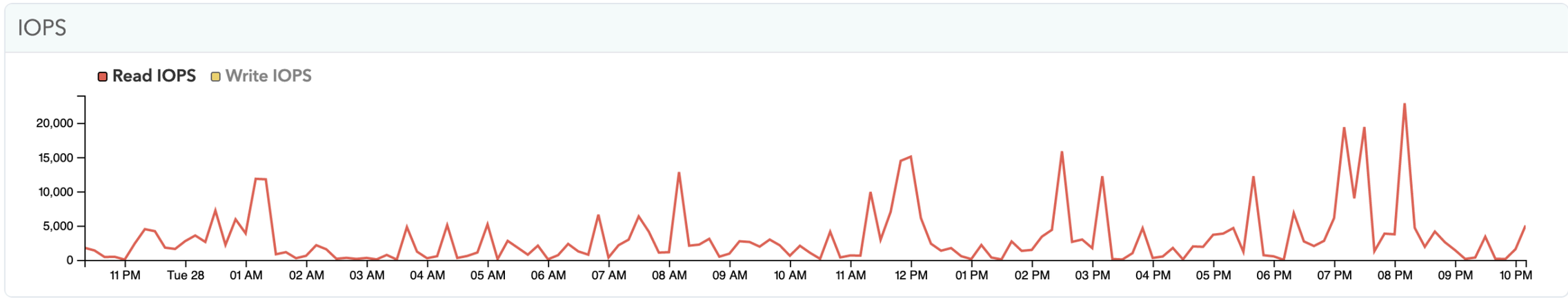
How to compare Postgres query plans & tune slow queries with pganalyze

@LukasFittl
hachyderm.io/@lukas

1. Identifying slow queries & their plans at scale
2. [New!] Plan Statistics in pganalyze
3. [New!] Plan Comparison in pganalyze
4. How to debug why a query is slow
5. Benchmarking alternate plans with settings & hints
6. [New!] Query Tuning Workbooks in pganalyze
7. Coming Soon: Query Tuning Advisor

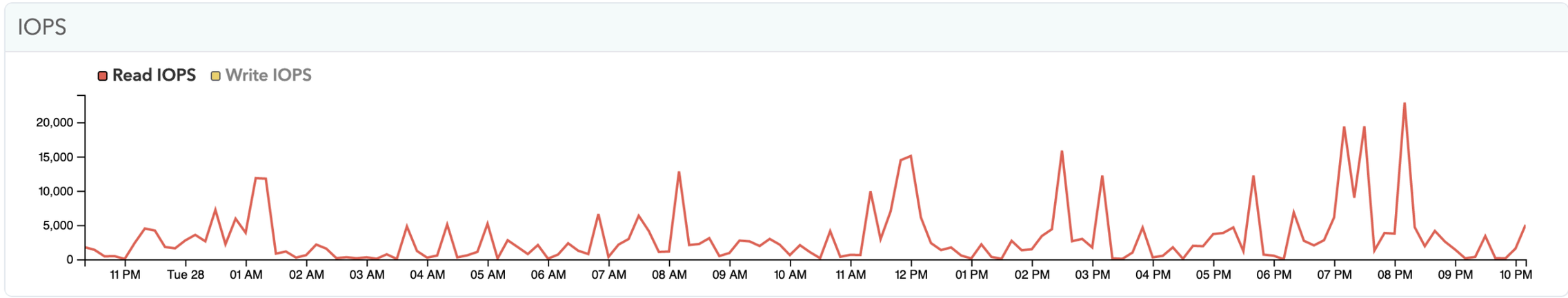
pganalyze

Identifying slow queries & their plans at scale



Why is our database spending so much [I/O Time | CPU Time | ...]?

pganalyze



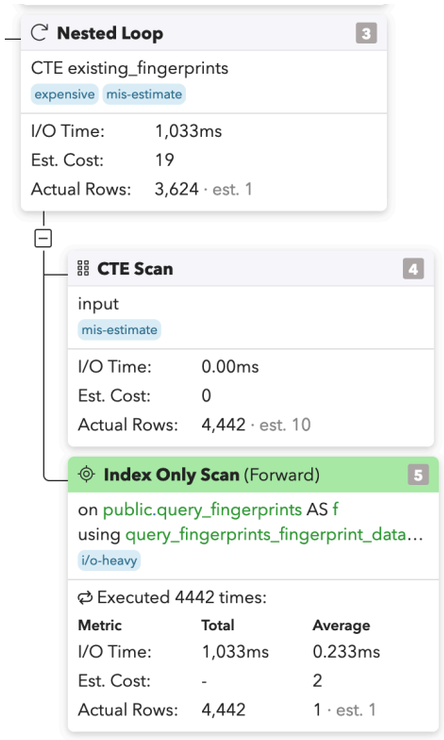
SELECT INSERT, UPDATE, DELETE DDL & other Compare to 7 days ago Search...

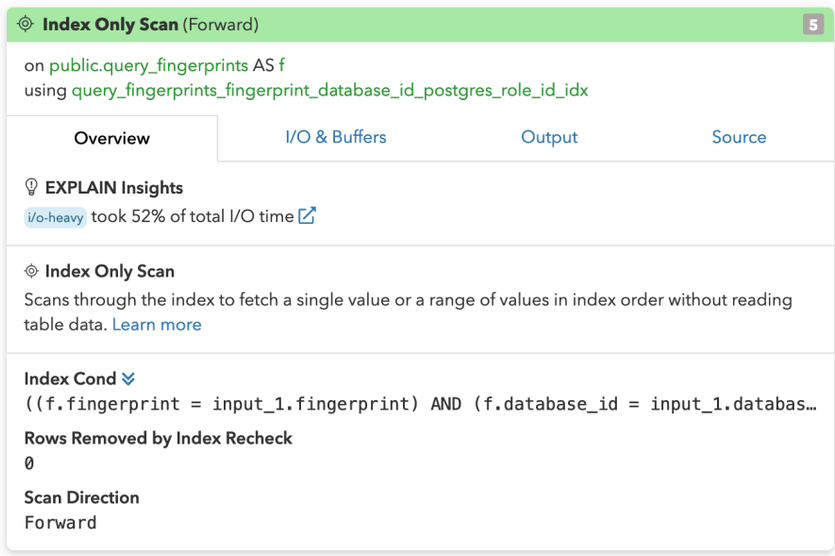
QUERY	ROLE	AVG TIME (MS)	CALLS / MIN	% OF ALL I/O	% OF ALL RUNTIME
WITH input AS (...), existing_fingerprints AS (...), update_queries AS (...)…	pgaweb_workers	3.78ms	14422.48	26.28%	17.28%
INSERT INTO query_stats_35d_20230328 (...) SELECT ... FROM unnest($1::int[],…	pgaweb_workers	145.54ms	246.50	8.02%	11.37%
INSERT INTO schema_index_stats_35d_20230329 (...) SELECT ... FROM unnest($1:…	pgaweb_workers	49.93ms	477.44	5.05%	7.55%
WITH total_times AS (...), table_queries AS (...), fingerprints AS (...), ra…	pgaweb_workers	123.95ms	181.59	9.20%	7.13%
WITH data AS (...), existing_rows AS (...), update_rows AS (...), insert_row…	pgaweb_workers	3.60ms	5229.28	9.15%	5.96%
WITH data(server_id, query_id, schema_table_scan_id, scan_node_type, scan_ta…	pgaweb_workers	18.60ms	760.78	4.95%	4.48%

pganalyze

```
WITH input AS (...)
SELECT *
  FROM query_fingerprints AS f
  JOIN input USING (database_id, fingerprint, postgres_role_id)
```

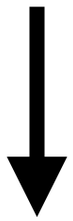
auto_explain + pganalyze





pganalyze

```
WITH input AS (...)
SELECT *
  FROM query_fingerprints AS f
  JOIN input USING (database_id, fingerprint, postgres_role_id)
```



EXPLAIN

```
->  Nested Loop  (cost=0.57..19.30 rows=1 width=45) (actual rows=3624 loops=1)
      Buffers: shared hit=19534 read=4214 dirtied=145
      I/O Timings: read=1033.376
      ->  CTE Scan on input 1  (cost=0.00..0.20 rows=10 width=60) (actual rows=4442 loops=1)
      CTE Name: input
->  Index Only Scan using … on public.query_fingerprints f (cost=0.57..1.91 rows=1 width=37) (…)
      Index Cond: ((…))
      Heap Fetches: 2603
      Buffers: shared hit=19534 read=4214 dirtied=145
      I/O Timings: read=1033.376
```

Is the query always slow, or just sometimes?

`WITH database_recent_table_stats AS (...), table_overhead_stats AS (...), table_total_writes AS (...`

Avg Time: 169.54ms | Calls Per Minute: 0.13 / min

fingerprint e6df89875f06a487 | role pgaweb_app | controller graphql | line /app/services/dataload/database.rb:102:in `total_i... | action graphql | View all query tags

Compare to 7 days ago

Overview | Index Advisor ? | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 58

Check-Up

1 new issue

Info Query **#43934696** takes 152 ms on average (1453 ms max, 2.25 MB read from disk per call, 168 calls in last 24h)

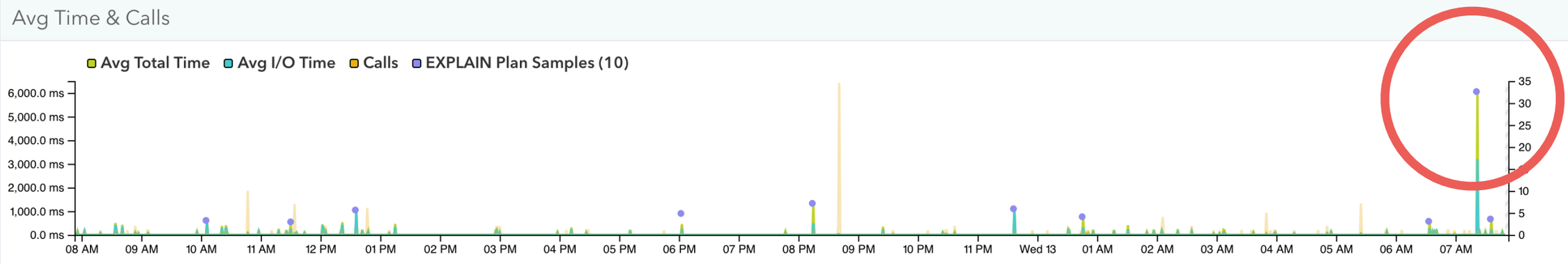
SQL Statement

```
/*controller:graphql,action:graphql,graphql:getDatabaseDetails.indexWriteOverhead,line:/app/services/dataload/database.rb:102:in `total_index_write_overhead_for_database',sentry
_trace_id:5c1dd0aae3384...
```

Show full query text

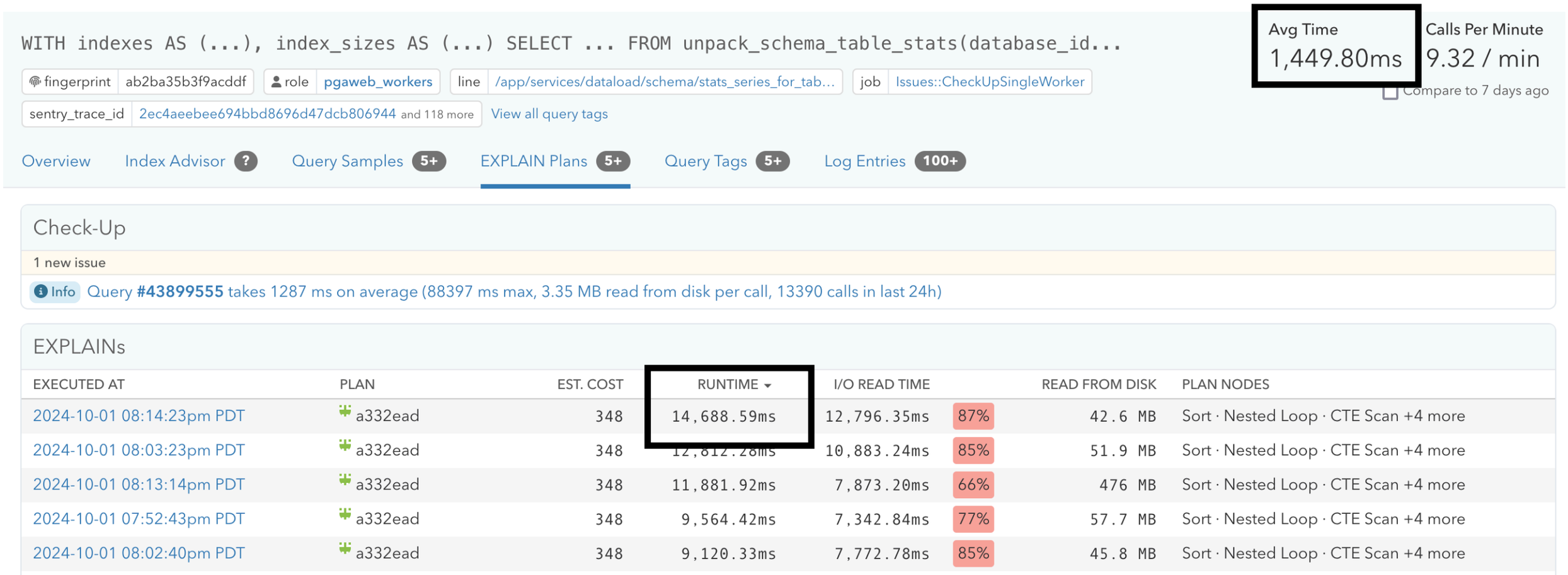
Tune query in workbook

Avg Time & Calls



pganalyze

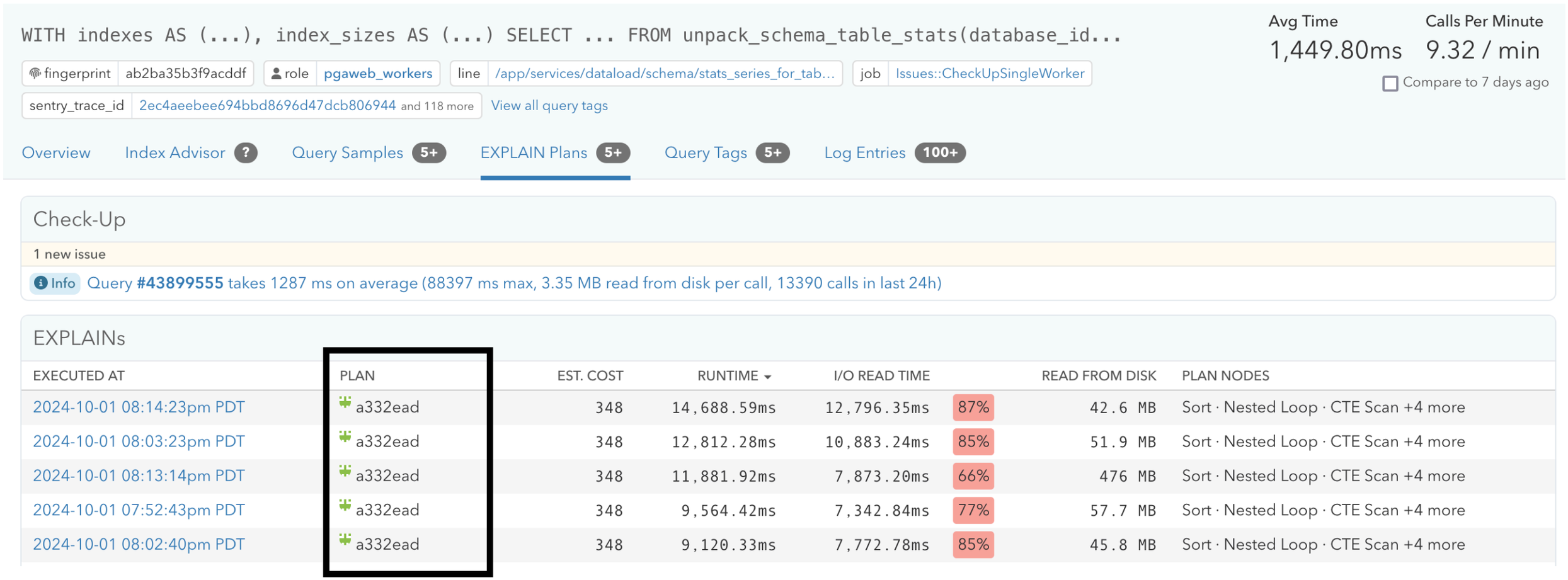
Is the query always slow, or just sometimes?



1.4s average vs **14.6 s** outlier execution

pganalyze

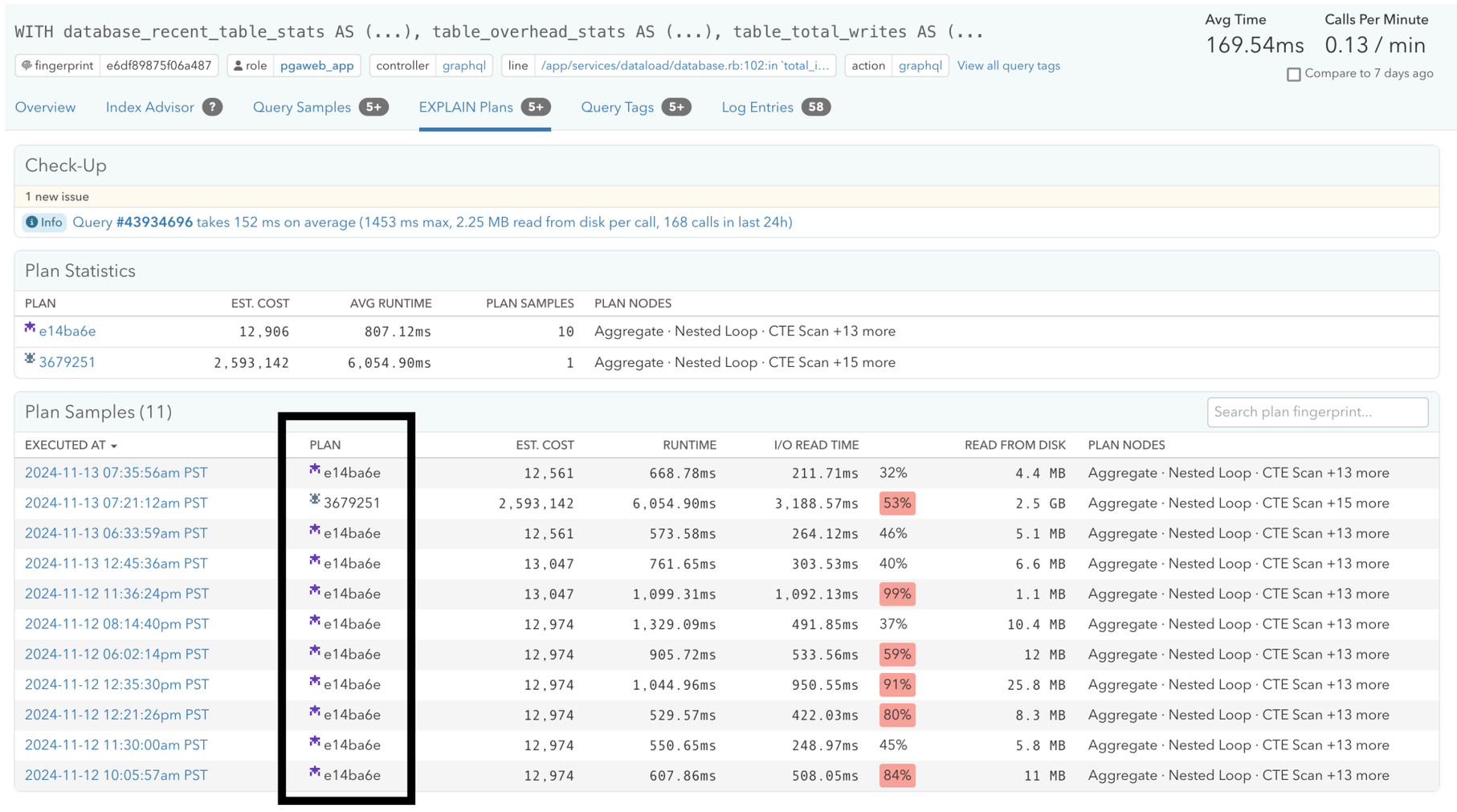
Is the plan the same, or does it change?



Plan Fingerprints show changes in plan structure

pganalyze

Is the plan the same, or does it change?



```
WITH database_recent_table_stats AS (...), table_overhead_stats AS (...), table_total_writes AS (...
```

fingerprint e6df89875f06a487 | role pgaweb_app | controller graphql | line /app/services/dataload/database.rb:102:in `total_i... | action graphql | View all query tags

Avg Time 169.54ms | Calls Per Minute 0.13 / min | Compare to 7 days ago

Overview | Index Advisor ? | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 58

Check-Up

1 new issue

Info Query **#43934696** takes 152 ms on average (1453 ms max, 2.25 MB read from disk per call, 168 calls in last 24h)

Plan Statistics

PLAN	EST. COST	AVG RUNTIME	PLAN SAMPLES	PLAN NODES
e14ba6e	12,906	807.12ms	10	Aggregate · Nested Loop · CTE Scan +13 more
3679251	2,593,142	6,054.90ms	1	Aggregate · Nested Loop · CTE Scan +15 more

Plan Samples (11)

Search plan fingerprint...

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME		READ FROM DISK	PLAN NODES
2024-11-13 07:35:56am PST	e14ba6e	12,561	668.78ms	211.71ms	32%	4.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-13 07:21:12am PST	3679251	2,593,142	6,054.90ms	3,188.57ms	53%	2.5 GB	Aggregate · Nested Loop · CTE Scan +15 more
2024-11-13 06:33:59am PST	e14ba6e	12,561	573.58ms	264.12ms	46%	5.1 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-13 12:45:36am PST	e14ba6e	13,047	761.65ms	303.53ms	40%	6.6 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:36:24pm PST	e14ba6e	13,047	1,099.31ms	1,092.13ms	99%	1.1 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 08:14:40pm PST	e14ba6e	12,974	1,329.09ms	491.85ms	37%	10.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 06:02:14pm PST	e14ba6e	12,974	905.72ms	533.56ms	59%	12 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:35:30pm PST	e14ba6e	12,974	1,044.96ms	950.55ms	91%	25.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:21:26pm PST	e14ba6e	12,974	529.57ms	422.03ms	80%	8.3 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:30:00am PST	e14ba6e	12,974	550.65ms	248.97ms	45%	5.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 10:05:57am PST	e14ba6e	12,974	607.86ms	508.05ms	84%	11 MB	Aggregate · Nested Loop · CTE Scan +13 more

Plan Fingerprints show changes in plan structure

pganalyze

pganalyze

[New!] Plan Statistics in pganalyze

New for all databases: Grouping of plan types

```
WITH database_recent_table_stats AS (...), table_overhead_stats AS (...), table_total_writes AS (...
```

fingerprint e6df89875f06a487 | role pgaweb_app | controller graphql | line /app/services/dataload/database.rb:102:in `total_i... | action graphql | View all query tags

Avg Time 169.54ms | Calls Per Minute 0.13 / min

Compare to 7 days ago

Overview | Index Advisor ? | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 58

Check-Up

1 new issue

Info Query **#43934696** takes 152 ms on average (1453 ms max, 2.25 MB read from disk per call, 168 calls in last 24h)

Plan Statistics

PLAN	EST. COST	AVG RUNTIME	PLAN SAMPLES	PLAN NODES
e14ba6e	12,906	807.12ms	10	Aggregate · Nested Loop · CTE Scan +13 more
3679251	2,593,142	6,054.90ms	1	Aggregate · Nested Loop · CTE Scan +15 more

Plan Samples (11)

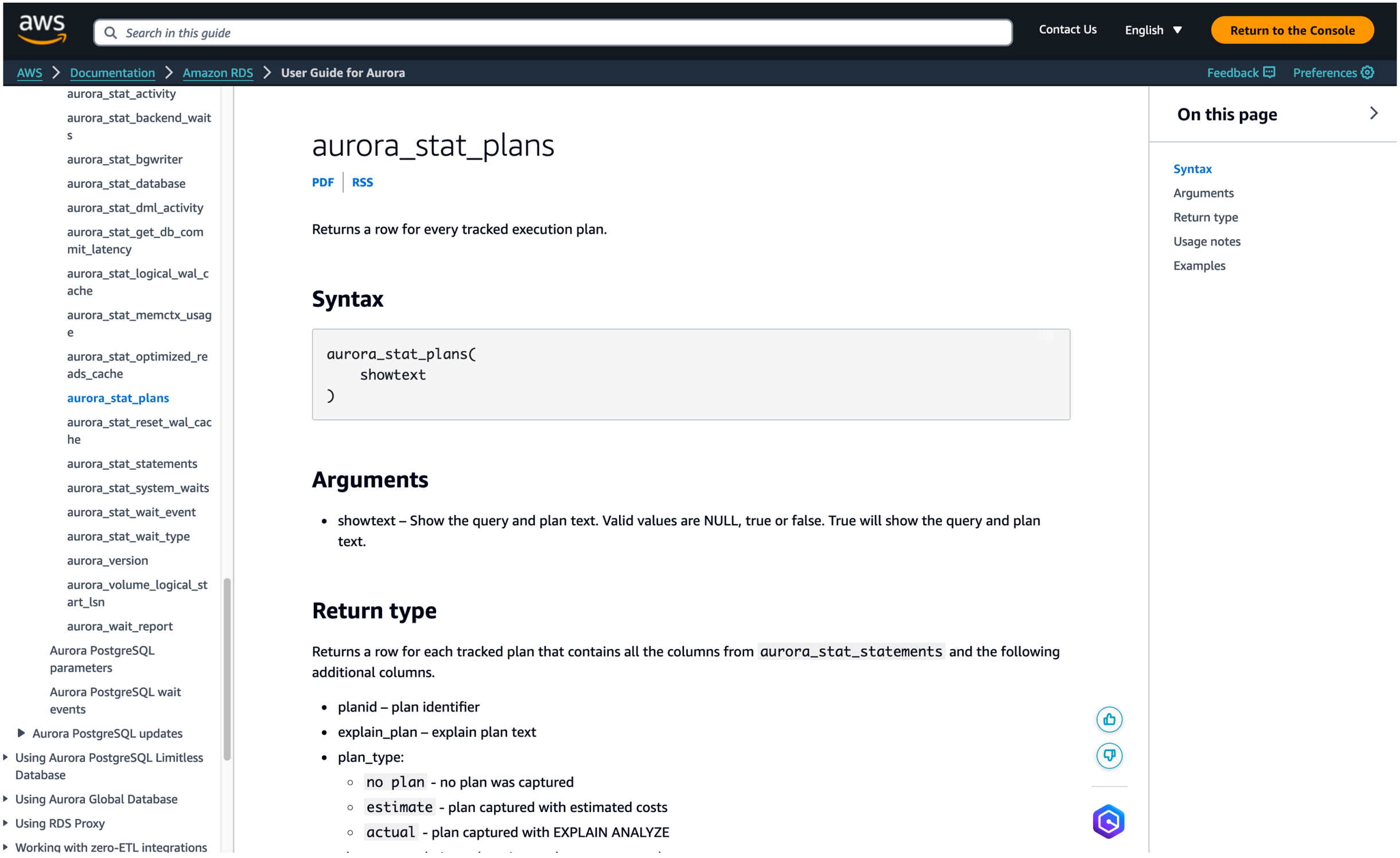
Search plan fingerprint

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME		READ FROM DISK	PLAN NODES
2024-11-13 07:35:56am PST	e14ba6e	12,561	668.78ms	211.71ms	32%	4.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-13 07:21:12am PST	3679251	2,593,142	6,054.90ms	3,188.57ms	53%	2.5 GB	Aggregate · Nested Loop · CTE Scan +15 more
2024-11-13 06:33:59am PST	e14ba6e	12,561	573.58ms	264.12ms	46%	5.1 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-13 12:45:36am PST	e14ba6e	13,047	761.65ms	303.53ms	40%	6.6 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:36:24pm PST	e14ba6e	13,047	1,099.31ms	1,092.13ms	99%	1.1 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 08:14:40pm PST	e14ba6e	12,974	1,329.09ms	491.85ms	37%	10.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 06:02:14pm PST	e14ba6e	12,974	905.72ms	533.56ms	59%	12 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:35:30pm PST	e14ba6e	12,974	1,044.96ms	950.55ms	91%	25.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:21:26pm PST	e14ba6e	12,974	529.57ms	422.03ms	80%	8.3 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:30:00am PST	e14ba6e	12,974	550.65ms	248.97ms	45%	5.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 10:05:57am PST	e14ba6e	12,974	607.86ms	508.05ms	84%	11 MB	Aggregate · Nested Loop · CTE Scan +13 more

pganalyze

New for Amazon Aurora: aurora_stat_plans

Aurora PostgreSQL versions 14.10, 15.5, and later versions.



New for Amazon Aurora: aurora_stat_plans

```
UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2
```

fingerprint 3315bfa60c2c07a3 | role benchmark | No query tags collected

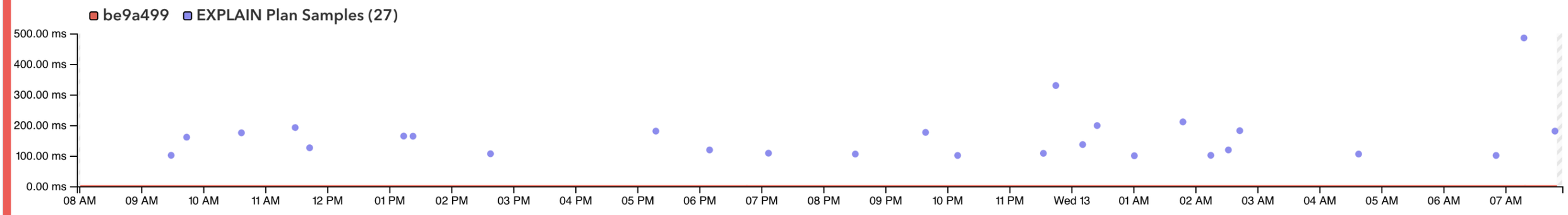
Avg Time 0.15ms | Calls Per Minute 10,991.73 / min | Compare to 7 days ago

Overview | Index Advisor | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 0 | Log Entries 100+

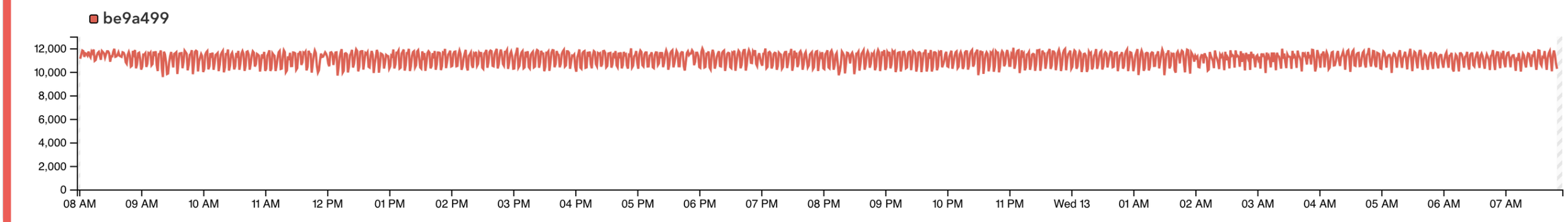
Plan Statistics

PLAN	EST. COST	AVG RUNTIME	CALLS / MIN	ORIGINAL PLAN ID	PLAN NODES
be9a499	8	0.15ms	11,125	-129590782	ModifyTable · Index Scan

Avg Time



Calls



Plan Samples (65)

Search plan fingerprint...

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME		READ FROM DISK	PLAN NODES
2024-11-13 07:48:00am PST	be9a499	8	180.62ms	0.00ms	0%	0 B	ModifyTable · Index Scan

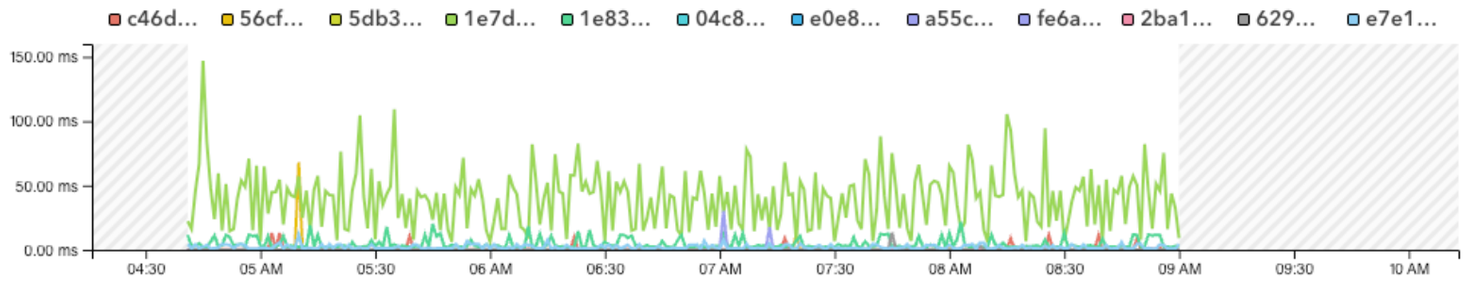
New for Amazon Aurora: aurora_stat_plans

Plan Statistics

PLAN	EST. COST	AVG RUNTIME	CALLS / MIN	ORIGINAL PLAN ID	PLAN NODES
62999af	2	0.11ms	1	1579471858	Hash Join · Hash · Seq Scan · Seq Scan
2ba12be	1	0.02ms	7	1505938809	Aggregate · Seq Scan
1e7d9e1	0	41.08ms	2	38895744	Function Scan
1e832d3	13	4.61ms	7	38895872	Function Scan
e7e1c46	15	2.12ms	21	1614807188	Aggregate · Function Scan

1-5 of 12

Avg Time



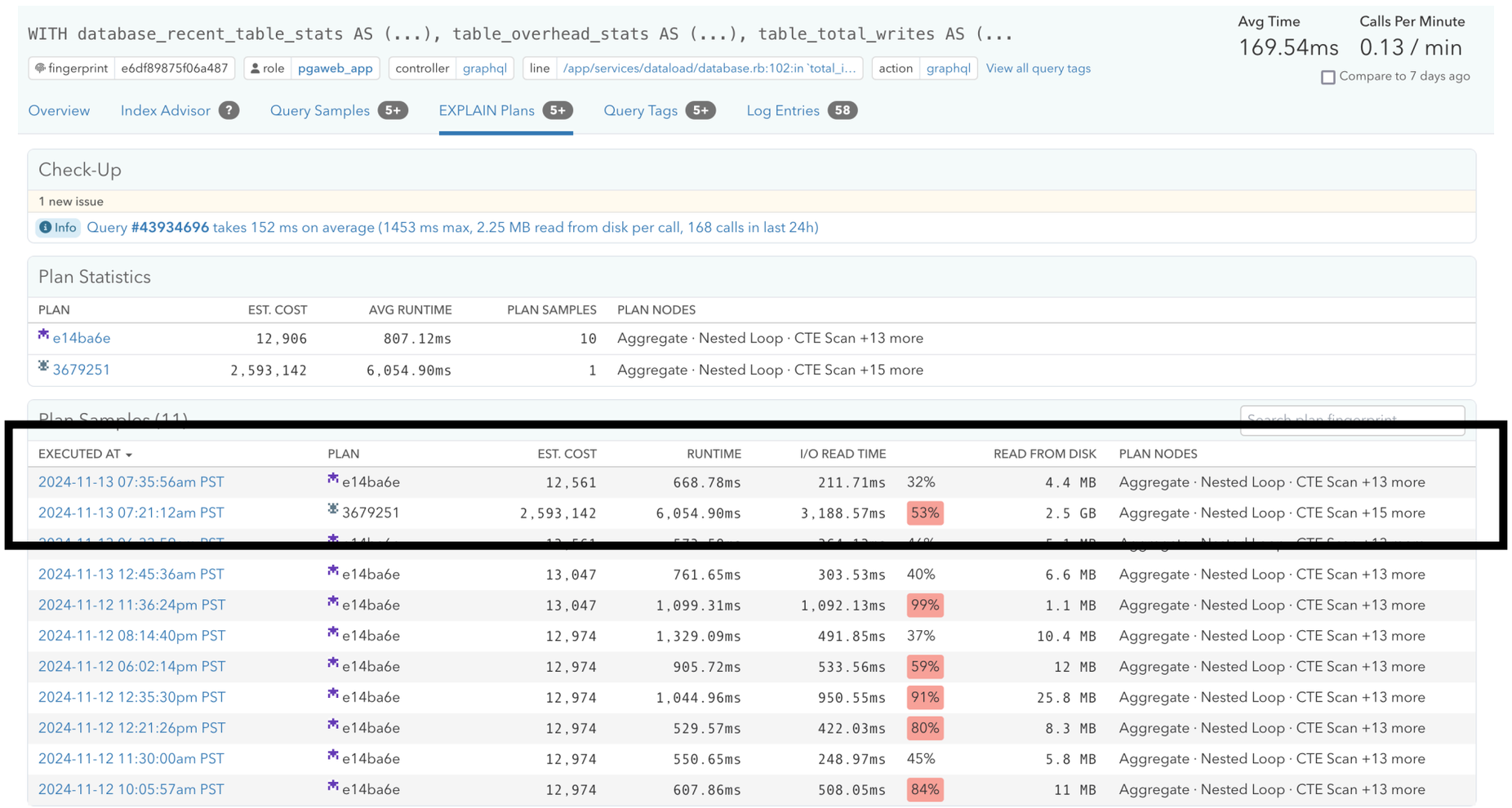
Calls

pganalyze

pganalyze

[New!] Plan Comparison in pganalyze

Comparing Automated EXPLAIN plans



WITH database_recent_table_stats AS (...), table_overhead_stats AS (...), table_total_writes AS (...

fingerprint e6df89875f06a487 | role pgaweb_app | controller graphql | line /app/services/dataload/database.rb:102:in `total_i... | action graphql | View all query tags

Avg Time 169.54ms | Calls Per Minute 0.13 / min

Compare to 7 days ago

Overview | Index Advisor ? | Query Samples 5+ | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 58

Check-Up

1 new issue

Info Query #43934696 takes 152 ms on average (1453 ms max, 2.25 MB read from disk per call, 168 calls in last 24h)

Plan Statistics

PLAN	EST. COST	AVG RUNTIME	PLAN SAMPLES	PLAN NODES
e14ba6e	12,906	807.12ms	10	Aggregate · Nested Loop · CTE Scan +13 more
3679251	2,593,142	6,054.90ms	1	Aggregate · Nested Loop · CTE Scan +15 more

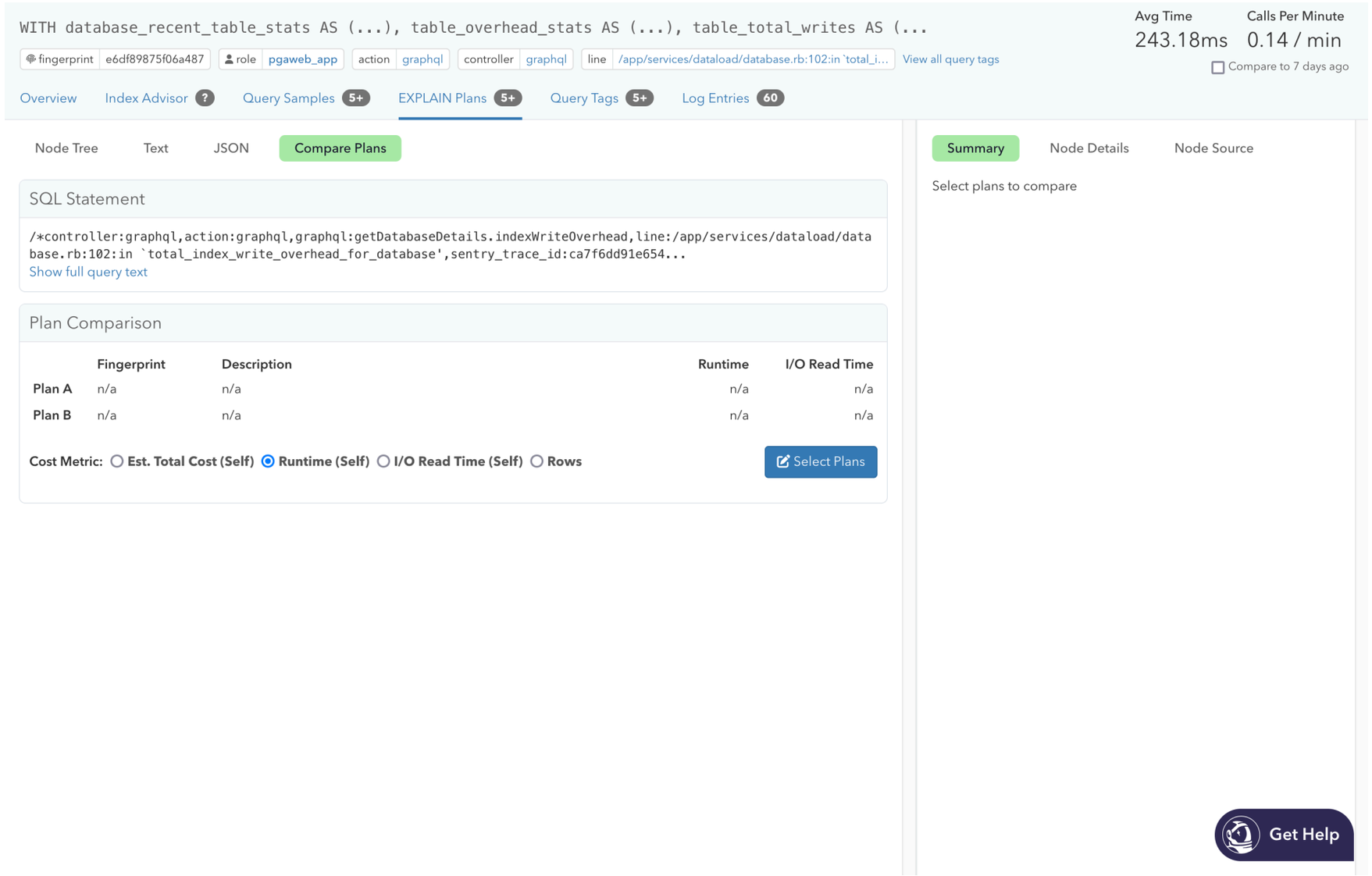
Plan Samples (11)

Search plan fingerprint

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME		READ FROM DISK	PLAN NODES
2024-11-13 07:35:56am PST	e14ba6e	12,561	668.78ms	211.71ms	32%	4.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-13 07:21:12am PST	3679251	2,593,142	6,054.90ms	3,188.57ms	53%	2.5 GB	Aggregate · Nested Loop · CTE Scan +15 more
2024-11-13 12:45:36am PST	e14ba6e	13,047	761.65ms	303.53ms	40%	6.6 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:36:24pm PST	e14ba6e	13,047	1,099.31ms	1,092.13ms	99%	1.1 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 08:14:40pm PST	e14ba6e	12,974	1,329.09ms	491.85ms	37%	10.4 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 06:02:14pm PST	e14ba6e	12,974	905.72ms	533.56ms	59%	12 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:35:30pm PST	e14ba6e	12,974	1,044.96ms	950.55ms	91%	25.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 12:21:26pm PST	e14ba6e	12,974	529.57ms	422.03ms	80%	8.3 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 11:30:00am PST	e14ba6e	12,974	550.65ms	248.97ms	45%	5.8 MB	Aggregate · Nested Loop · CTE Scan +13 more
2024-11-12 10:05:57am PST	e14ba6e	12,974	607.86ms	508.05ms	84%	11 MB	Aggregate · Nested Loop · CTE Scan +13 more

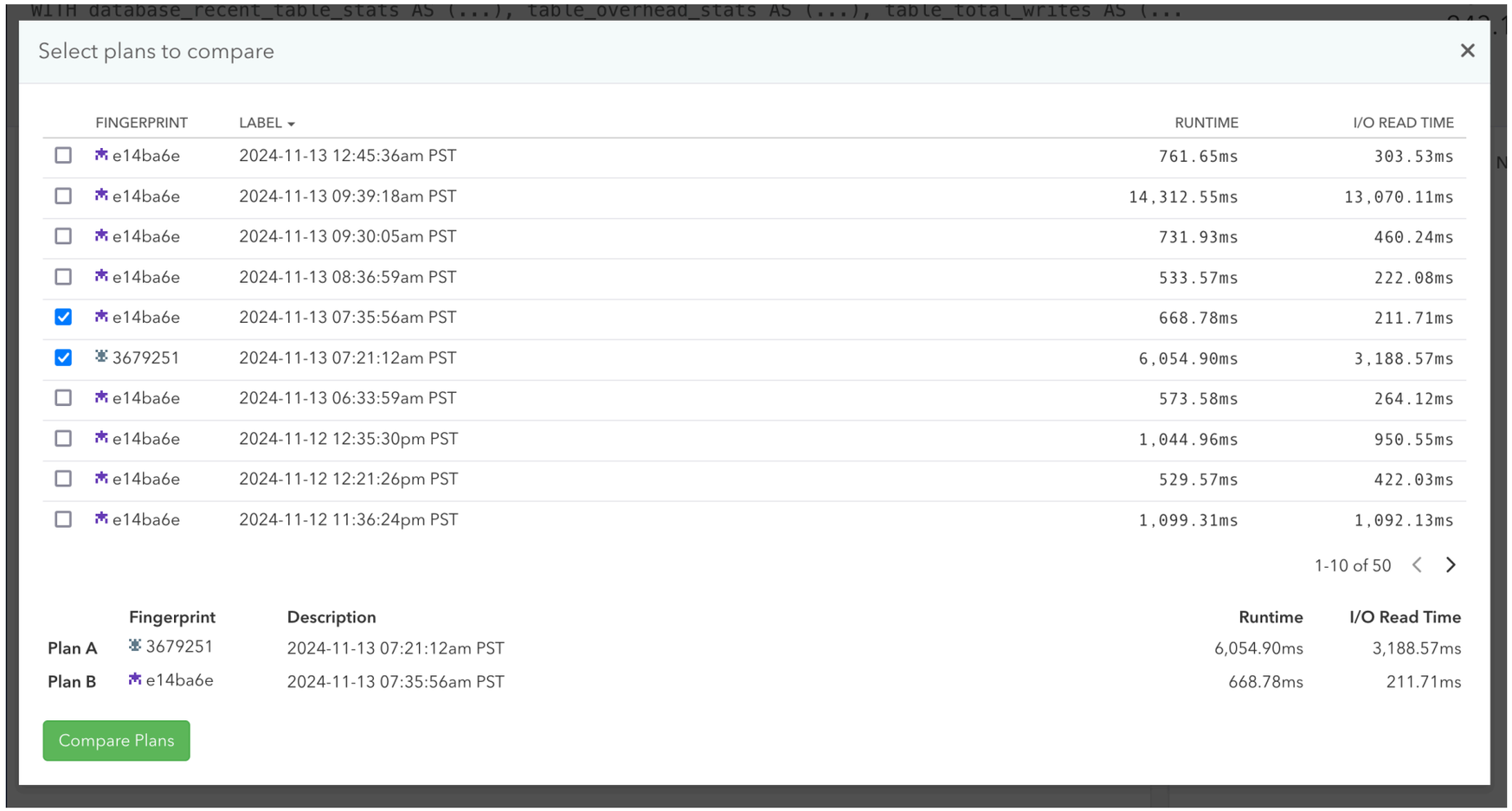
What's different between the first and the second plan?

Comparing Automated EXPLAIN plans



pganalyze

Comparing Automated EXPLAIN plans



pganalyze

Comparing Automated EXPLAIN plans

Node Tree | Text | JSON | Compare Plans

SQL Statement

```
/*controller:graphql,action:graphql,graphql:getDatabaseDetails.indexWriteOverhead,line:/app/services/dataload/database.rb:102:in `total_index_write_overhead_for_database',sentry_trace_id:ca7f6dd91e654...
```

Show full query text

Plan Comparison

	Fingerprint	Description	Runtime	I/O Read Time
Plan A	3679251	2024-11-13 07:21:12am PST	6,054.90ms	3,188.57ms
Plan B	e14ba6e	2024-11-13 07:35:56am PST	668.78ms	211.71ms

Cost Metric: ○ **Est. Total Cost (Self)** ◉ **Runtime (Self)** ○ **I/O Read Time (Self)** ○ **Rows**

Select Plans

Plan A	Plan B	Plan A: Runtime	Plan B: Runtime
-> Aggregate	-> Aggregate	0.00ms	0.00ms
-> Hash Join	-> Hash Join	0.74ms	61.76ms
-> Function Scan	-> Function Scan	0.71ms	72.97ms
-> Hash	-> Hash	74.77ms	489.14ms
-> Bitmap Heap Scan		0.00ms	0.31ms
-> Bitmap Index Scan[3]		0.01ms	
	-> Index Scan[3]	0.12ms	
-> Aggregate	-> Aggregate		1.63ms
-> Hash Join		0.10ms	2.41ms
-> Nested Loop	-> Nested Loop	305.26ms	
-> CTE Scan	-> CTE Scan	0.04ms	37.95ms
	-> Nested Loop	76.35ms	626.42ms
-> Index Scan[2]	-> Index Scan[2]		0.00ms
-> Hash		0.01ms	0.00ms
-> Seq Scan		1,192.79ms	
	-> Index Scan[1]	4,480.30ms	
-> Nested Loop	-> Nested Loop		0.01ms
-> Aggregate	-> Aggregate	0.00ms	0.34ms
-> CTE Scan	-> CTE Scan	0.01ms	0.32ms
-> CTE Scan	-> CTE Scan	6,054.86ms	667.38ms
		0.00ms	0.19ms

Summary | Node Details | Node Source

	Plan A	Plan B
Seen At	Nov 13 07:21am	Nov 13 07:35am
Plan Fingerprint	3679251	e14ba6e
Runtime	6,054.90ms	668.78ms
I/O Read Time	3,188.57ms	211.71ms
Read From Disk	2.5 GB	4.4 MB
Total Est. Cost	2,593,142	12,561

Index usage

A	B	Index
✓		1. index_schema_index_metadata_on_index_id
✓	✓	2. schema_indices_table_id_name_idx
✓	✓	3. schema_tables_database_id_schema_name_table_name_idx

Get Help

pganalyze

Comparing Automated EXPLAIN plans

Node Tree | Text | JSON | Compare Plans

SQL Statement

```
/*controller:graphql,action:graphql,graphql:getDatabaseDetails.indexWriteOverhead,line:/app/services/dataload/database.rb:102:in `total_index_write_overhead_for_database',sentry_trace_id:ca7f6dd91e654...
```

Show full query text

Plan Comparison

	Fingerprint	Description	Runtime	I/O Read Time
Plan A	3679251	2024-11-13 07:21:12am PST	6,054.90ms	3,188.57ms
Plan B	e14ba6e	2024-11-13 07:35:56am PST	668.78ms	211.71ms

Cost Metric: ○ **Est. Total Cost (Self)** ○ **Runtime (Self)** ○ **I/O Read Time (Self)** ◉ **Rows**

Select Plans

Plan A	Plan B	Plan A: Rows	Plan B: Rows
-> Aggregate	-> Aggregate	1	1
-> Aggregate	-> Aggregate	5	2,461
-> Hash Join	-> Hash Join	690	338,928
-> Function Scan	-> Function Scan	690	339,462
-> Hash	-> Hash	5	2,461
-> Bitmap Heap Scan		5	
-> Bitmap Index Scan[3]		5	
	-> Index Scan[3]		2,461
-> Aggregate	-> Aggregate	5	2,461
-> Hash Join		8	
-> Nested Loop	-> Nested Loop	8	5,009
-> CTE Scan	-> CTE Scan	5	2,461
	-> Nested Loop		2
-> Index Scan[2]	-> Index Scan[2]	1	2
-> Hash		5,050,837	
-> Seq Scan		5,050,837	
	-> Index Scan[1]		1
-> Nested Loop	-> Nested Loop	5	2,461
-> Aggregate	-> Aggregate	1	1
-> CTE Scan	-> CTE Scan	5	2,461
-> CTE Scan	-> CTE Scan	5	2,461

Summary | Node Details | Node Source

	Plan A	Plan B
Seen At	Nov 13 07:21am	Nov 13 07:35am
Plan Fingerprint	3679251	e14ba6e
Runtime	6,054.90ms	668.78ms
I/O Read Time	3,188.57ms	211.71ms
Read From Disk	2.5 GB	4.4 MB
Total Est. Cost	2,593,142	12,561

Index usage

A	B	Index
✓		1. index_schema_index_metadata_on_index_id
✓	✓	2. schema_indices_table_id_name_idx
✓	✓	3. schema_tables_database_id_schema_name_table_name_idx

Get Help

pganalyze

Comparing Automated EXPLAIN plans

Node Tree | Text | JSON | **Compare Plans**

SQL Statement

```
/*controller:graphql,action:graphql,graphql:getDatabaseDetails.indexWriteOverhead,line:/app/services/dataload/database.rb:102:in `total_index_write_overhead_for_database',sentry_trace_id:ca7f6dd91e654...
```

Show full query text

Plan Comparison

	Fingerprint	Description	Runtime	I/O Read Time
Plan A	3679251	2024-11-13 07:21:12am PST	6,054.90ms	3,188.57ms
Plan B	e14ba6e	2024-11-13 07:35:56am PST	668.78ms	211.71ms

Cost Metric: ○ **Est. Total Cost (Self)** ○ **Runtime (Self)** ○ **I/O Read Time (Self)** ◉ **Rows**

Select Plans

Plan A	Plan B	Plan A: Rows	Plan B: Rows
-> Aggregate	-> Aggregate	1	1
-> Aggregate	-> Aggregate	5	2,461
-> Hash Join	-> Hash Join	690	338,928
-> Function Scan	-> Function Scan	690	339,462
-> Hash	-> Hash	5	2,461
-> Bitmap Heap Scan		5	
-> Bitmap Index Scan[3]		5	
	-> Index Scan[3]		2,461
-> Aggregate	-> Aggregate	5	2,461
-> Hash Join		8	
-> Nested Loop	-> Nested Loop	8	5,009
-> CTE Scan	-> CTE Scan	5	2,461
	-> Nested Loop		2
-> Index Scan[2]	-> Index Scan[2]	1	2
-> Hash		5,050,837	
-> Seq Scan		5,050,837	
	-> Index Scan[1]		1
-> Nested Loop	-> Nested Loop	5	2,461
-> Aggregate	-> Aggregate	1	1
-> CTE Scan	-> CTE Scan	5	2,461
-> CTE Scan	-> CTE Scan	5	2,461

Summary | **Node Details** | Node Source

From Plan A

Seq Scan

on public.schema_index_metadata AS sim

Scans through the entire table row-by-row in an arbitrary order.
Learn more

Insights (2)

expensive was estimated to be expensive (cost 392911 vs avg cost 144063)

i/o-heavy took 98% of total I/O time

I/O & Buffers

	Shared	Local	Temp
Hit	92.5 MB	0 B	-
Read	2.5 GB	0 B	0 B
Dirtied	17.3 MB	0 B	-
Written	0 B	0 B	0 B

I/O Read Time: 3,130.47ms
I/O Write Time: 0.00ms

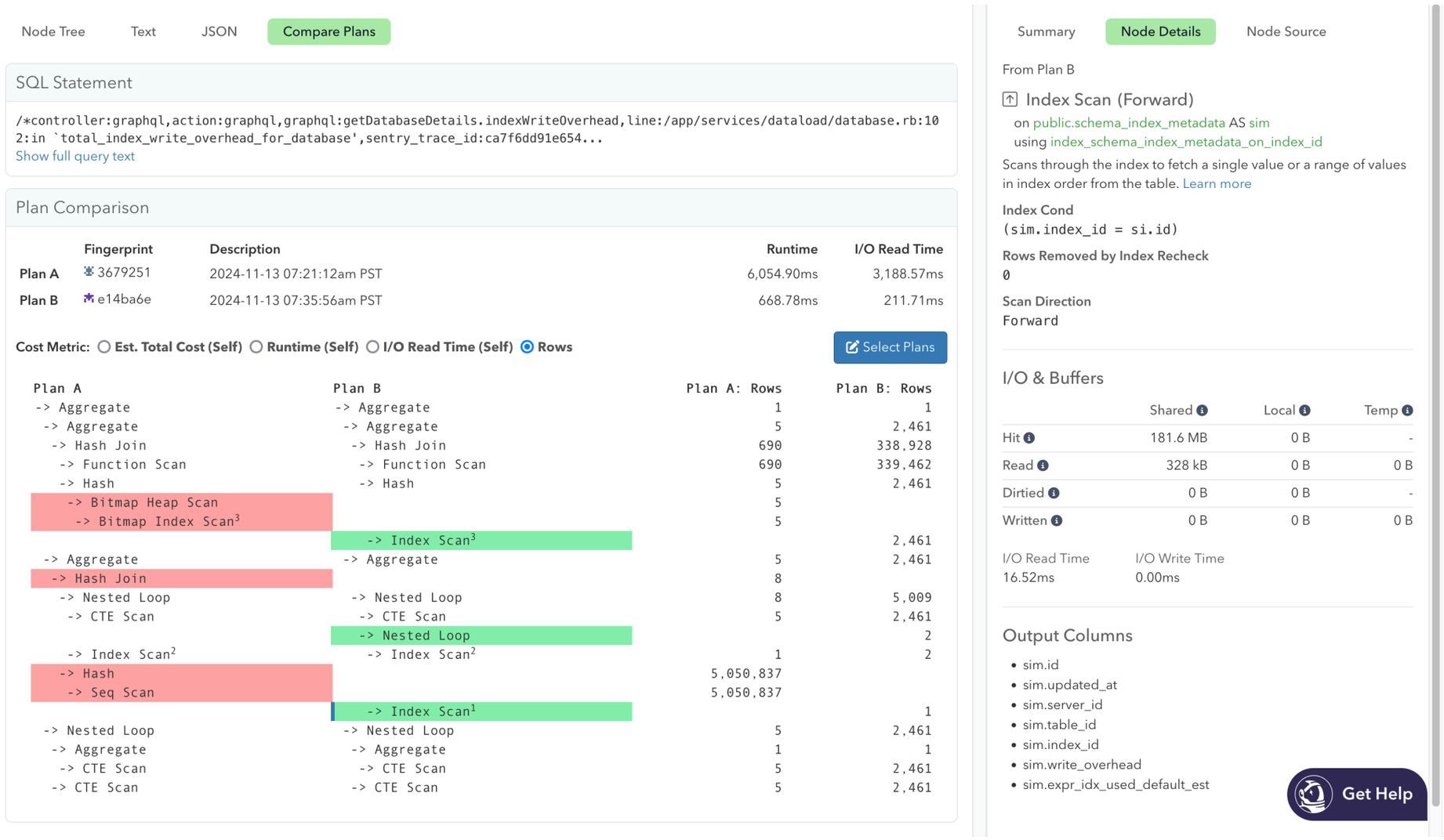
Output Columns

- sim.write_overhead
- sim.id
- sim.expr_idx_used_default_est
- sim.updated_at
- sim.index_id

Get Help

pganalyze

Comparing Automated EXPLAIN plans



pganalyze

CTEs and InitPlans are split out in the diff

pganalyze

Server: prod-db-main (Primary)

Database: pgaweb

Last 24 hours

ORGANIZATION: pganalyze

- Dashboard
- Query Performance
- Query Tuning
- Index Advisor
- VACUUM Advisor
- Schema Statistics
- Log Insights
- Connections
- Config Settings
- System
- Alerts & Check-Up
- Settings

Maciek Sak…

```
WITH total_times AS (...), qfp AS (...), query_times AS (...) SELECT ... FROM total_times LEFT JO...
```

fingerprint 6d554e1b6c03800b | role pgaweb_app | line /app/services/dataload/queries/query_stats_for_que… | controller graphql | action graphql | View all query tags

Avg Time: 99.21ms

Calls Per Minute: 0.51 / min

Compare to 7 days ago

Overview | Index Advisor ? | Query Samples 4 | EXPLAIN Plans 5+ | Query Tags 5+ | Log Entries 100+

Node Tree | Text | JSON | Compare Plans

Show full query text

Plan Comparison

	Fingerprint	Description	Runtime	I/O Read Time
Plan A	c55dd60	2024-11-12 09:38:31am PST	620.45ms	459.29ms
Plan B	19bc620	2024-11-12 09:40:44am PST	1,076.20ms	760.11ms

Cost Metric: ○ Est. Total Cost (Self) ◉ Runtime (Self) ○ I/O Read Time (Self) ○ Rows

Select Plans

Plan A	Plan B	Plan A: Runtime	Plan B: Runtime
-> Merge Join	-> Merge Join	0.00ms	0.00ms
-> Sort	-> Sort	0.24ms	0.47ms
-> Result	-> Result	0.31ms	0.58ms
-> Append	-> Append	0.12ms	0.24ms
	-> Index Only Scan[2]		0.08ms
-> Index Only Scan[3]	-> Index Only Scan[3]	0.11ms	0.26ms
-> Index Only Scan[4]	-> Index Only Scan[4]	0.19ms	0.18ms
-> Materialize	-> Materialize	0.17ms	0.39ms
-> Sort	-> Sort	0.32ms	0.69ms
-> CTE Scan	-> CTE Scan	617.66ms	1,069.88ms
CTE qfp			
-> Index Scan[1]	-> Index Scan[1]	0.01ms	0.01ms
CTE query_times			
-> Aggregate	-> Aggregate	2.19ms	5.00ms
-> Sort	-> Sort	0.29ms	0.61ms
-> Nested Loop	-> Nested Loop	0.82ms	1.90ms
-> Function Scan	-> Function Scan	613.93ms	1,061.32ms
-> CTE Scan	-> CTE Scan	0.00ms	0.00ms
InitPlan 2 (returns $1)			
-> CTE Scan	-> CTE Scan	0.01ms	0.01ms

Summary | Node Details | Node Source

	Plan A	Plan B
Seen At	Nov 12 09:38am	Nov 12 09:40am
Plan Fingerprint	c55dd60	19bc620
Runtime	620.45ms	1,076.20ms
I/O Read Time	459.29ms	760.11ms
Read From Disk	10.1 MB	15.7 MB
Total Est. Cost	378	634

Index usage

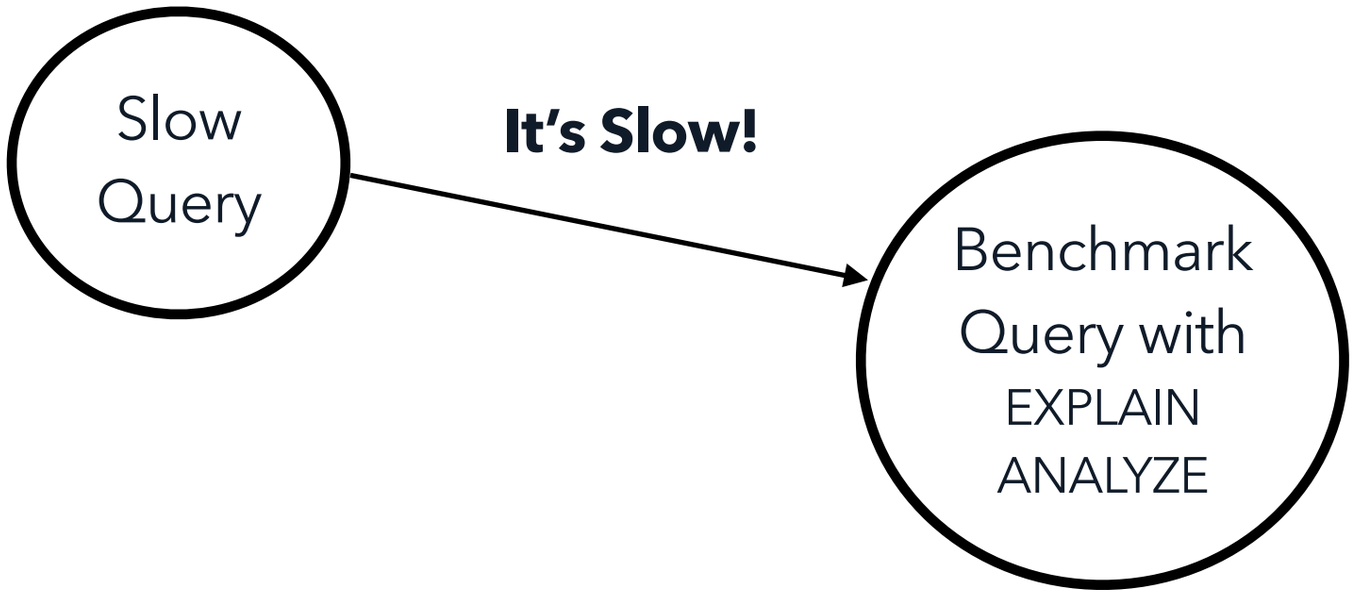
A	B	Index
✓	✓	1. query_fingerprints_query_id_idx
	✓	2. query_overview_stats_35d_20241110_pkey
✓	✓	3. query_overview_stats_35d_20241111_pkey
✓	✓	4. query_overview_stats_35d_20241112_pkey

Get Help

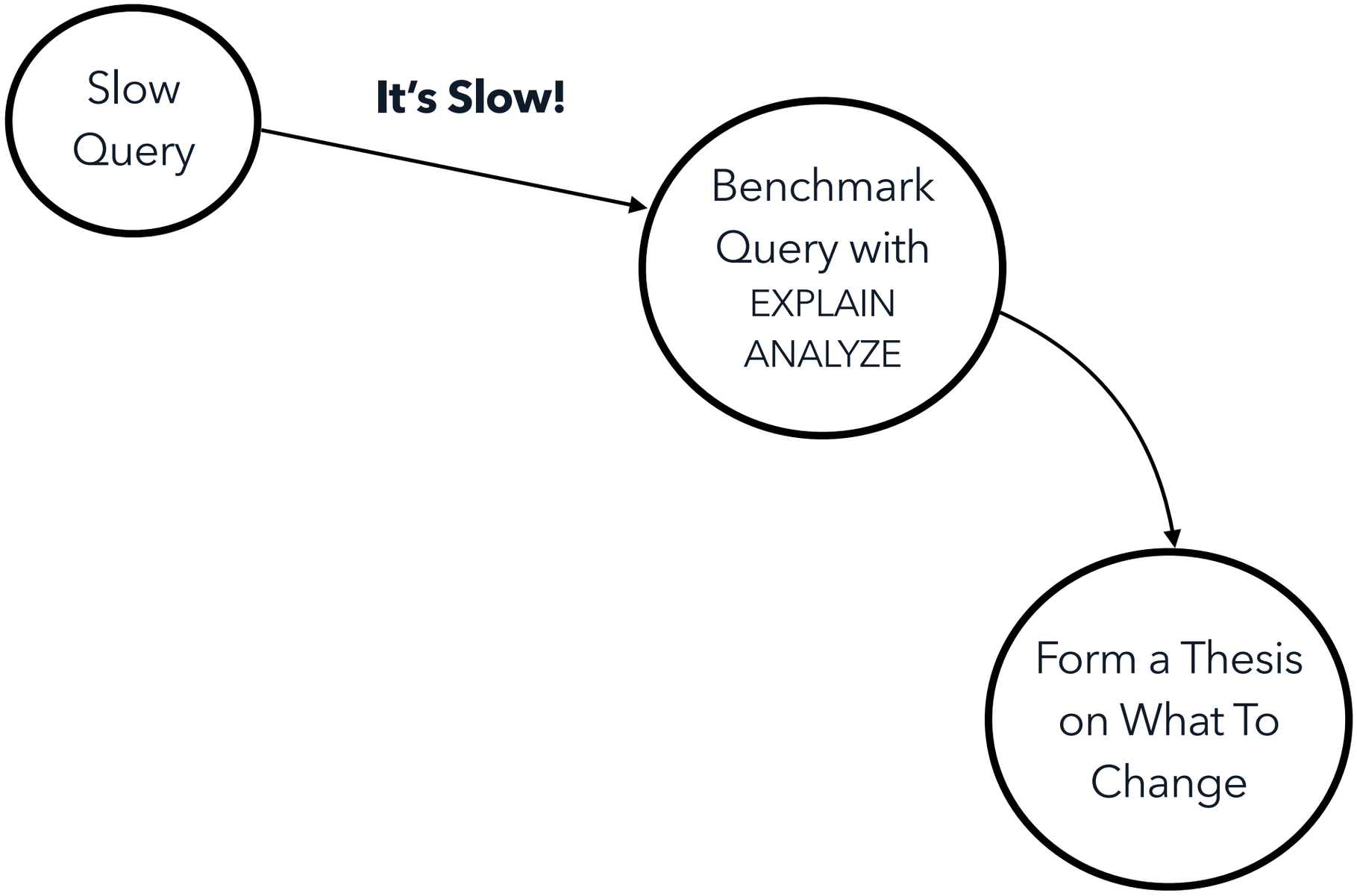
pganalyze

pganalyze

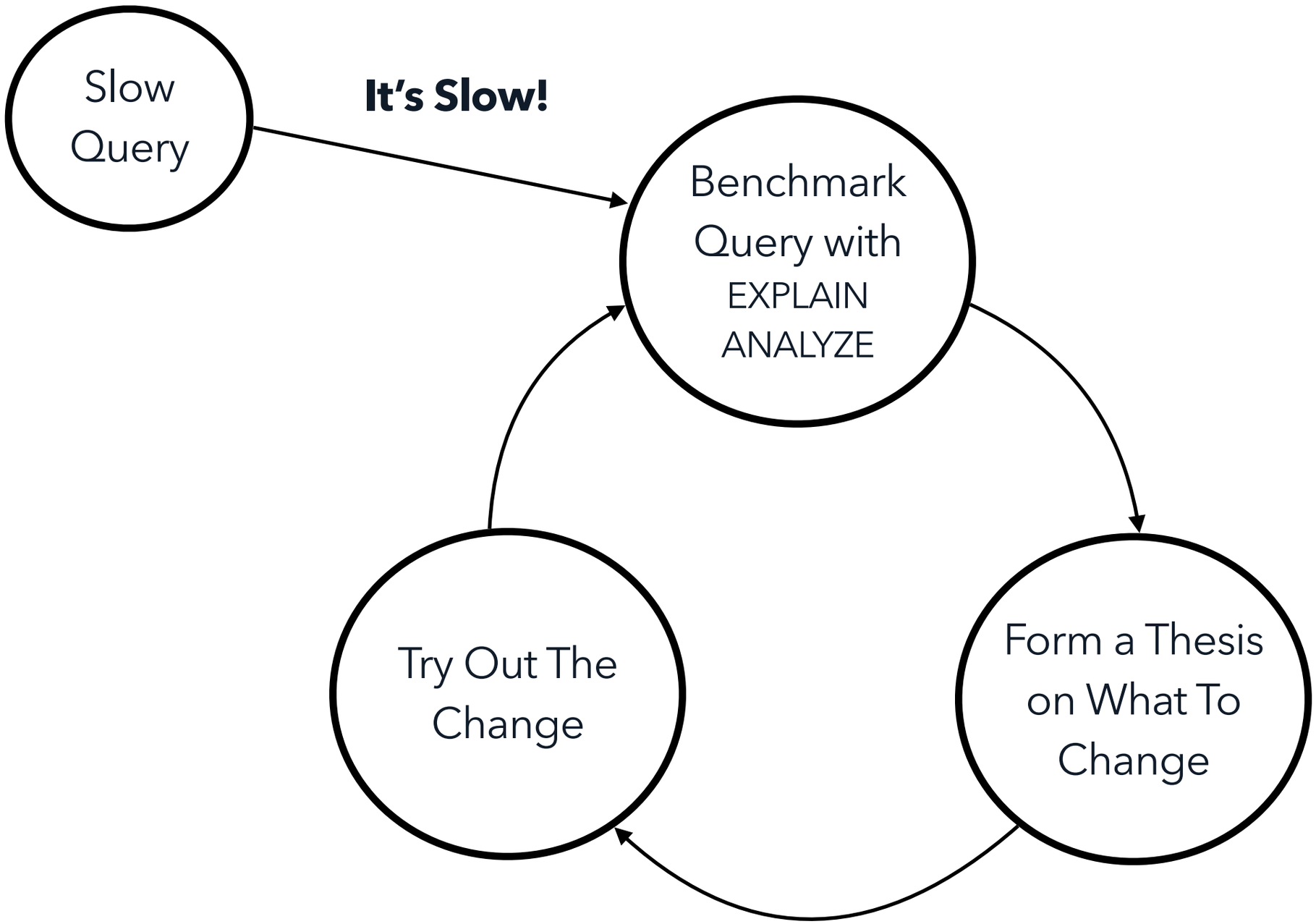
How to debug why a query is slow


Slow
Query
It's Slow!
Benchmark
Query with
EXPLAIN
ANALYZE

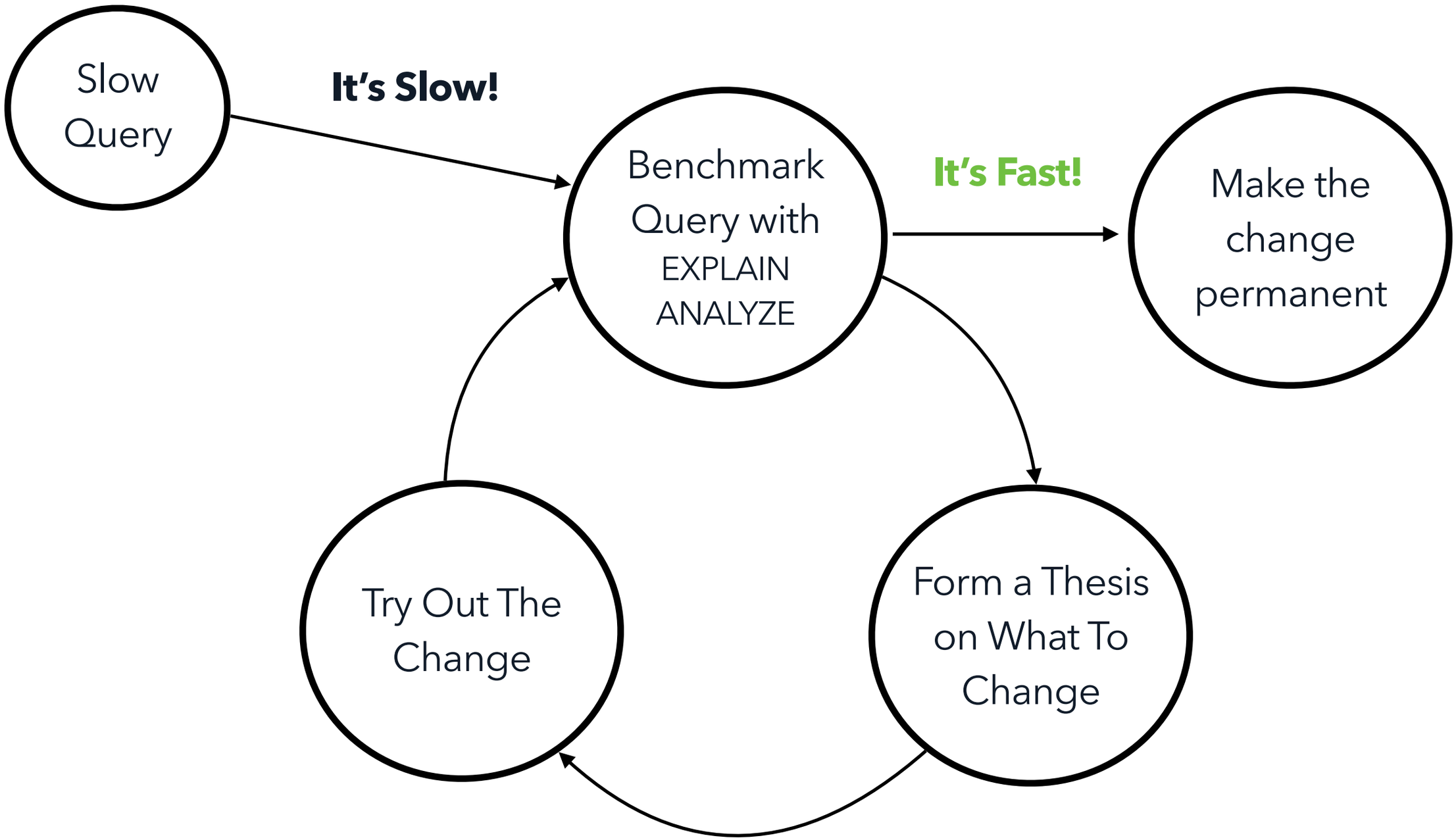

pganalyze


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
Form a Thesis on What To Change

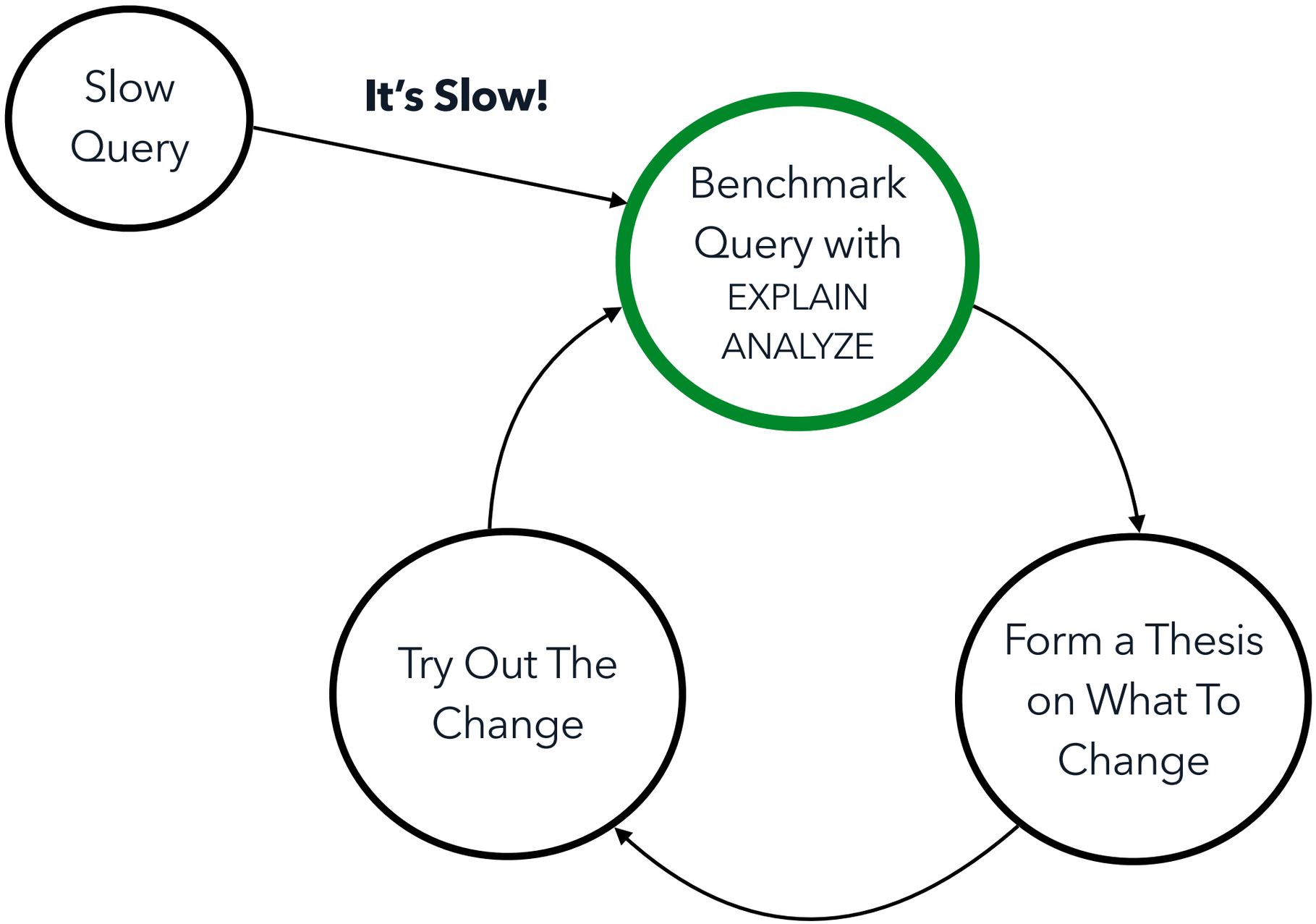
pganalyze


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
Form a Thesis on What To Change
Try Out The Change

pganalyze


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
It's Fast!
Make the change permanent
Form a Thesis on What To Change
Try Out The Change

pganalyze


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
Form a Thesis on What To Change
Try Out The Change

pganalyze

EXPLAIN without ANALYZE

= The plan the planner chose (but no actual statistics)

EXPLAIN (ANALYZE)

= The plan chosen + runtime statistics

EXPLAIN(ANALYZE, BUFFERS)

= The plan chosen + runtime statistics + I/O statistics

```
postgres=# EXPLAIN SELECT * FROM test WHERE c = 123;
                              QUERY PLAN
-----------------------------------------------------------------------
 Gather  (cost=1000.00..97366.28 rows=1 width=8)
   Workers Planned: 2
   ->  Parallel Seq Scan on test  (cost=0.00..96366.18 rows=1 width=8)
         Filter: (c = 123)
(4 rows)
```

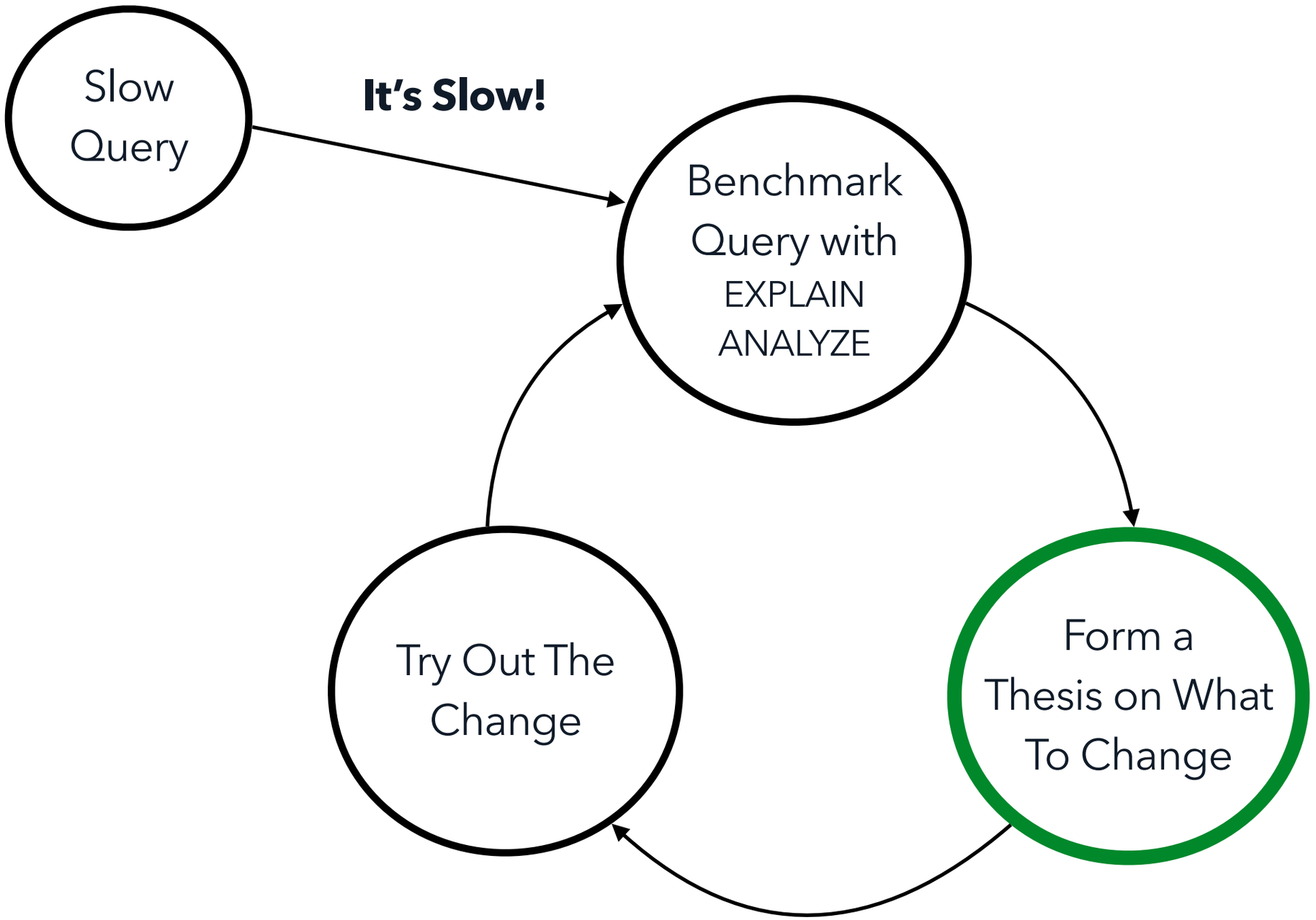
```
postgres=# EXPLAIN ANALYZE SELECT * FROM test WHERE c = 123;
                                                     QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..97366.28 rows=1 width=8) (actual time=307.117..307.328 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on test  (cost=0.00..96366.18 rows=1 width=8) (actual time=250.789..283.322 rows=0 loops=3)
         Filter: (c = 123)
         Rows Removed by Filter: 3333333
 Planning Time: 0.189 ms
 Execution Time: 307.371 ms
(8 rows)
```

```
postgres=# EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM test WHERE c = 456;
                                                     QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..97366.28 rows=1 width=8) (actual time=303.560..304.600 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   Buffers: shared hit=2757 read=41531
   I/O Timings: shared read=95.324
   ->  Parallel Seq Scan on test  (cost=0.00..96366.18 rows=1 width=8) (actual time=256.848..286.938 rows=0 loops=3)
         Filter: (c = 456)
         Rows Removed by Filter: 3333333
         Buffers: shared hit=2757 read=41531
         I/O Timings: shared read=95.324
 Planning Time: 0.231 ms
 Execution Time: 304.649 ms
(12 rows)
```

BUFFERS shows you the impact of the physical contents of the table (i.e. dead rows, empty space)

1 buffer = 8 kB buffer page

(on most Postgres installs)


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
Form a Thesis on What To Change
Try Out The Change

pganalyze

“The planner's task is fuzzy, there can be many valid plans for the same query, and its not always clear which one is best.”

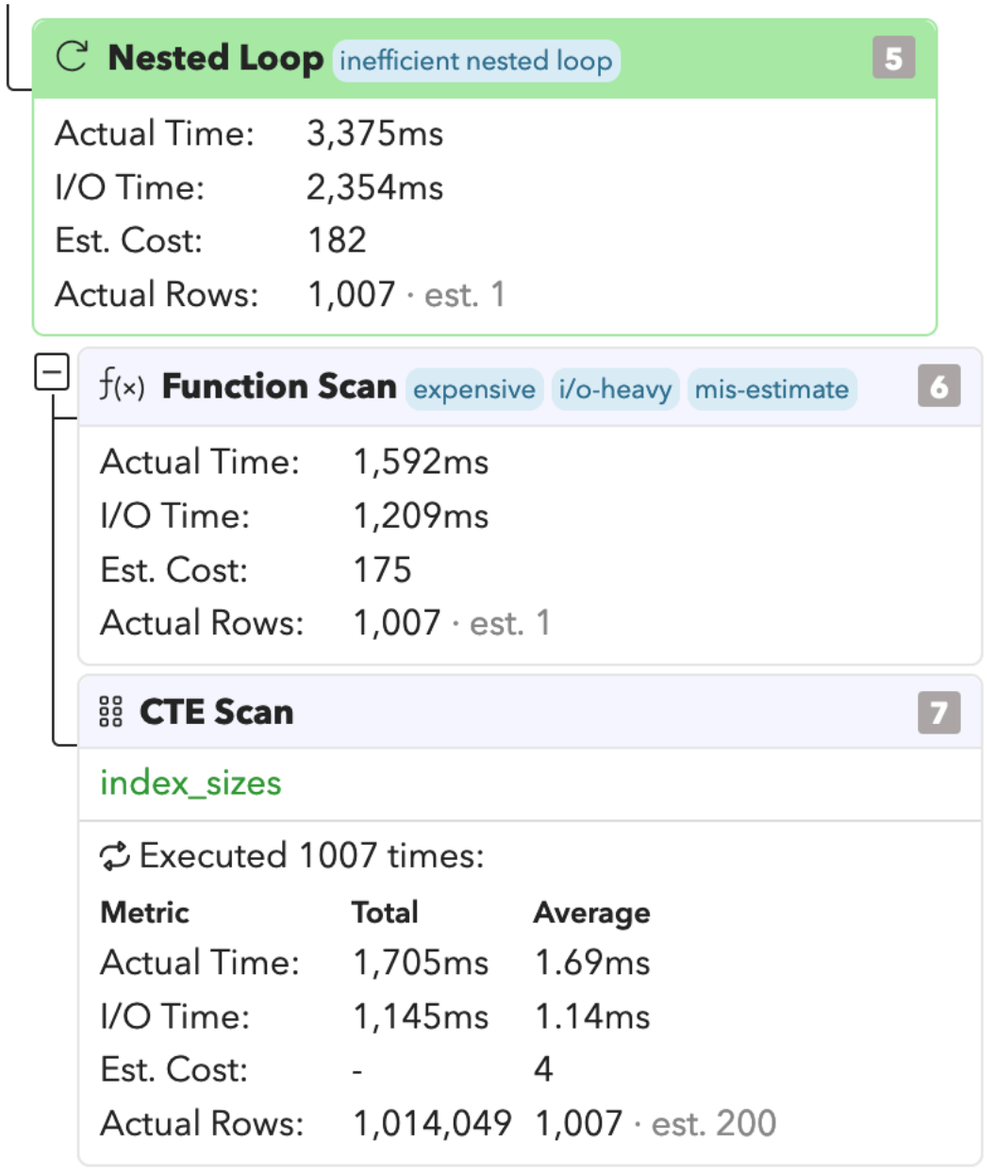
- Tom Lane in “Hacking the Query Planner” at PGCon ’11

Postgres planner responsibilities:

1. Find a good query plan.
2. Don't spend too much time (or memory) finding it.
3. Support the extensible aspects of Postgres.

What the planner doesn’t do:

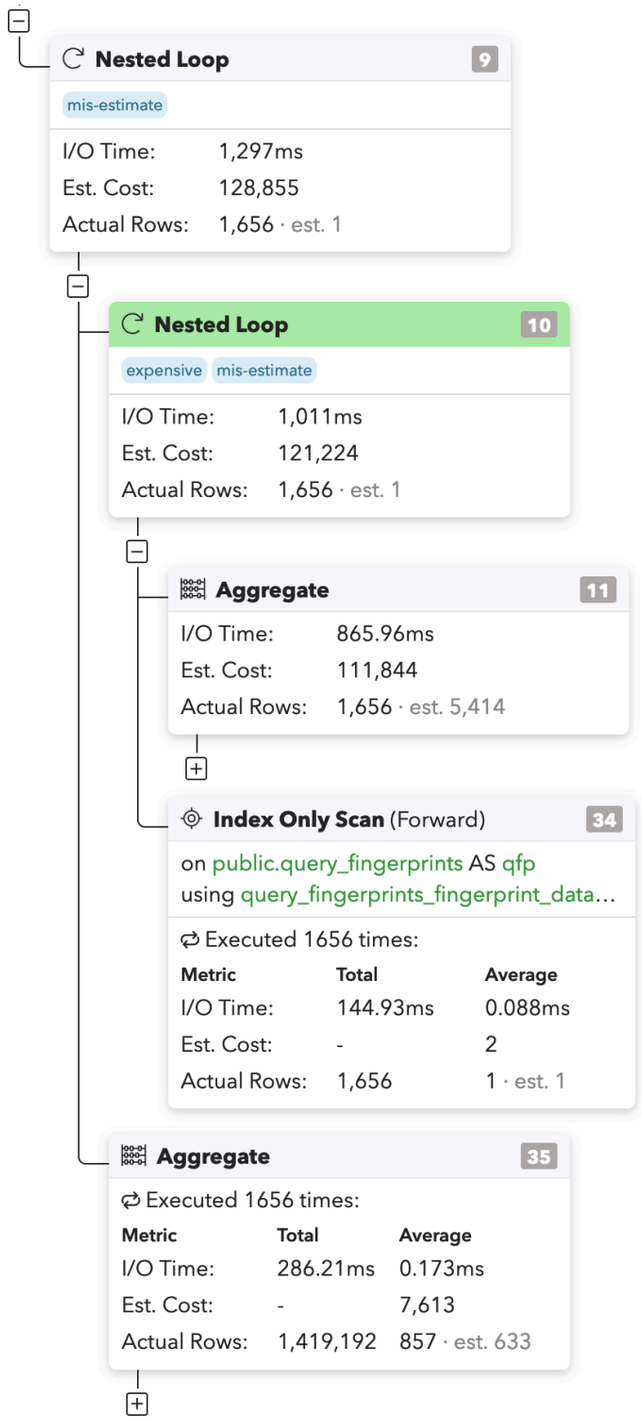
- Find all possible query plans
  (it discards seemingly worse plans quickly)
- Change a plan when its expectations don’t hold true
  (e.g. a lot more rows match than expected)
- Keep track of execution performance
  (it will happily keep producing slow queries)



New pganalyze EXPLAIN Insight: Inefficient Nested Loop

```
->  Nested Loop  (cost=0.25..181.76 rows=1 width=152)
                 (actual rows=1007)
```

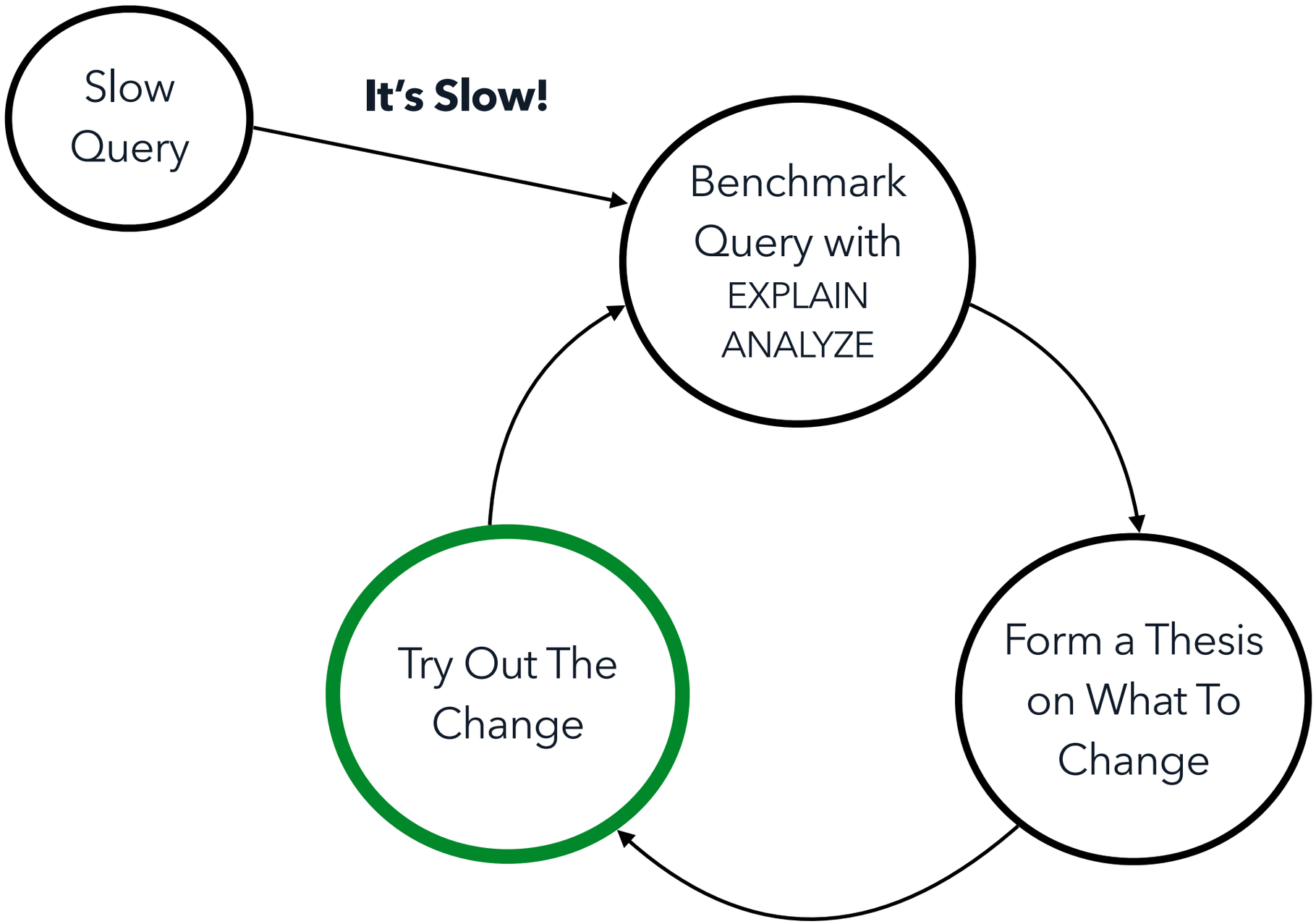
pganalyze



Both the lower Aggregate and the Index Only Scan had somewhat accurate row estimates.

But yet the Nested Loop estimate is wildly off, causing the upper Aggregate to run 1656 times, instead of the expected 1 time.

pganalyze


Slow Query
It's Slow!
Benchmark Query with EXPLAIN ANALYZE
Form a Thesis on What To Change
Try Out The Change

pganalyze

To Understand

Why A “Bad” Plan Was Chosen

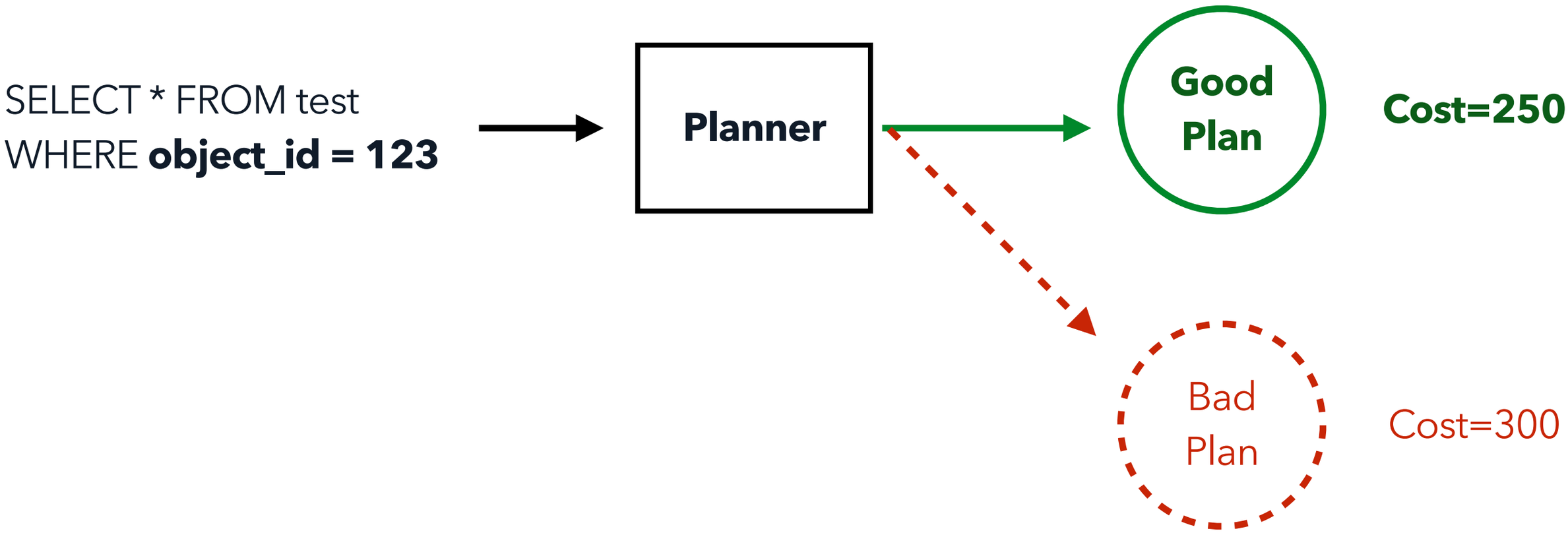
Start By Forcing The Good Plan

pganalyze

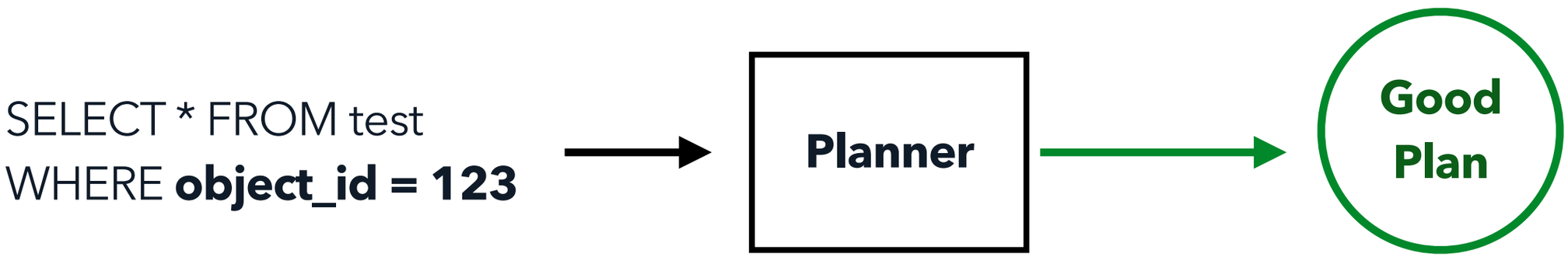
Benchmarking alternate plans with settings & hints

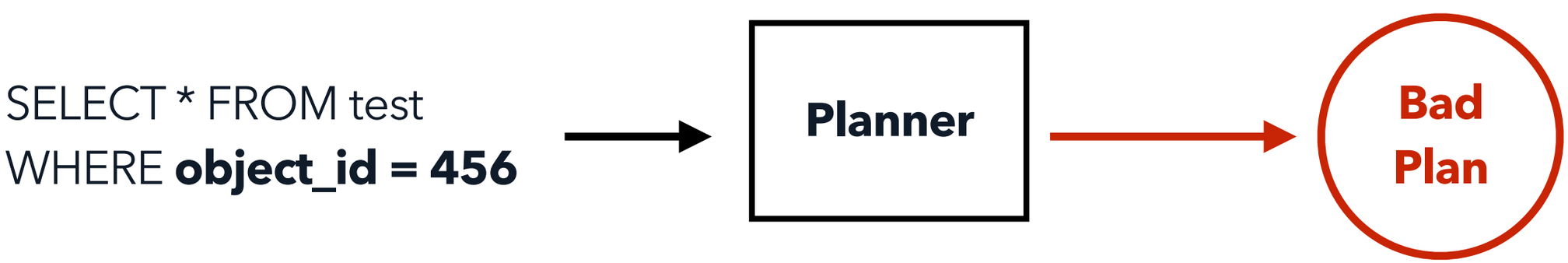
SELECT * FROM test
WHERE **object_id = 123**

pganalyze


SELECT * FROM test
WHERE **object_id = 123**
Planner
Good Plan
Cost=250
Bad Plan
Cost=300

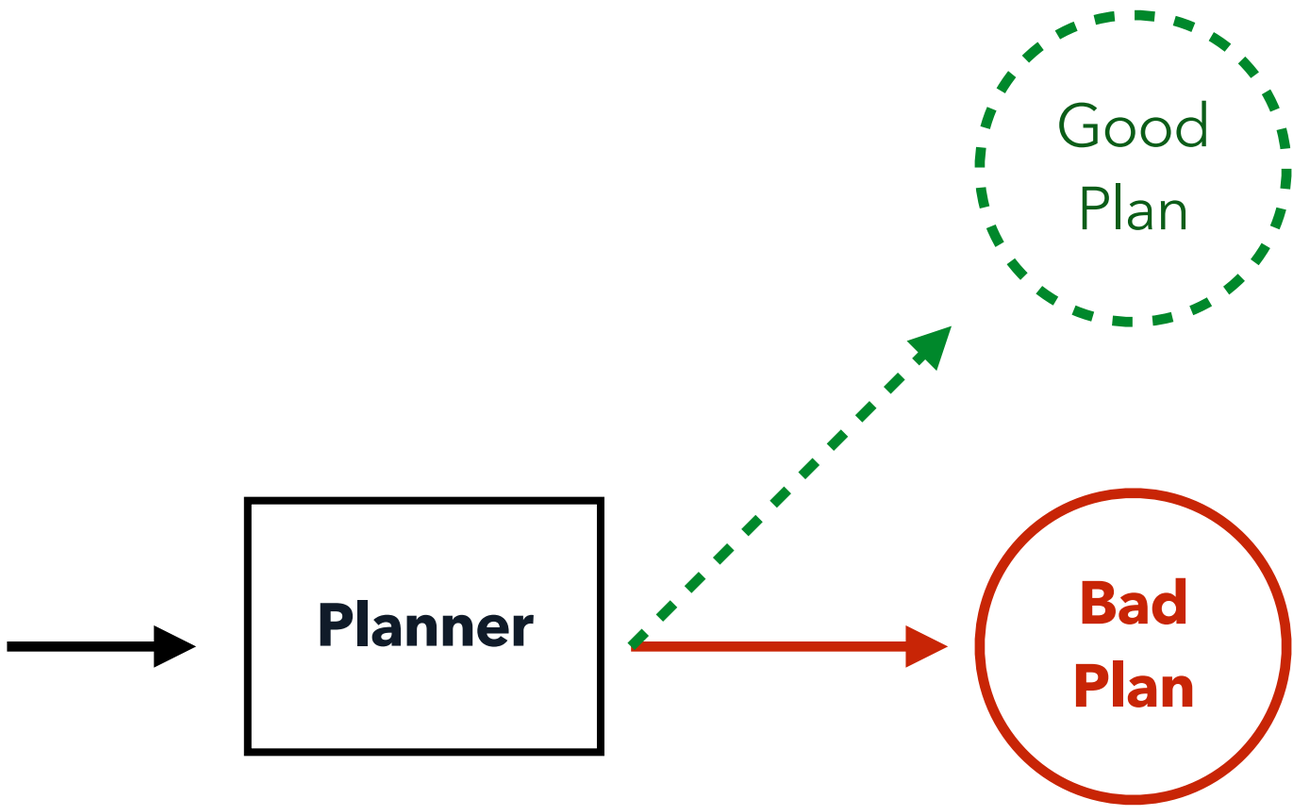
pganalyze


SELECT * FROM test
WHERE **object_id = 123**
Planner
Good
Plan

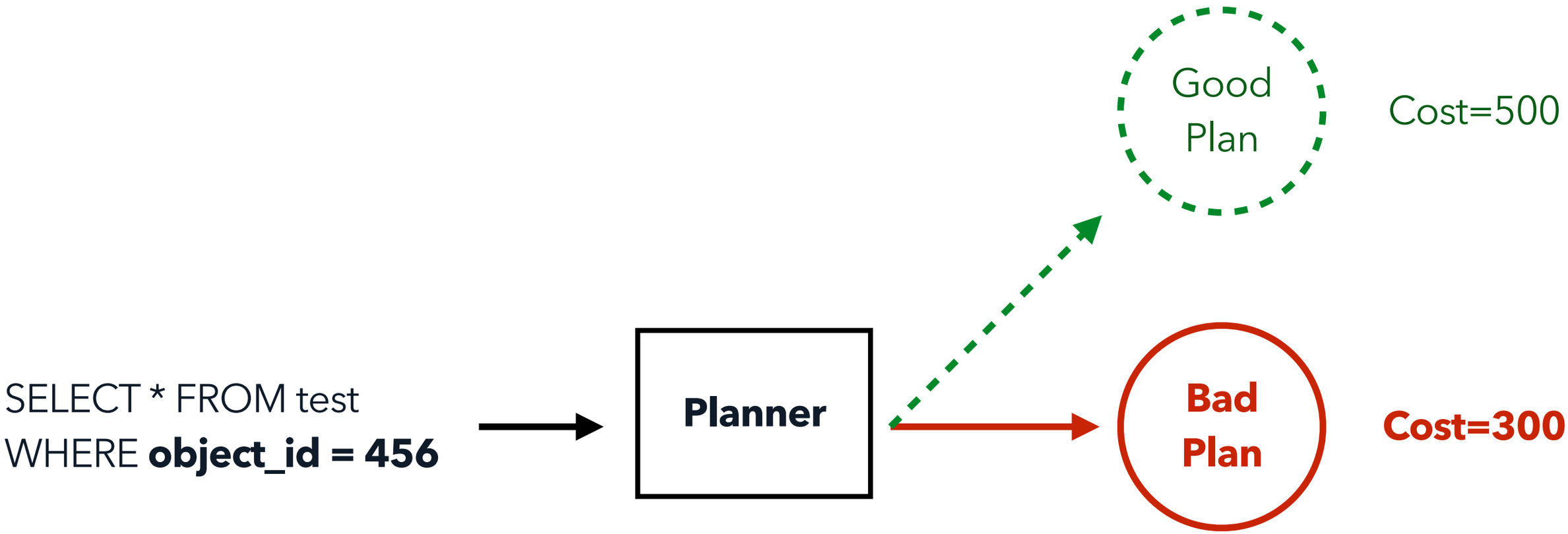

SELECT * FROM test
WHERE **object_id = 456**
Planner
Bad
Plan

pganalyze

SELECT * FROM test
WHERE **object_id = 456**



pganalyze


Good
Plan
Cost=500
SELECT * FROM test
WHERE **object_id = 456**
Planner
**Bad
Plan**
Cost=300

pganalyze

The easiest test:

If your bad plan
involves a **planner feature**,
turn it off.

Seq Scan

Cost=300

Index Scan

Cost=500

SET enable_seqscan = off

Seq Scan

Cost=10000000000.00

Index Scan

Cost=500

pganalyze

20.7.1. Planner Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a *temporary* solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways to improve the quality of the plans chosen by the optimizer include adjusting the planner cost constants (see **Section 20.7.2**), running `ANALYZE` manually, increasing the value of the **default_statistics_target** configuration parameter, and increasing the amount of statistics collected for specific columns using `ALTER TABLE SET STATISTICS`.

`enable_async_append (boolean)`

Enables or disables the query planner's use of async-aware append plan types. The default is `on`.

`enable_bitmapscan (boolean)`

Enables or disables the query planner's use of bitmap-scan plan types. The default is `on`.

`enable_gathermerge (boolean)`

Enables or disables the query planner's use of gather merge plan types. The default is `on`.

`enable_hashagg (boolean)`

Enables or disables the query planner's use of hashed aggregation plan types. The default is `on`.

`enable_hashjoin (boolean)`

Enables or disables the query planner's use of hash-join plan types. The default is `on`.

`enable_incremental_sort (boolean)`

Enables or disables the query planner's use of incremental sort steps. The default is `on`.

`enable_indexscan (boolean)`

Enables or disables the query planner's use of index-scan plan types. The default is `on`.

`enable_indexonlyscan (boolean)`

Enables or disables the query planner's use of index-only-scan plan types (see **Section 11.9**). The default is `on`.

`enable_material (boolean)`

Enables or disables the query planner's use of materialization. It is impossible to suppress materialization entirely, but turning this variable off prevents the planner from inserting materialize nodes except in cases where it is required for correctness. The default is `on`.

`enable_memoize (boolean)`

Enables or disables the query planner's use of memoize plans for caching results from parameterized scans inside nested-loop joins. This plan type allows scans to the underlying plans to be skipped when the results for the current parameters are already in the cache. Less commonly looked up results may be evicted from the cache when more space is required for new entries. The default is `on`.

Source: Postgres Documentation - Query Planning

Once you have the right plan,
look at the individual plan nodes
and find out where the
cost mis-estimate originates

If you see a **Hash** or **Merge Join** being used instead of a **Nested Loop** + **Parameterized Index Scan**, try:

```
SET enable_mergejoin = off;
SET enable_hashjoin = off;
```

For more complicated cases,

Utilize pg_hint_plan to force the good plan

(to find the root cause of the cost mis-estimate)

```
EXPLAIN SELECT EXISTS (
  SELECT 1 FROM schema_column_stats scs WHERE scs.invalidated_at_snapshot_id IS NULL AND scs.table_id IN (
    SELECT id FROM schema_tables WHERE invalidated_at_snapshot_id IS NULL AND database_id = 12345));

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
Result  (cost=9.13..9.14 rows=1 width=1)
   InitPlan 1 (returns $1)
     ->  Nested Loop  (cost=1.00..971672.56 rows=119623 width=0)
           ->  Index Only Scan using index_schema_column_stats_on_table_id on schema_column_stats scs
               (cost=0.43..372676.50 rows=23553966 width=8)
           ->  Memoize  (cost=0.57..0.61 rows=1 width=8)
                 Cache Key: scs.table_id
                 Cache Mode: logical
                 ->  Index Scan using schema_tables_pkey on schema_tables  (cost=0.56..0.60 rows=1 width=8)
                       Index Cond: (id = scs.table_id)
                       Filter: ((invalidated_at_snapshot_id IS NULL) AND (database_id = 12345))
```

Bad plan, with join order = (schema_column_stats schema_tables)

```
SET enable_memoize = off;

EXPLAIN SELECT EXISTS (
  SELECT 1 FROM schema_column_stats scs WHERE scs.invalidated_at_snapshot_id IS NULL AND scs.table_id IN (
    SELECT id FROM schema_tables WHERE invalidated_at_snapshot_id IS NULL AND database_id = 12345));

                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------
Result  (cost=13.13..13.14 rows=1 width=1)
   InitPlan 1 (returns $1)
     ->  Nested Loop  (cost=0.99..1451807.35 rows=119623 width=0)
           ->  Index Scan using schema_tables_database_id_schema_name_table_name_idx on schema_tables
               (cost=0.56..37778.03 rows=34753 width=8)
     Index Cond: (database_id = 12345)
->  Index Only Scan using index_schema_column_stats_on_table_id on schema_column_stats scs
    (cost=0.43..26.68 rows=1401 width=8)
     Index Cond: (table_id = schema_tables.id)
```

Good plan, with join order = (schema_tables schema_column_stats)

```
/*+ Leading((scs schema_tables)) IndexOnlyScan(scs index_schema_column_stats_on_table_id) IndexScan(schema_tables
schema_tables_pkey) Set(enable_memoize off) */
EXPLAIN SELECT EXISTS (
  SELECT 1 FROM schema_column_stats scs WHERE scs.invalidated_at_snapshot_id IS NULL AND scs.table_id IN (
    SELECT id FROM schema_tables WHERE invalidated_at_snapshot_id IS NULL AND database_id = 12345));

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
Result  (cost=122.90..122.91 rows=1 width=1)
   InitPlan 1 (returns $1)
     ->  Nested Loop  (cost=0.99..14582869.23 rows=119623 width=0)
           ->  Index Only Scan using index_schema_column_stats_on_table_id on schema_column_stats scs
               (cost=0.43..372676.50 rows=23553966 width=8)
           ->  Index Scan using schema_tables_pkey on schema_tables  (cost=0.56..0.60 rows=1 width=8)
                 Index Cond: (id = scs.table_id)
                 Filter: ((invalidated_at_snapshot_id IS NULL) AND (database_id = 12345))
```

Bad plan, with join order = (schema_tables schema_column_stats)

Good plan:

1,451,807 cost

```
->  Nested Loop  (cost=0.99..1451807.35 rows=119623 wid
      ->  Index Scan using schema_tables_database_id_sc
          (cost=0.56..37778.03 rows=34753 width=8)
```

Bad plan without Memoize:

14,582,869 cost

```
->  Nested Loop  (cost=0.99..14582869.23 rows=119623 wi
      ->  Index Only Scan using index_schema_column_sta
          (cost=0.43..372676.50 rows=23553966 width=8)
```

Bad plan with Memoize:

971,672 cost

```
->  Nested Loop  (cost=1.00..971672.56 rows=119623 widt
      ->  Index Only Scan using index_schema_column_sta
          (cost=0.43..372676.50 rows=23553966 width=8)
```

DB column stats check: Add filter on server_id to improve performance #2693

Edit | Code

Merged | lfittl merged 1 commit into `main` from `improve-get-column-stats-helper-check` 3 weeks ago

Conversation 0 | Commits 1 | Checks 3 | Files changed 3 | +19 −5

lfittl commented last month • edited

The previous query was producing one of two plans in practice:

(1)

```
NestedLoop(schema_column_stats schema_tables)
- IndexScan(schema_tables_database_id_schema_name_table_name_idx)
  Index Cond: (database_id = $1)
- IndexOnlyScan(index_schema_column_stats_on_table_id)
   Index Cond: (table_id = schema_tables.id)
```

(2)

```
NestedLoop(schema_column_stats schema_tables)
- IndexOnlyScan(index_schema_column_stats_on_table_id)
  Index Cond: -
- Memoize
-- IndexScan(schema_tables_pkey)
   Index Cond: (id = schema_column_stats.table_id)
   Filter: (database_id = $1)
```

Plan (1) is the right choice here, however in the pathological case this is not chosen, due to an overestimate on the number of matching rows in schema_tables (~40k instead of 100).

Plan (2) appears to happen because the Memoize costing calculates a cache rate of ~95%, and thus makes the many iterations over schema_tables very cheap.

After multiple fruitless attempts at fixing the estimation for (1), instead make the plan with Memoize behave less bad, by introducing a filter on server_id resulting in one of these two plan choices:

Reviewers

msakrejda ✓

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging this pull request may close these issues.

Index Advisor overview: Investigate issue wit...

Notifications Customize

Unsubscribe

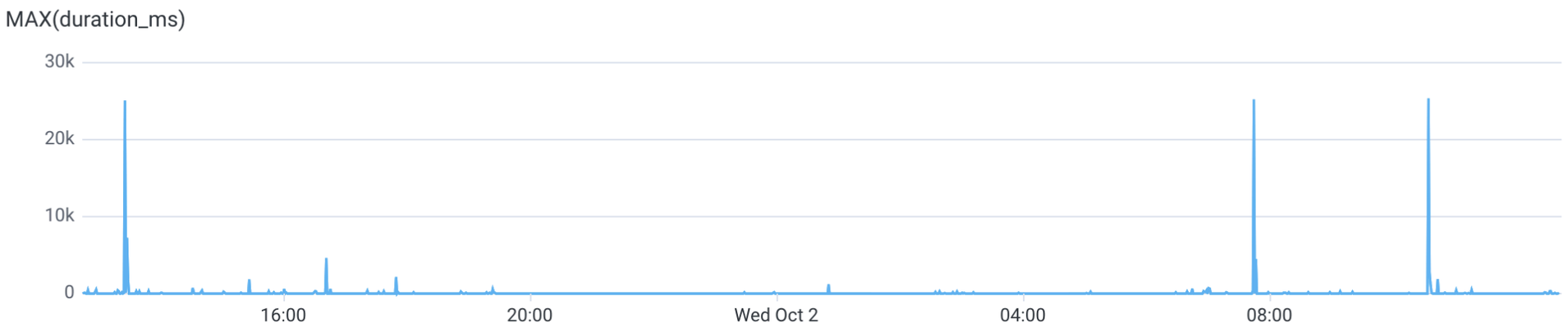
You're receiving notifications because you're watching this repository.

pganalyze

pganalyze

Query Tuning with pganalyze

Let’s start with a trace of a slow web request

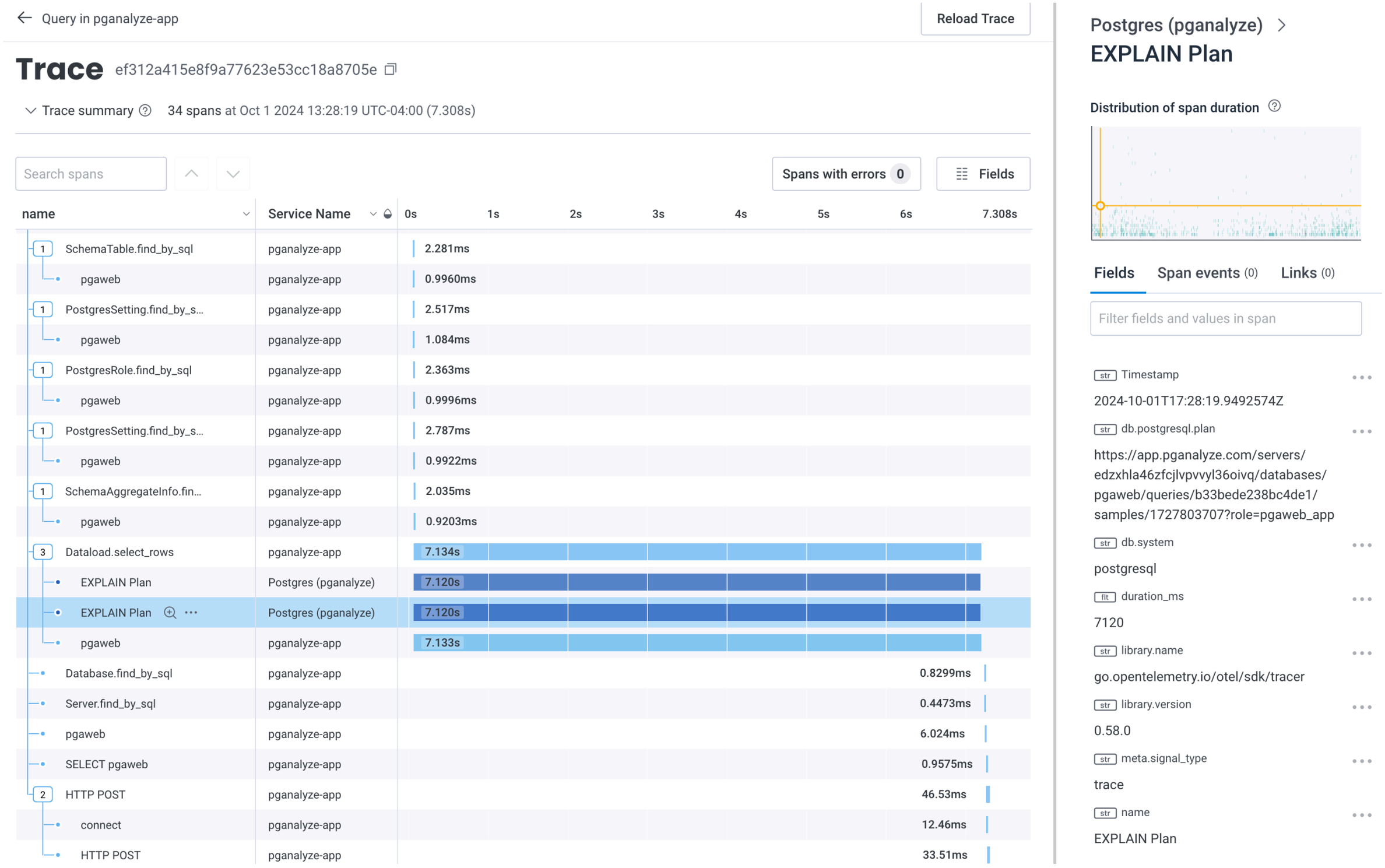


Overview | BubbleUp | Correlations | **Traces** | Explore Data

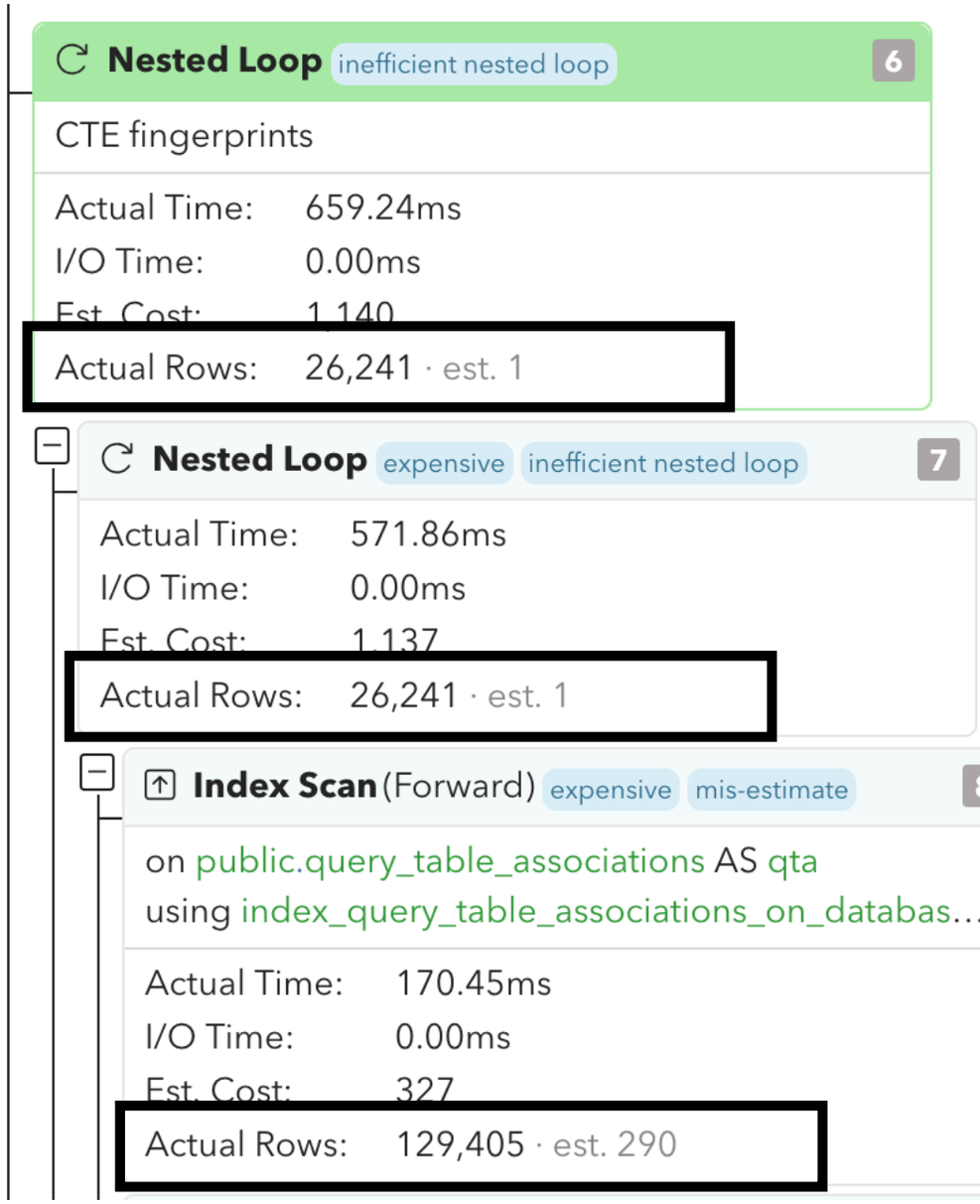
Shows up to 10 traces with the slowest spans from the selected time range. Learn more.

	Root Service Name	Root Name	Root Duration ms	Number of Spans	Span Summary	Trace ID
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	2,772.77562	26		8d59171091ac7ee7f4f5382d2754027c
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	2,133.37864	26		c0c4d95a6dd4647637b248a0a6161a29
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	25,356.02089	30		ec2decbb788ce9eaaae1d9d3b6bf1625
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	2,710.38595	28		a3278f71c6837a281da62551e7c9645a
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	4,484.71493	34		87856ed9c2651500187ef9bdd690d387
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	25,220.97727	29		60a1ce3242e9aebf397d32f03d2620dd
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	2,132.79932	30		f6dee27f224f8c0f0daabf3313c9bde4
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	4,626.37905	30		28a4c4ff9bd368d6cc42814d760b1391
	pganalyze-app	Api::GraphqlController#graphql (SchemaTableQueries)	7,257.8716	34		ef312a415e8f9a77623e53cc18a8705e

Let’s start with a trace of a slow web request



Multiple Mis-Estimates of Nested Loops



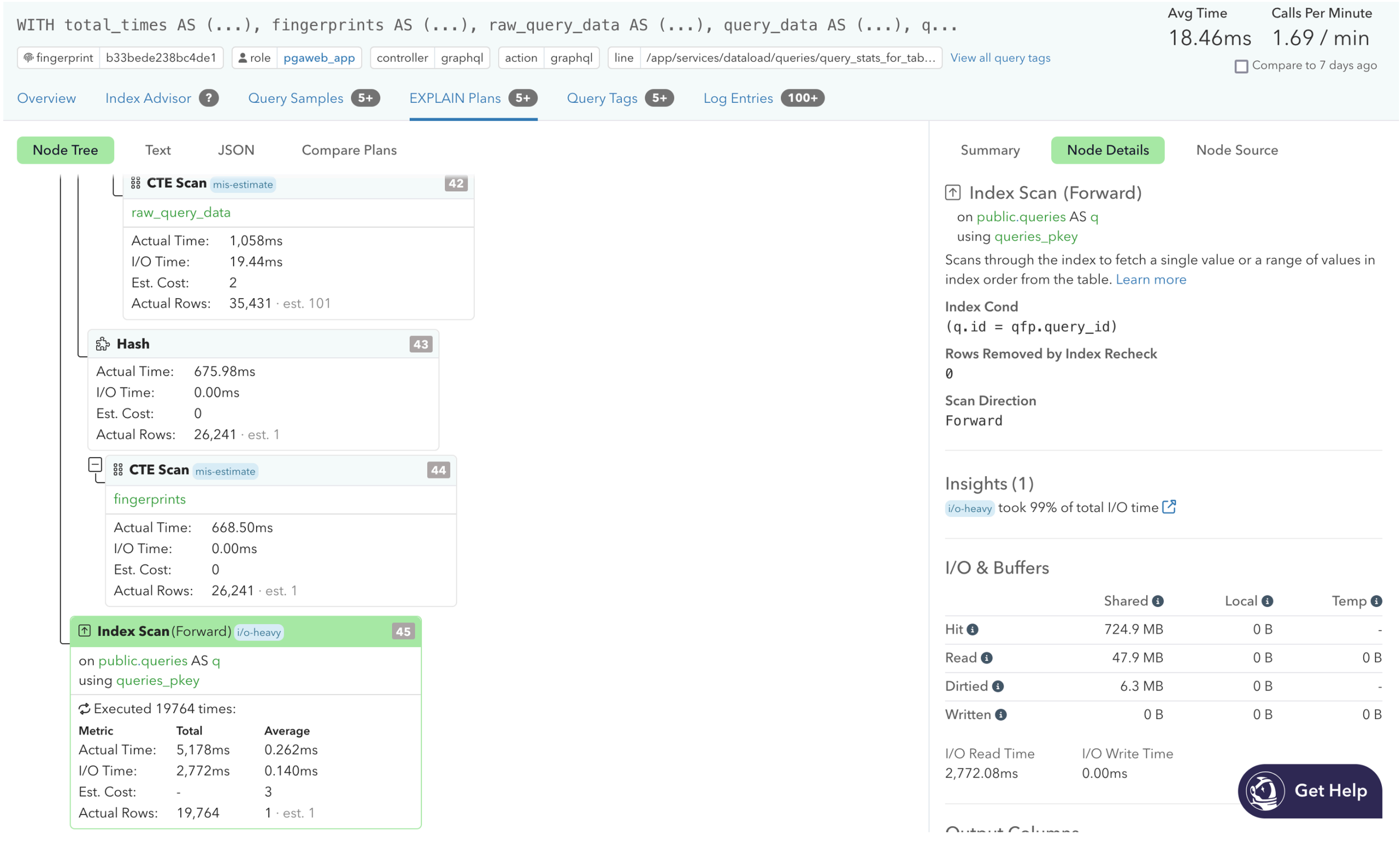
Under Estimate

Under Estimate

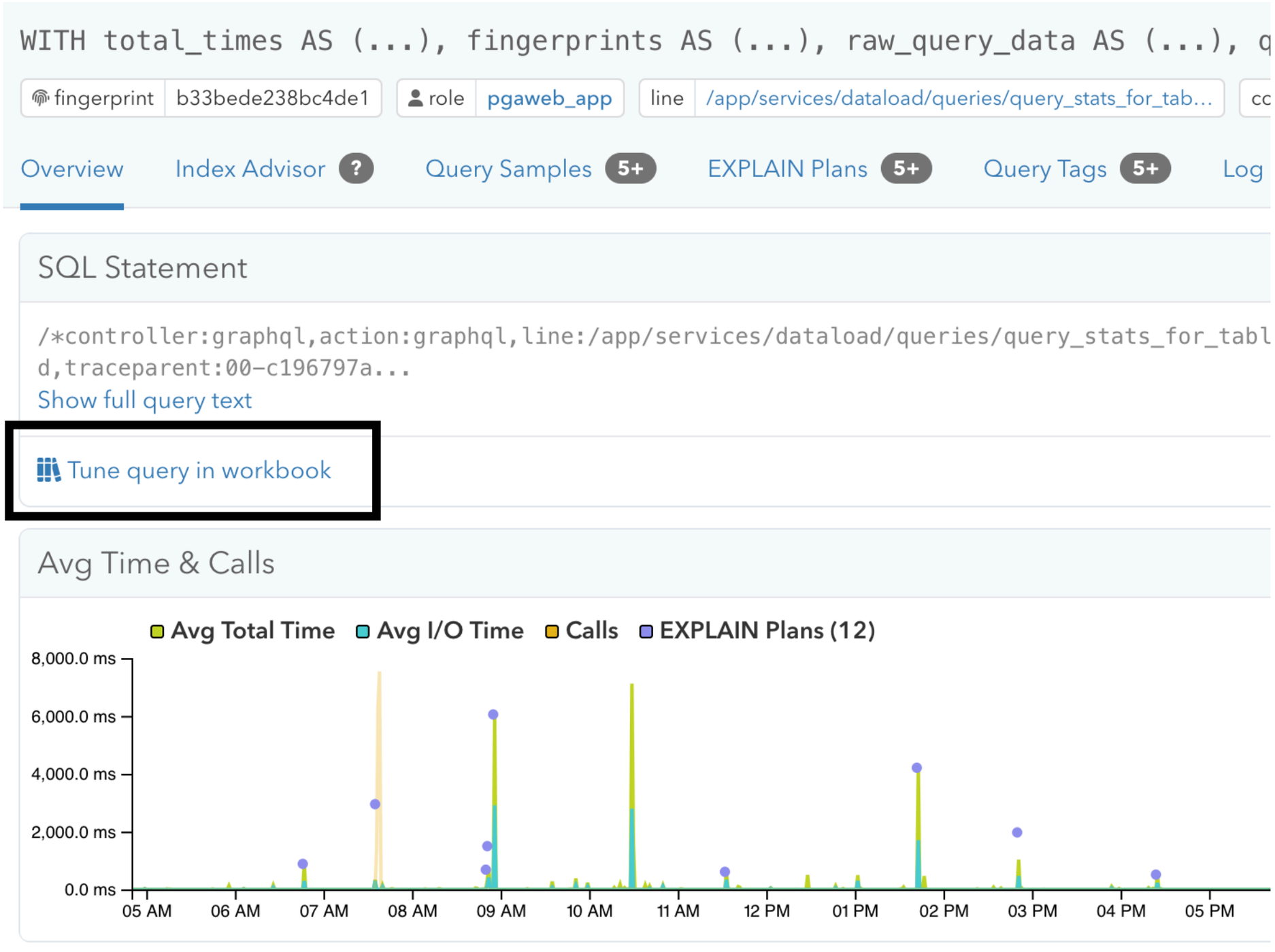
Under Estimate

pganalyze

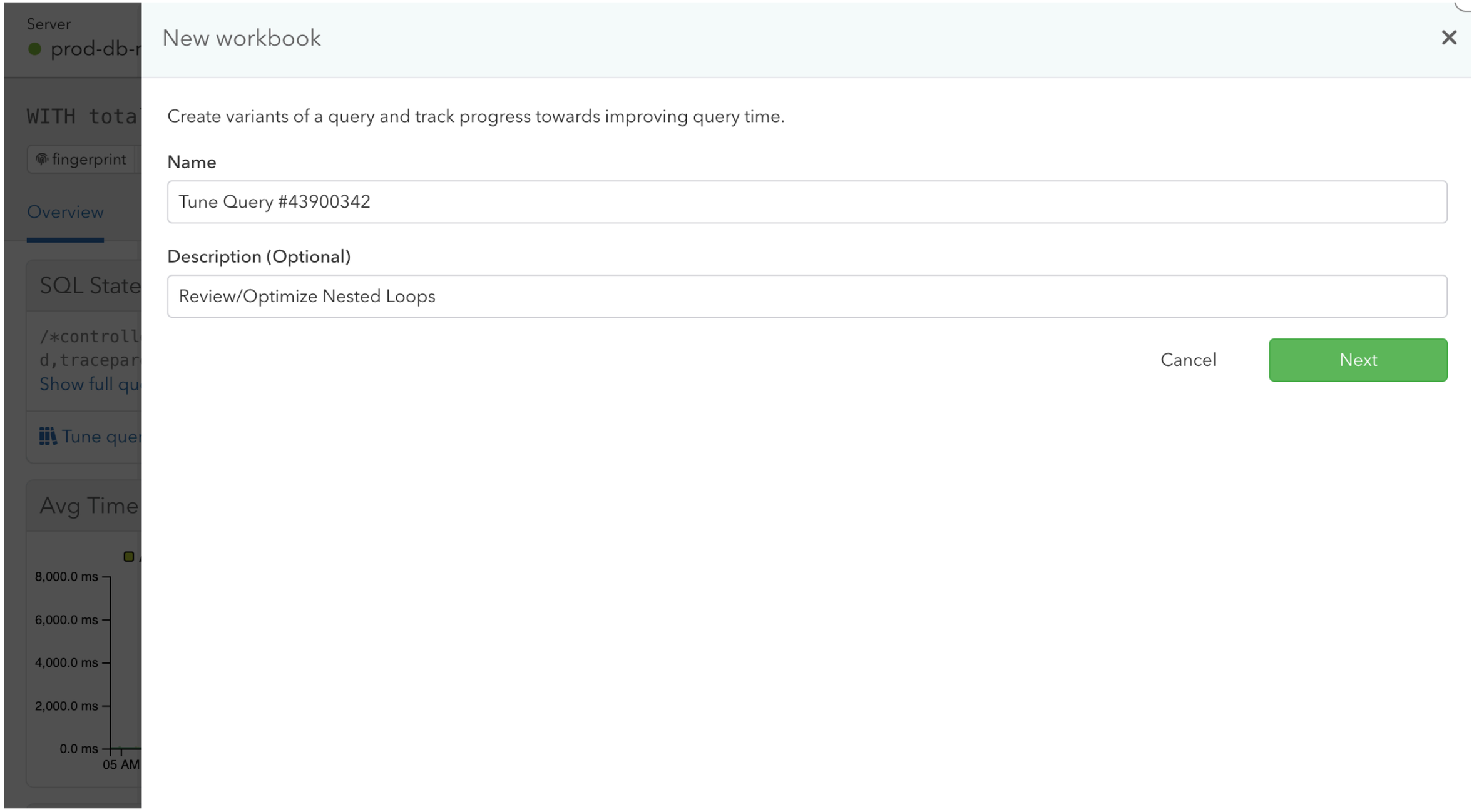
Index Scan in a Loop takes 99% of I/O Time



Let's Tune The Query!



Let’s Tune The Query!



pganalyze

Automatic Naming of Parameters

Tune Query #43900342

```
/*controller:graphql,action:graphql,line:/app/services/dataload/queries/query_stats_for_table.r
b:181:in `query_stats_for_table',sentry_trace_id:11c0590f5359469fbc1dd94a99fbe18d,traceparent:00-
c196797ad1cd8128c0baf25162809ad4-c3722293bedf5213-01,tracestate:pganalyze=t:1727812015.8315823*/
WITH total_times AS (
SELECT SUM(query_stats_blk_read_time_sum + query_stats_blk_write_time_sum) AS total_iotime,
SUM(query_stats_total_time_sum) AS total_runtime
FROM query_overview_stats_35d qos
WHERE qos.database_id = $database_id AND qos.collected_at BETWEEN $collected_at_3 AND $collected_
at_4
),
fingerprints AS (
SELECT qf.*
FROM query_table_associations qta
JOIN query_occurrences o ON o.query_id = qta.query_id AND o.database_id = $database_id_6 AND o.la
st >= $param_10::date
JOIN query_fingerprints qf ON qf.query_id = qta.query_id
WHERE qta.database_id = $database_id_2
AND table_name IN ($table_name, $param_11 || $param_12 || $param_9)
),
```

Paste a query sample to extract parameters

Custom query

```
WITH total_times AS (
SELECT SUM(query_stats_blk_read_time_sum + query_stats_blk_write_time_sum) AS total_iotime,
SUM(query_stats_total_time_sum) AS total_runtime
FROM query_overview_stats_35d qos
WHERE qos.database_id = [redacted] AND qos.collected_at BETWEEN '2024-09-28 04:30:00' AND '2024-09-28
14:30:00'
),
```

Add parameters from query

Add parameters manually

pganalyze

Or use parameters from Query Samples

Query #43899395

```
/*controller:graphql,action:graphql,line:/app/services/dataload/queries/query_stats_for_query.rb:121:in `query_stats_for_query',sentry_trace_
id:46dc62c6c56640459260db465f758cf1,traceparent:00-945d5f07...
```

Show full query text

Select Parameters from Query Samples

Search parameters...

QUERY SAMPLE	PLAN FINGERPRINT	RUNTIME
☐ `$calls = 0, $database_id = 23, $database_id_2 = 23, $query_id = 43900138, $database_id_3 = 23, $database_i` 2024-11-06 05:03:04pm PST	ca19ca8	150.98ms
☑ `$calls = 0, $database_id = 23, $database_id_2 = 23, $query_id = 43934683, $database_id_3 = 23, $database_i` 2024-11-08 03:13:53am PST	c3099cd	138.15ms
☑ `$calls = 0, $database_id = 23, $database_id_2 = 23, $query_id = 43914630, $database_id_3 = 23, $database_i` 2024-11-08 07:14:44am PST	c3099cd	132.03ms

› Custom Parameters

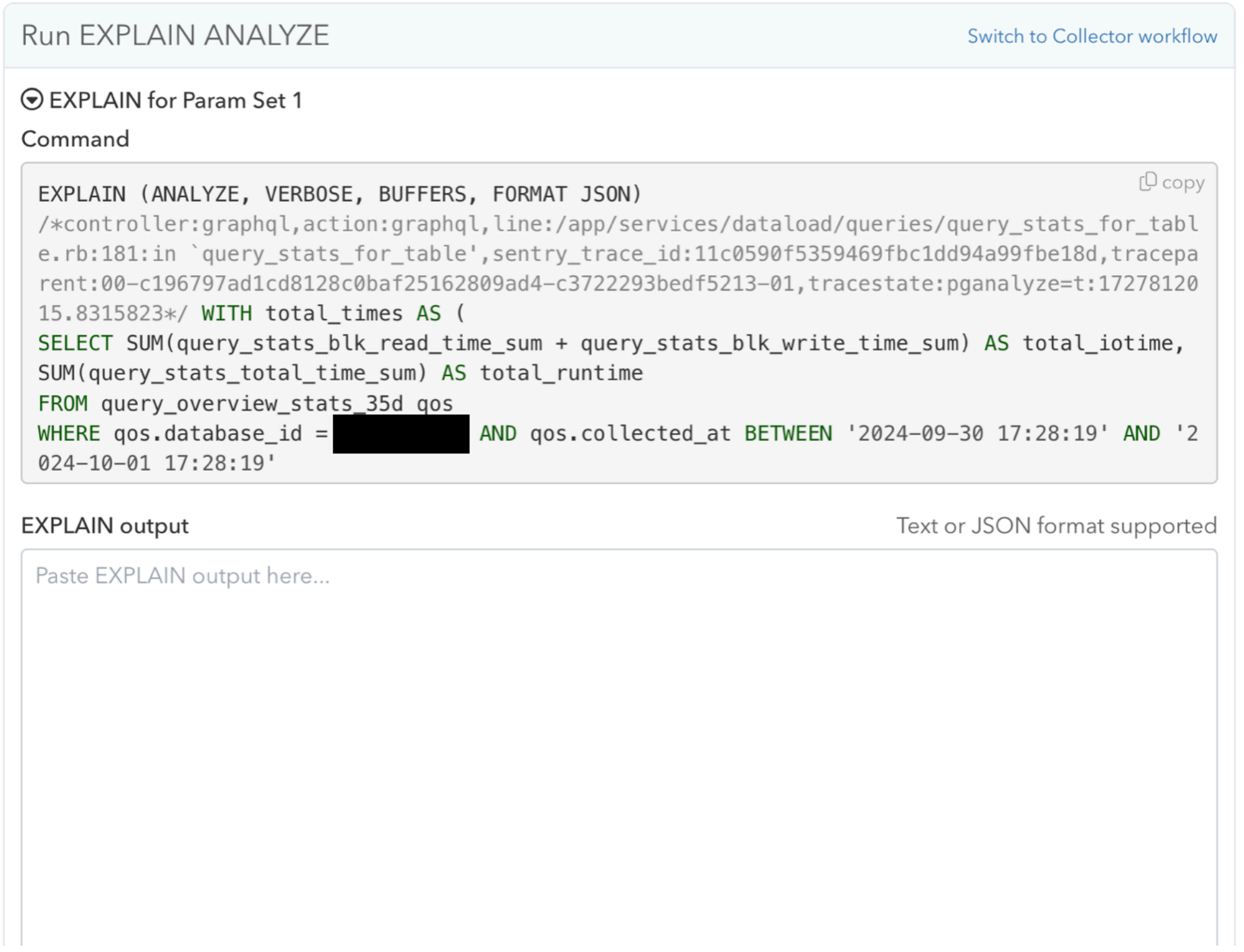
2 parameter sets selected

Cancel creation

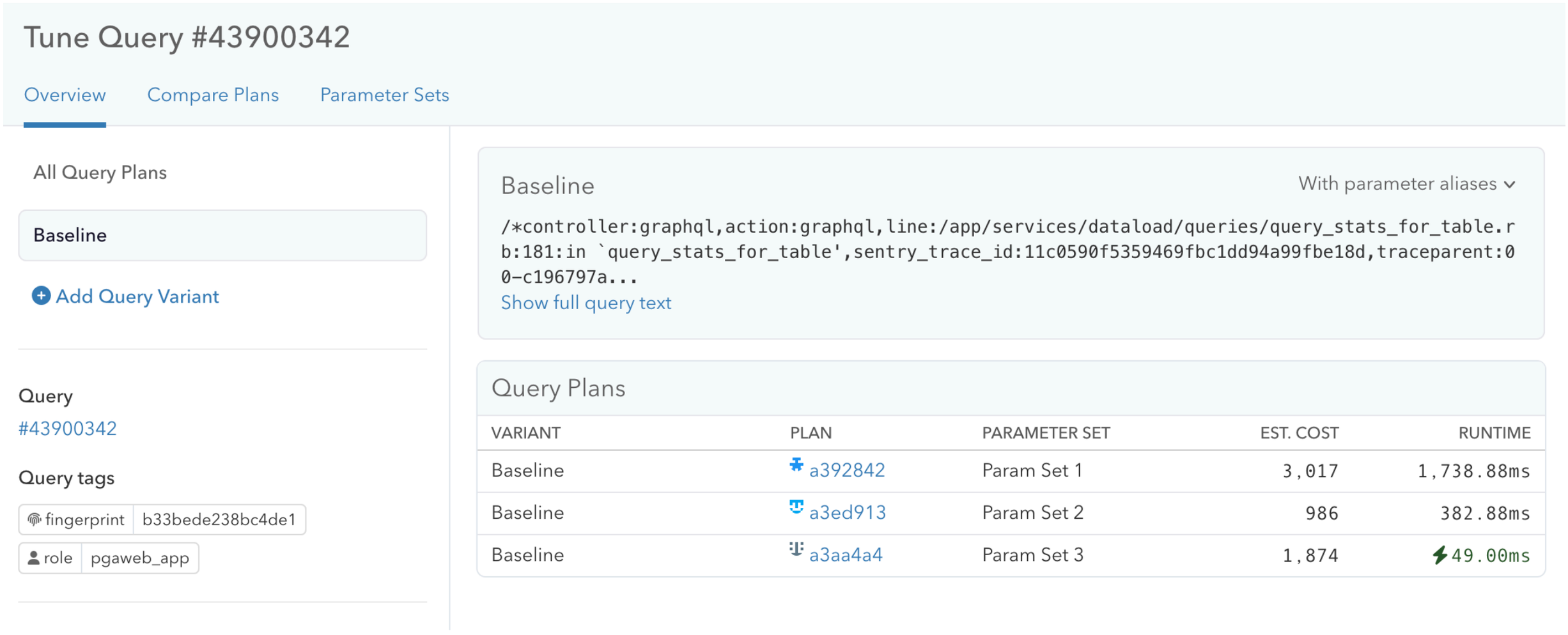
Run EXPLAIN...

pganalyze

Benchmark the same query, with different parameters



We've recorded the Baseline



pganalyze

Why are the plans different?

Cost Metric: ○ Est. Total Cost (Self) ○ Runtime (Self) ○ I/O Read Time (Self) ◉ Rows

Plan A	Plan B	Plan A: Rows	Plan B: Rows
-> Limit	-> Limit	100	23
-> Aggregate	-> Aggregate	1	1
-> Append	-> Append	1,440	1,440
-> Index Scan	-> Index Scan	391	36
-> Index Scan	-> Index Scan	1,049	1,404
-> Nested Loop	-> Nested Loop	31,973	56
-> Nested Loop	-> Nested Loop	31,973	244
-> Index Scan	-> Index Scan	128,992	244
-> Index Scan		0	
-> Index Scan	-> Index Scan	1	1
	-> Index Scan		0
-> Append	-> Append	35,597	32
-> Subquery Scan	-> Subquery Scan	3,551	2
-> Aggregate	-> Aggregate	3,551	2
-> CTE Scan	-> CTE Scan	31,973	56
-> Function Scan	-> Function Scan	30,499	66
-> Subquery Scan	-> Subquery Scan	9,897	0
-> Aggregate	-> Aggregate	9,897	0
-> Sort		26,504	
-> Nested Loop		26,504	
-> CTE Scan		31,973	
-> Index Scan		1	
	-> Result		0
-> Subquery Scan	-> Subquery Scan	0	0
-> Aggregate	-> Aggregate	0	0
-> Result	-> Result	0	0
-> Subquery Scan	-> Subquery Scan	15,976	19
-> Aggregate	-> Aggregate	15,976	19

Different Join Order

CTE fingerprints
-> Nested Loop (cost=1.84..1140.04 rows=1 width=45) (actual time=0.166..428.961 rows=31973 loops=1)
-> Nested Loop (cost=1.27..1137.25 rows=1 width=16) (actual time=0.157..349.766 rows=31973 loops=1)
-> Index Scan using **index_query_table_associations_on_database_id_and_table_name** on public.query_table_associations qta (cost=0.70..327.43 rows=290 width=8) (actual time=0.022..64.070 rows=128992 loops=1)
-> Index Scan using **index_query_occurrences_on_query_id** on public.query_occurrences o (cost=0.57..2.79 rows=1 width=8) (actual time=0.002..0.002 rows=0 loops=128992)
-> Index Scan using **query_fingerprints_query_id_idx** on public.query_fingerprints qf (cost=0.57..2.77 rows=1 width=45) (actual time=0.002..0.002 rows=1 loops=31973)

CTE fingerprints
-> Nested Loop (cost=1.84..8.14 rows=1 width=45) (actual time=0.058..2.619 rows=56 loops=1)
-> Nested Loop (cost=1.27..7.52 rows=1 width=53) (actual time=0.032..1.473 rows=244 loops=1)
-> Index Scan using **index_query_table_associations_on_database_id_and_table_name** on public.query_table_associations qta (cost=0.70..4.72 rows=1 width=8) (actual time=0.021..0.288 rows=244 loops=1)
-> Index Scan using **query_fingerprints_query_id_idx** on public.query_fingerprints qf (cost=0.57..2.79 rows=1 width=45) (actual time=0.004..0.004 rows=1 loops=244)
-> Index Scan using **index_query_occurrences_on_query_id** on public.query_occurrences o (cost=0.57..0.61 rows=1 width=8) (actual time=0.004..0.004 rows=0 loops=244**)**

Use query variants to test hypothesis

Name (Optional)

Try different join order

Baseline Query

```
/*controller:graphql,action:graphql,line:/app/services/dataload/queries/query_stats_for_table.rb:181:in `query_stats_for_tabl
e',sentry_trace_id:11c0590f5359469fbc1dd94a99fbe18d,traceparent:00-c196797a...
```

Show full query text

Variant Query

```
/*+ Leading((query_table_associations query_occurrences) query_fingerprints) */

WITH total_times AS (
  SELECT SUM(query_stats_blk_read_time_sum + query_stats_blk_write_time_sum) AS total_iotime,
         SUM(query_stats_total_time_sum) AS total_runtime
    FROM query_overview_stats_35d qos
   WHERE qos.database_id = $database_id AND qos.collected_at BETWEEN $collected_at_3 AND $collected_at_4
),
fingerprints AS (
  SELECT qf.*
```

Cancel

Check Query

pganalyze

Use query variants to test hypothesis

Query Plans

Filter by Parameter Set...

VARIANT	PLAN	PARAMETER SET	EST. COST	RUNTIME
Baseline	a339f88	Param Set 1	20,436	35.57ms
Baseline	a359a9c	Param Set 2	21,918	⚠ 1,126.47ms
Baseline	a3909ef	Param Set 3	21,892	⚠ 3,512.68ms
Re-run with warm cache	a339f88	Param Set 1	20,436	28.63ms
Re-run with warm cache	a359a9c	Param Set 2	21,918	⚡ 5.29ms
Re-run with warm cache	a3909ef	Param Set 3	21,892	36.75ms
Always use min_occurred_at ...	a339f88	Param Set 1	20,436	29.80ms
Always use min_occurred_at ...	a3da688	Param Set 2	40,355	376.17ms
Always use min_occurred_at ...	a3d2c88	Param Set 3	40,413	122.17ms

pganalyze

6 ways to guide the planner:

1. For simple scan selectivity, look into CREATE STATISTICS
2. For join selectivity, try increasing statistics target
3. Review cost settings (e.g. random_page_cost)
4. Create multi-column indexes that align with the planner's biases (e.g. for bounded sorts)
5. For complex queries with surprising join order, try forcing materialization (WITH x AS MATERIALIZED…)
6. For multi-tenant apps, consider adding more explicit clauses like "WHERE customer_id = 123"

pganalyze

Coming Soon:

Query Tuning Advisor

Common “Pathological” Bad Plans:

1. Inefficient Nested Loops (nested loops when rows are not 1)
2. GROUP BY causing wrong index usage due to sort order
3. ORDER BY + LIMIT causing wrong index usage due to “quickstart” scan
4. Functional dependency causing wrong index usage
5. JSONB expression causing wrong index usage
6. Lossy Bitmap Heap Scan leading to increased table access
7. Early sort because of large result set mis-estimate
8. Memoize on Postgres 14+ causing inefficient plans

Join Our Early Access Program for Query Tuning Workbooks!

- Open to current pganalyze cloud customers
- Focused on **1:1 sessions** to tune a particular slow query together
- You'll get access to the Query Tuning Workbooks feature ahead of the official launch
- We'll use your feedback in the process of developing Query Tuning Advisor
- If you're interested, let us know **in the post-webinar survey or by email**

thanks!

Get a free trial of pganalyze

PGANALYZE.COM

Get free pganalyze eBooks and Postgres blog posts

PGANALYZE.COM/RESOURCES **PGANALYZE.COM/BLOG** **PGANALYZE.COM/NEWSLETTER**