (actual time=0.074..0.074 rows=1 loops=1)
 Index Cond: ((mytable.col_a = '3443949'::numeric) AND (mytable.col_b = '2222696'::numeric))
 Buffers: shared hit=4
Planning time: 0.480 ms
Execution time: 0.252 ms
(8 rows)

← 40.922 ms ????

Other aRRRgh Manifestations (ORM)

postgres=# \d mytable

Table "public.mytable"				
Column	Type	Collation	Nullable	Default
col_a	numeric		not null	
col_b	numeric		not null	
col1	character varying(128)			
col2	character varying(512)			
col3	character varying(128)			
col4	timestamp without time zone			
col5	character varying(128)			

duration: 40.922 ms statement: EXPLAIN (ANALYZE, BUFFERS) UPDATE mytable SET col1 = 'A', col2 = 'text', (...) WHERE col_a = '3443949' AND col_b = '2222696';

Update on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=889.070..889.070 rows=0 loops=1)

-> Seq Scan on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=847.736..850.867 rows=1 loops=1)

Filter: ((mytable.col_a)::double precision = '3443949'::double precision) AND ((mytable.col_b)::double precision = '2222696'::double precision)

Rows Removed by Filter: 3336167

Other aRRRgh Manifestations (ORM)

postgres=# \d mytable

Table "public.mytable"				
Column	Type	Collation	Nullable	Default
col_a	numeric		not null	
col_b	numeric		not null	
col1	character varying(128)			
col2	character varying(512)			
col3	character varying(128)			
col4	timestamp without time zone			
col5	character varying(128)			

40.922 ms !!!!

duration: 40.922 ms statement: EXPLAIN (ANALYZE, BUFFERS) UPDATE mytable SET col1 = 'A', col2 = 'text', (...) WHERE col_a = '3443949' AND col_b = '2222696';

Update on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=889.070..889.070 rows=0 loops=1)

-> Seq Scan on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=847.736..850.867 rows=1 loops=1)

Filter: ((mytable.col_a)::double precision = '3443949'::double precision) AND ((mytable.col_b)::double precision = '2222696'::double precision))

Rows Removed by Filter: 3336167

Other aRRRgh Manifestations (ORM)

```
postgres=# \d mytable
```

Table "public.mytable"				
Column	Type	Collation	Nullable	Default
col_a	numeric		not null	
col_b	numeric		not null	
col1	character varying(128)			
col2	character varying(512)			
col3	character varying(128)			
col4	timestamp without time zone			
col5	character varying(128)			

40.922 ms !!!!

duration: 40.922 ms statement: EXPLAIN (ANALYZE, BUFFERS) UPDATE mytable SET col1 = 'A', col2 = 'text', (...) WHERE col_a = '3443949' AND col_b = '2222696';

Update on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=889.070..889.070 rows=0 loops=1)
-> Seq Scan on mytable (cost=0.00..89304.06 rows=83 width=1364) (actual time=847.736..850.867 rows=1 loops=1)
Filter: ((mytable.col_a)::double precision = '3443949'::double precision) AND ((mytable.col_b)::double precision = '2222696'::double precision)
Rows Removed by Filter: 3336167

auto_explain

auto_explain

- Prints **EXPLAIN** plans to your log

auto_explain

- Prints **EXPLAIN** plans to your log
- Can do **EXPLAIN ANALYZE** (and BUFFERS, FORMAT, etc.)

auto_explain

- Prints **EXPLAIN** plans to your log
- Can do **EXPLAIN ANALYZE** (and BUFFERS, FORMAT, etc.)
- Can even log trigger statistics and nested statements

auto_explain

- Prints **EXPLAIN** plans to your log
- Can do **EXPLAIN ANALYZE** (and BUFFERS, FORMAT, etc.)
- Can even log trigger statistics and nested statements
- Can be done on a per-session basis with **LOAD auto_explain;**

auto_explain

- Prints **EXPLAIN** plans to your log
- Can do **EXPLAIN ANALYZE** (and BUFFERS, FORMAT, etc.)
- Can even log trigger statistics and nested statements
- Can be done on a per-session basis with **LOAD auto_explain;**
- Creates additional I/O on disk

Some More Tools



u9bL : Optimization for: Optim...

explain.depesz.com/s/u9bL

☆🔗🛑🌱⚙️🌐📄👤⋮

explain.depesz.com

PostgreSQL's explain analyze made readable

new explainhistoryhelpaboutcontact

login

Result: u9bL : Optimization for: Optimization for: plan #5MXq; plan #XhjS

Settings

Optimization path:

- #5MXq:
- #XhjS : Optimization for: plan #5MXq
- #u9bL : Optimization for: Optimization for: plan #5MXq; plan #XhjS

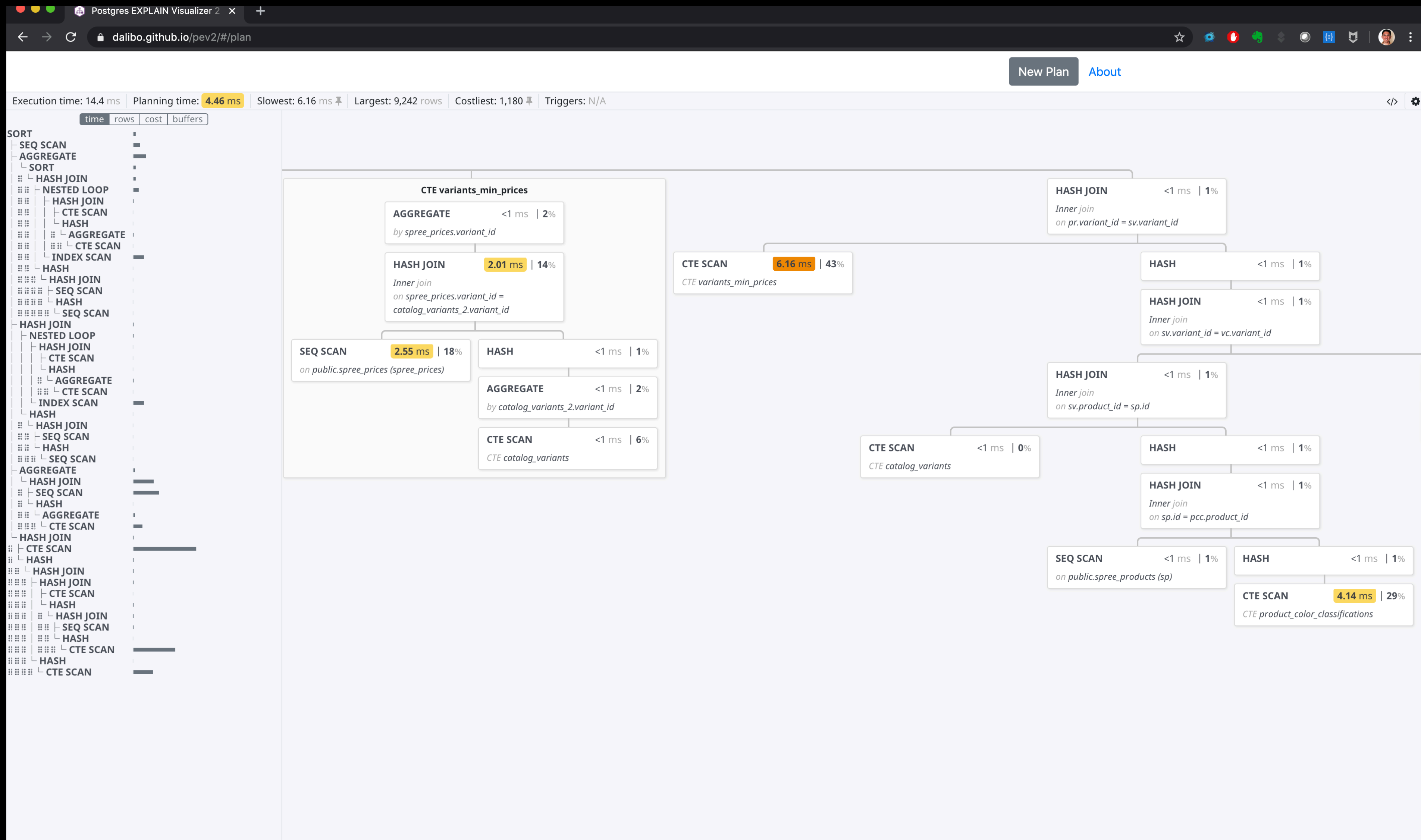
Optimization(s) for this plan:

- #REF1 : Optimization for: Optimization for: Optimization for: plan #5MXq; plan #XhjS; plan #u9bL

HTMLSOURCESTATS

Add optimization

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.151	15,811.072	↓ 1,000.0	1,000	1	→ Limit (cost=1,940.76..1,965.58 rows=1 width=16) (actual time=786.960..15,811.072 rows=1,000 loops=1)
2.	699.059	15,810.921	↓ 1,000.0	1,000	1	→ Nested Loop (cost=1,940.76..1,965.58 rows=1 width=16) (actual time=786.959..15,810.921 rows=1,000 loops=1) Join Filter: (home_addresses.account_id = work_addresses.account_id) Rows Removed by Join Filter: 10211736
3.	3.897	50.123	↓ 4,549.0	4,549	1	→ Nested Loop (cost=1,940.48..1,956.90 rows=1 width=32) (actual time=15.693..50.123 rows=4,549 loops=1)
4.	9.818	9.818	↓ 4,551.0	4,551	1	→ Index Scan using account_addresses_coordinates_type_equals_home_gist_idx on account_addresses home_addresses (cost=0.28..8.67 rows=1 width=16) (actual time=0.148..9.818 rows=4,551 loops=1) Index Cond: (coordinates && '0101000020E6100000D408A2934A9B5EC0CFB40E5883E24240'::geography) Filter: (('0101000020E6100000D408A2934A9B5EC0CFB40E5883E24240'::geography && _st_expand(coordinates, '5000'::double precision)) AND _st_dwithin('0101000020E6100000D408A2934A9B5EC0CFB40E5883E24240'::geography, coordinates, '5000'::double precision, true)) Rows Removed by Filter: 28
5.	29.064	36.408	↑ 1.0	1	4,551	→ Index Only Scan using accounts_pkey on accounts (cost=1,940.20..1,948.22 rows=1 width=16) (actual time=0.008..0.008 rows=1 loops=4,551) Index Cond: (id = home_addresses.account_id) Filter: (NOT (hashed SubPlan 1)) Rows Removed by Filter: 0 Heap Fetches: 3641
6.						SubPlan (for Index Only Scan)
7.	7.344	7.344	↓ 1.0	42,099	1	→ Seq Scan on account_bans (cost=0.00..1,834.60 rows=42,076 width=16) (actual time=0.007..7.344 rows=42,099 loops=1) Filter: (deleted_at IS NULL) Rows Removed by Filter: 89
8.	15,061.739	15,061.739	↓ 2,245.0	2,245	4,549	→ Index Scan using account_addresses_coordinates_type_equals_work_gist_idx on account_addresses work_addresses (cost=0.28..8.67 rows=1 width=16) (actual time=0.102..3.311 rows=2,245 loops=4,549) Index Cond: (coordinates && '0101000020E61000001C107C670F895EC0D8F11593EDB54240'::geography) Filter: (('0101000020E61000001C107C670F895EC0D8F11593EDB54240'::geography && _st_expand(coordinates, '5000'::double precision)) AND _st_dwithin('0101000020E61000001C107C670F895EC0D8F11593EDB54240'::geography, coordinates, '5000'::double precision, true)) Rows Removed by Filter: 50



Postgres Explain Visualizer (PEV) - explain.dalibo.com

Questions?