

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



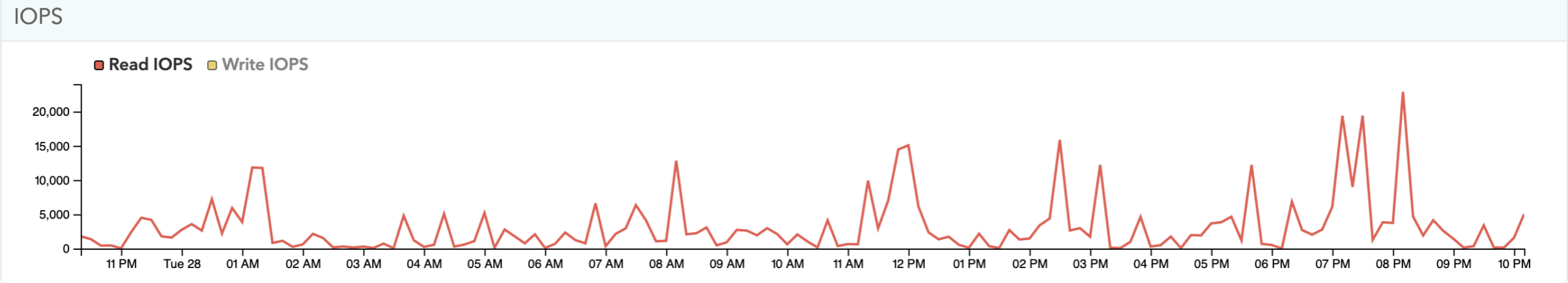
@LukasFittl
hachyderm.io/@lukas

- 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**



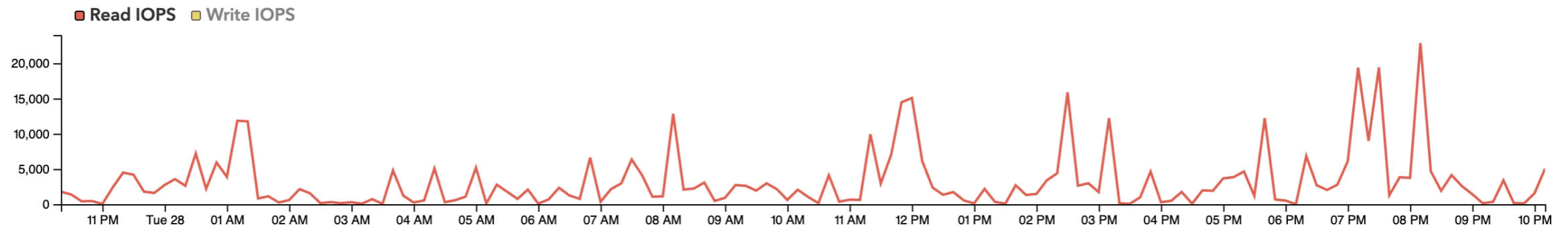


Identifying slow queries & their plans at scale



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

IOPS



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%

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



auto_explain + pganalyze

Nested Loop 3

CTE existing_fingerprints
expensive mis-estimate

I/O Time: 1,033ms
Est. Cost: 19
Actual Rows: 3,624 · est. 1

CTE Scan 4

input
mis-estimate

I/O Time: 0.00ms
Est. Cost: 0
Actual Rows: 4,442 · est. 10

Index Only Scan (Forward) 5

on public.query_fingerprints AS f
using query_fingerprints_fingerprint_data...
i/o-heavy

Executed 4442 times:

Metric	Total	Average
I/O Time:	1,033ms	0.233ms
Est. Cost:	-	2
Actual Rows:	4,442	1 · est. 1

Index Only Scan (Forward) 5

on public.query_fingerprints AS f
using query_fingerprints_fingerprint_database_id_postgres_role_id_idx

Overview | I/O & Buffers | Output | Source

EXPLAIN Insights
i/o-heavy took 52% of total I/O time

Index Only Scan
Scans through the index to fetch a single value or a range of values in index order without reading table data. [Learn more](#)

Index Cond
((f.fingerprint = input_1.fingerprint) AND (f.database_id = input_1.databas...)

Rows Removed by Index Recheck
0

Scan Direction
Forward

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



```
-> 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)

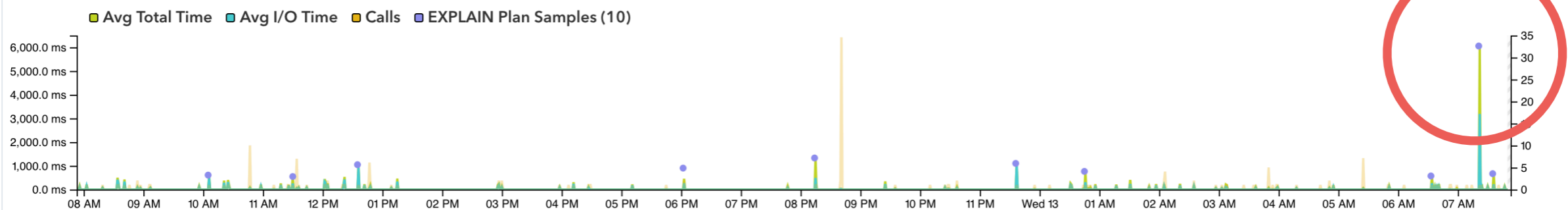
SQL Statement

```
/*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



Is the query always slow, or just sometimes?

```
WITH indexes AS (...), index_sizes AS (...) SELECT ... FROM unpack_schema_table_stats(database_id...
```

Avg Time
1,449.80ms Calls Per Minute
9.32 / min
 Compare to 7 days ago

fingerprint ab2ba35b3f9acddf role pgaweb_workers line /app/services/dataload/schema/stats_series_for_tab... job Issues::CheckUpSingleWorker
sentry_trace_id 2ec4aeebee694bbd8696d47dcb806944 and 118 more View all query tags

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

Check-Up

1 new issue

Info Query #43899555 takes 1287 ms on average (88397 ms max, 3.35 MB read from disk per call, 13390 calls in last 24h)

EXPLAINs

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME	READ FROM DISK	PLAN NODES
2024-10-01 08:14:23pm PDT	⚡ a332ead	348	14,688.59ms	12,796.35ms	87%	42.6 MB Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:03:23pm PDT	⚡ a332ead	348	12,812.28ms	10,883.24ms	85%	51.9 MB Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:13:14pm PDT	⚡ a332ead	348	11,881.92ms	7,873.20ms	66%	476 MB Sort · Nested Loop · CTE Scan +4 more
2024-10-01 07:52:43pm PDT	⚡ a332ead	348	9,564.42ms	7,342.84ms	77%	57.7 MB Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:02:40pm PDT	⚡ a332ead	348	9,120.33ms	7,772.78ms	85%	45.8 MB Sort · Nested Loop · CTE Scan +4 more

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



Is the plan the same, or does it change?

```
WITH indexes AS (...), index_sizes AS (...) SELECT ... FROM unpack_schema_table_stats(database_id...
```

Avg Time 1,449.80ms Calls Per Minute 9.32 / min

fingerprint ab2ba35b3f9acddf role pgaweb_workers line /app/services/dataload/schema/stats_series_for_tab... job Issues::CheckUpSingleWorker

Compare to 7 days ago

sentry_trace_id 2ec4aeebee694bbd8696d47dcb806944 and 118 more View all query tags






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

Check-Up

1 new issue

Info Query #43899555 takes 1287 ms on average (88397 ms max, 3.35 MB read from disk per call, 13390 calls in last 24h)

EXPLAINs

EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME		READ FROM DISK	PLAN NODES
2024-10-01 08:14:23pm PDT	 a332ead	348	14,688.59ms	12,796.35ms	87%	42.6 MB	Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:03:23pm PDT	 a332ead	348	12,812.28ms	10,883.24ms	85%	51.9 MB	Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:13:14pm PDT	 a332ead	348	11,881.92ms	7,873.20ms	66%	476 MB	Sort · Nested Loop · CTE Scan +4 more
2024-10-01 07:52:43pm PDT	 a332ead	348	9,564.42ms	7,342.84ms	77%	57.7 MB	Sort · Nested Loop · CTE Scan +4 more
2024-10-01 08:02:40pm PDT	 a332ead	348	9,120.33ms	7,772.78ms	85%	45.8 MB	Sort · Nested Loop · CTE Scan +4 more

Plan Fingerprints show changes in plan structure



Is the plan the same, or does it change?

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)

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)

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





[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 (...)

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)

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



New for Amazon Aurora: aurora_stat_plans

Aurora PostgreSQL versions 14.10, 15.5, and later versions.

[Contact Us](#) [English](#) [Return to the Console](#)

[AWS](#) > [Documentation](#) > [Amazon RDS](#) > [User Guide for Aurora](#) [Feedback](#) [Preferences](#)

- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans**
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

Aurora PostgreSQL parameters

Aurora PostgreSQL wait events

- ▶ Aurora PostgreSQL updates
- ▶ Using Aurora PostgreSQL Limitless Database
- ▶ Using Aurora Global Database
- ▶ Using RDS Proxy
- ▶ Working with zero-ETL integrations

aurora_stat_plans

[PDF](#) | [RSS](#)

Returns a row for every tracked execution plan.

Syntax

```
aurora_stat_plans(  
    showtext  
)
```

Arguments

- `showtext` – Show the query and plan text. Valid values are NULL, true or false. True will show the query and plan text.




Return type

Returns a row for each tracked plan that contains all the columns from `aurora_stat_statements` and the following additional columns.

- `planid` – plan identifier
- `explain_plan` – explain plan text
- `plan_type`:
 - `no plan` - no plan was captured
 - `estimate` - plan captured with estimated costs
 - `actual` - plan captured with EXPLAIN ANALYZE

On this page

- [Syntax](#)
- [Arguments](#)
- [Return type](#)
- [Usage notes](#)
- [Examples](#)



New for Amazon Aurora: aurora_stat_plans

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

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

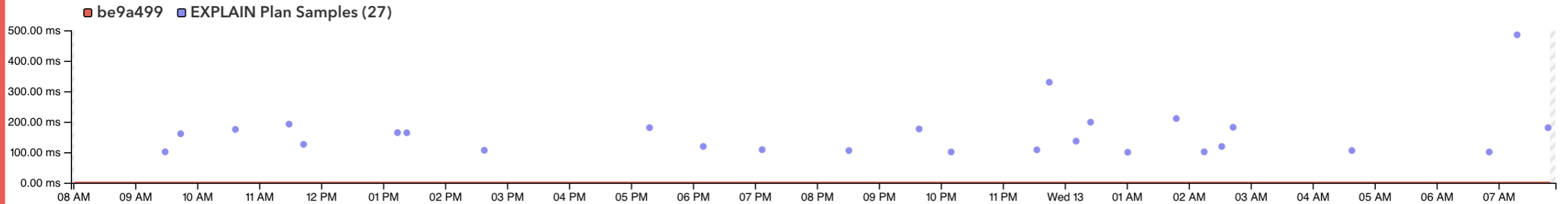
fingerprint 3315bfa60c2c07a3 role benchmark No query tags collected

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

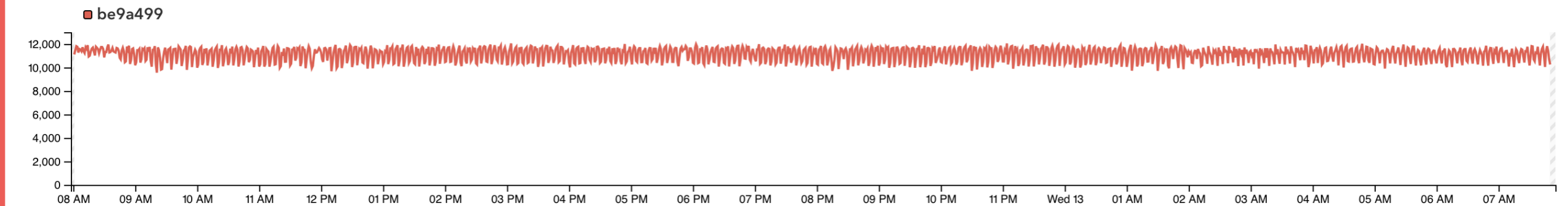
Plan Statistics

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)

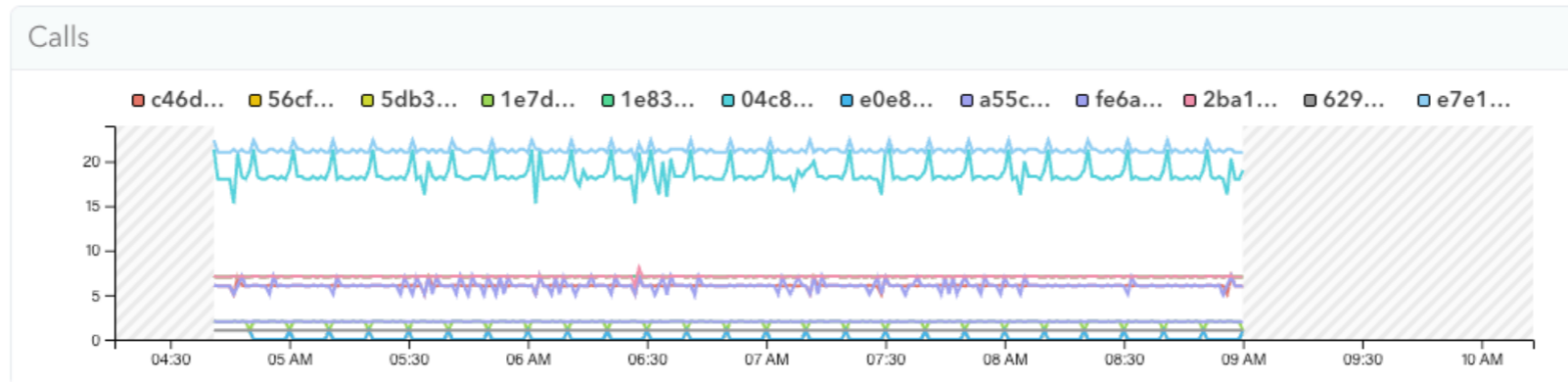
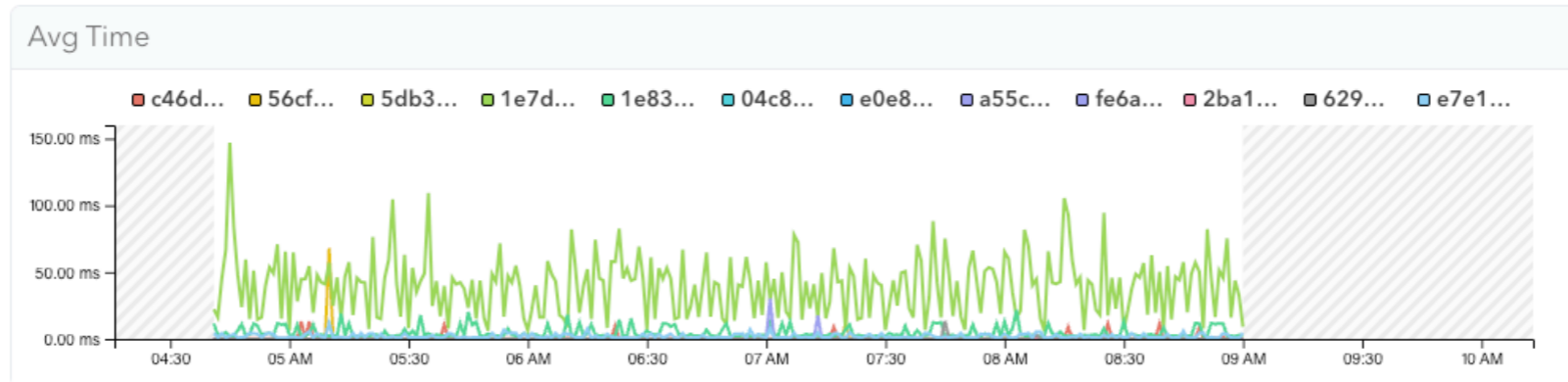
EXECUTED AT	PLAN	EST. COST	RUNTIME	I/O READ TIME	READ FROM DISK	PLAN NODES
2024-11-13 07:48:00am PST	be9a499	8	180.62ms	0.00ms 0%	0 B	ModifyTable · Index Scan

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 < >





[New!] Plan Comparison in pganalyze

Comparing Automated EXPLAIN plans

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)

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)

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 07:22:59am PST	e14ba6e	12,561	573.58ms	251.13ms	44%	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

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



Comparing Automated EXPLAIN plans

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

Avg Time 243.18ms Calls Per Minute 0.14 / min

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

Compare to 7 days ago

Overview Index Advisor ? Query Samples 5+ EXPLAIN Plans 5+ Query Tags 5+ Log Entries 60

Node Tree Text JSON Compare Plans

SQL Statement

```
/*controller:graphql,action:graphql,graphql:getDatabaseDetails.indexWriteOverhead,line:/app/services/dataload/data  
base.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	n/a	n/a	n/a	n/a
Plan B	n/a	n/a	n/a	n/a

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

Select Plans

Summary Node Details Node Source

Select plans to compare

Get Help

Comparing Automated EXPLAIN plans

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

Select plans to compare ✕

	FINGERPRINT	LABEL	RUNTIME	I/O READ TIME
<input type="checkbox"/>	*e14ba6e	2024-11-13 12:45:36am PST	761.65ms	303.53ms
<input type="checkbox"/>	*e14ba6e	2024-11-13 09:39:18am PST	14,312.55ms	13,070.11ms
<input type="checkbox"/>	*e14ba6e	2024-11-13 09:30:05am PST	731.93ms	460.24ms
<input type="checkbox"/>	*e14ba6e	2024-11-13 08:36:59am PST	533.57ms	222.08ms
<input checked="" type="checkbox"/>	*e14ba6e	2024-11-13 07:35:56am PST	668.78ms	211.71ms
<input checked="" type="checkbox"/>	*3679251	2024-11-13 07:21:12am PST	6,054.90ms	3,188.57ms
<input type="checkbox"/>	*e14ba6e	2024-11-13 06:33:59am PST	573.58ms	264.12ms
<input type="checkbox"/>	*e14ba6e	2024-11-12 12:35:30pm PST	1,044.96ms	950.55ms
<input type="checkbox"/>	*e14ba6e	2024-11-12 12:21:26pm PST	529.57ms	422.03ms
<input type="checkbox"/>	*e14ba6e	2024-11-12 11:36:24pm PST	1,099.31ms	1,092.13ms

1-10 of 50 < >

	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

[Compare Plans](#)



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

Summary

	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



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

	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

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:10
2: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



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	-> Index Scan ³	5	2,461
-> Bitmap Index Scan ³	-> Index Scan ³	5	2,461
-> Aggregate	-> Aggregate	5	2,461
-> Hash Join	-> Hash Join	8	5,009
-> Nested Loop	-> Nested Loop	8	5,009
-> CTE Scan	-> CTE Scan	5	2,461
-> Index Scan ²	-> Nested Loop	2	2
-> Hash	-> Index Scan ²	1	2
-> Seq Scan	-> Index Scan ¹	5,050,837	1
-> Nested Loop	-> Nested Loop	5,050,837	2,461
-> 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 B

Index Scan (Forward)

on public.schema_index_metadata AS sim
using index_schema_index_metadata_on_index_id

Scans through the index to fetch a single value or a range of values in index order from the table. [Learn more](#)

Index Cond
(sim.index_id = si.id)

Rows Removed by Index Recheck
0

Scan Direction
Forward

I/O & Buffers

	Shared	Local	Temp
Hit	181.6 MB	0 B	-
Read	328 kB	0 B	0 B
Dirtied	0 B	0 B	-
Written	0 B	0 B	0 B


I/O Read Time: 16.52ms I/O Write Time: 0.00ms

Output Columns

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



CTEs and InitPlans are split out in the diff



Server
prod-db-main Primary

Database
pgaweb

Last 24 hours

Avg Time
99.21ms
Calls Per Minute
0.51 / min

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

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
Plan A		Plan B		
-> 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 ³		-> Index Only Scan ²	0.11ms	0.26ms
-> Index Only Scan ⁴		-> Index Only Scan ³	0.19ms	0.18ms
-> Materialize		-> Index Only Scan ⁴	0.17ms	0.39ms
-> Sort		-> Materialize	0.32ms	0.69ms
-> CTE Scan		-> Sort	617.66ms	1,069.88ms
-> CTE Scan		-> CTE Scan		
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		-> Sort	0.82ms	1.90ms
-> Function Scan		-> Nested Loop	613.93ms	1,061.32ms
-> CTE Scan		-> Function Scan	0.00ms	0.00ms
InitPlan 2 (returns \$1)		-> CTE Scan		
-> CTE Scan		-> CTE Scan	0.01ms	0.01ms

Summary

	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

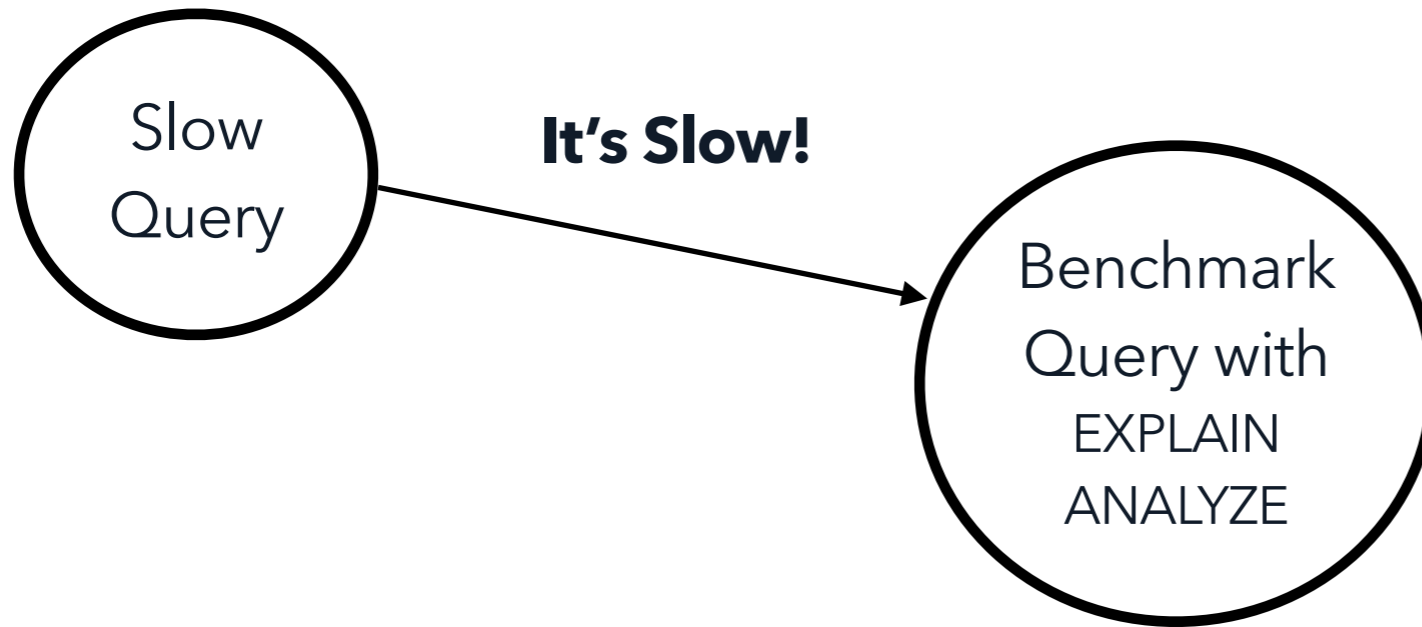
A	B	Index
✓	✓	1. query_fingerprints_query_id_idx
✓	✓	2. query_overview_stats_35d_20241110_pkey
✓	✓	3. query_overview_stats_35d_20241111_pkey
✓	✓	4. query_overview_stats_35d_20241112_pkey

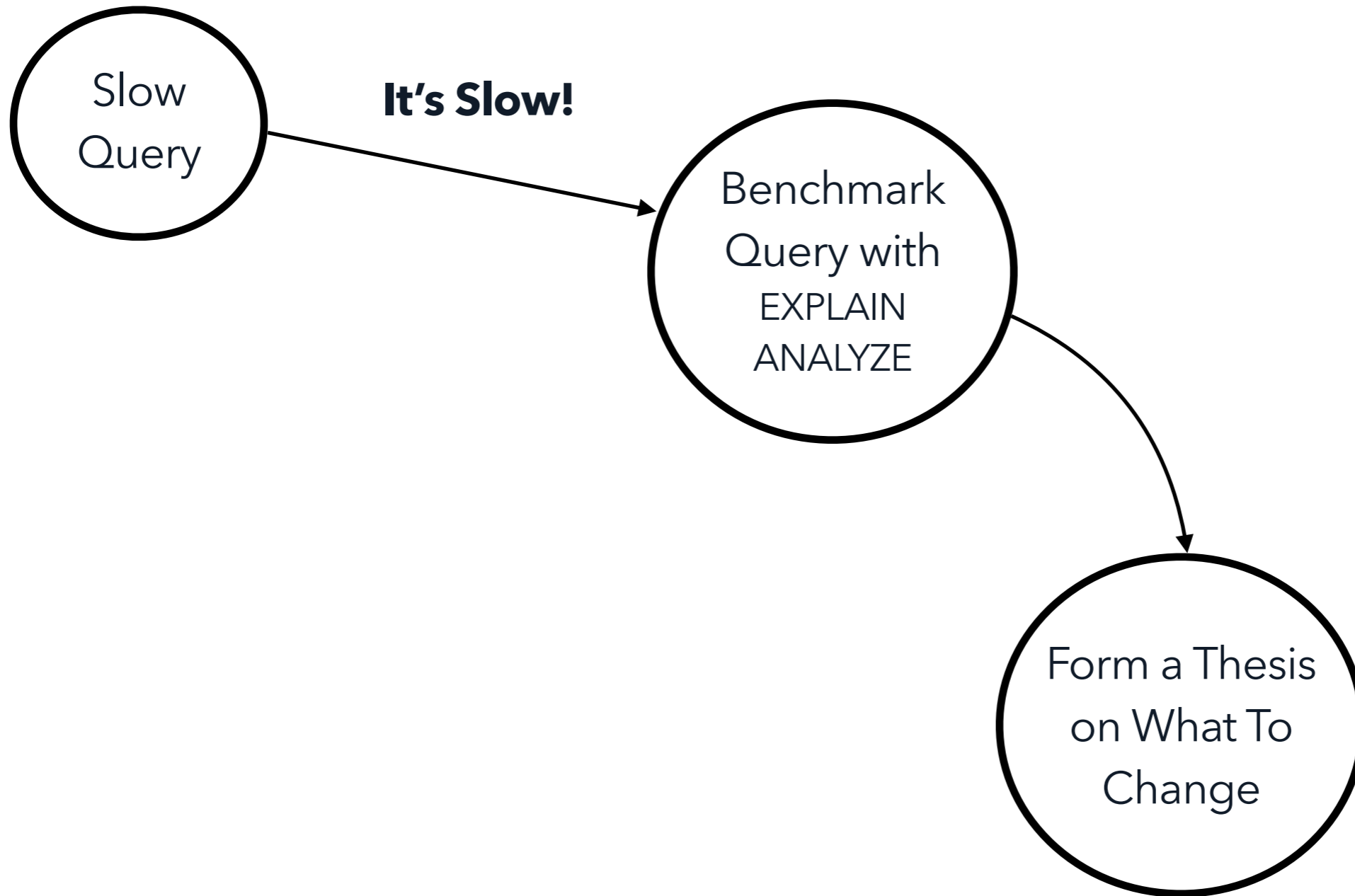
Maciek Sak... Get Help

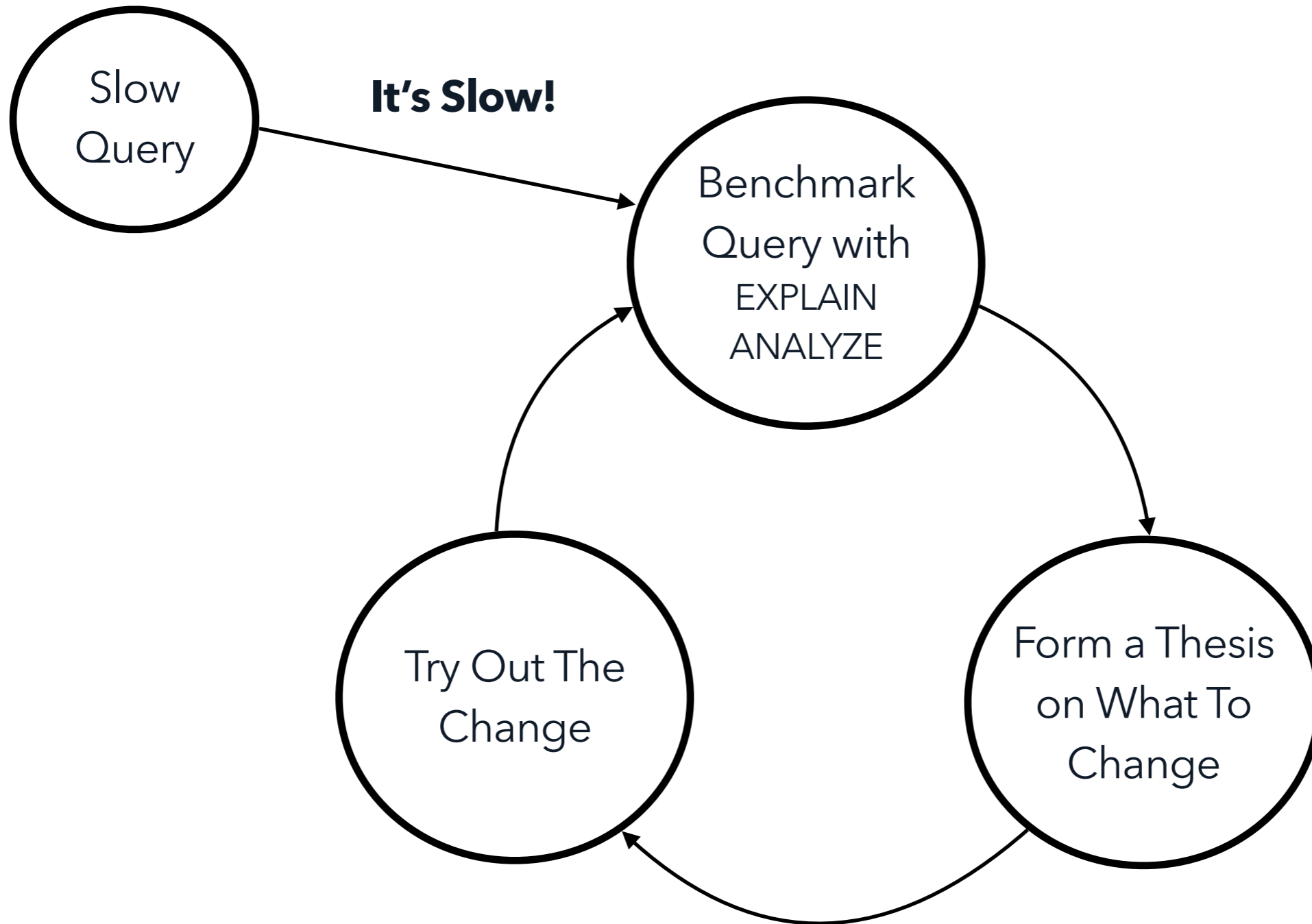


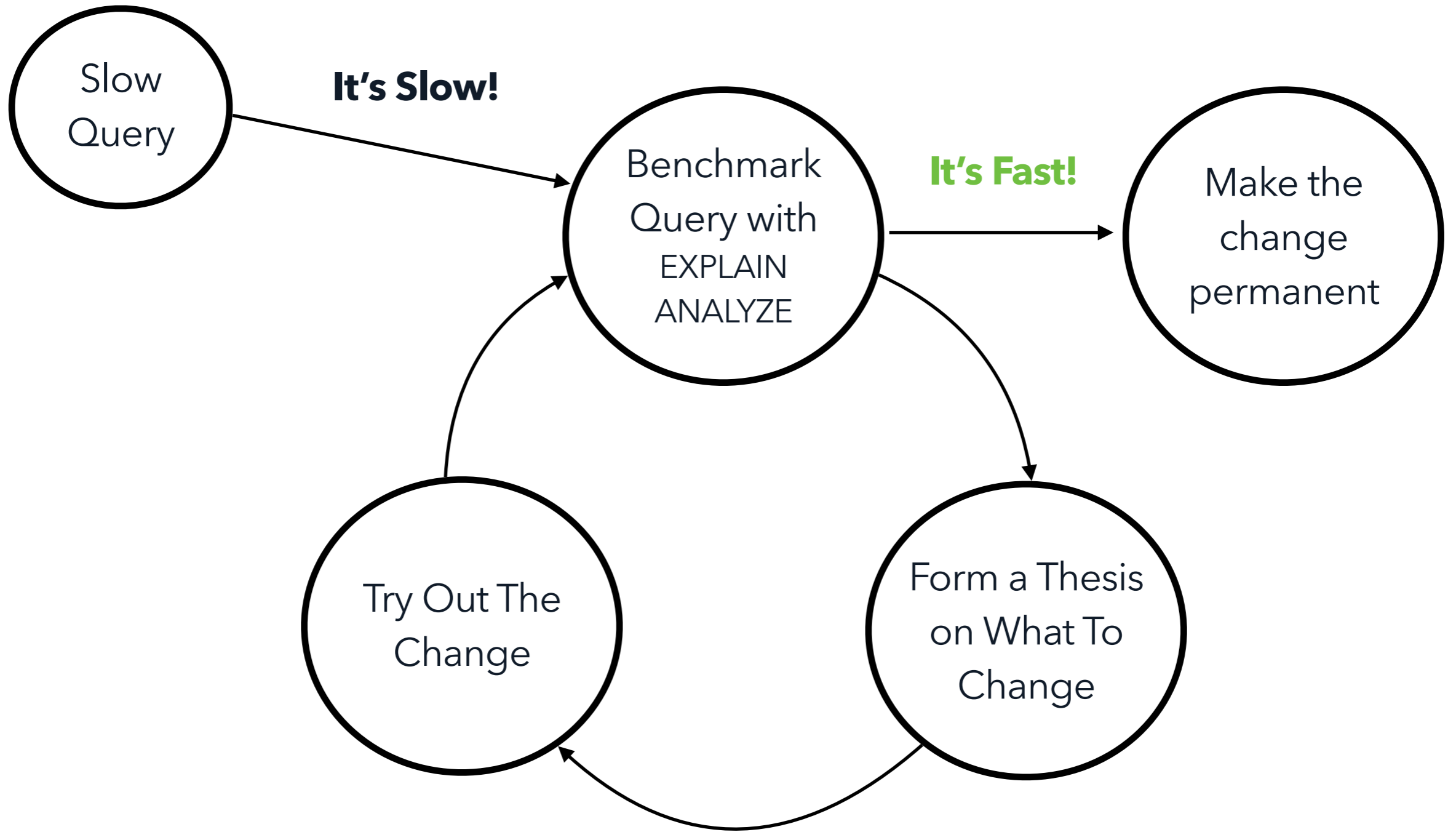


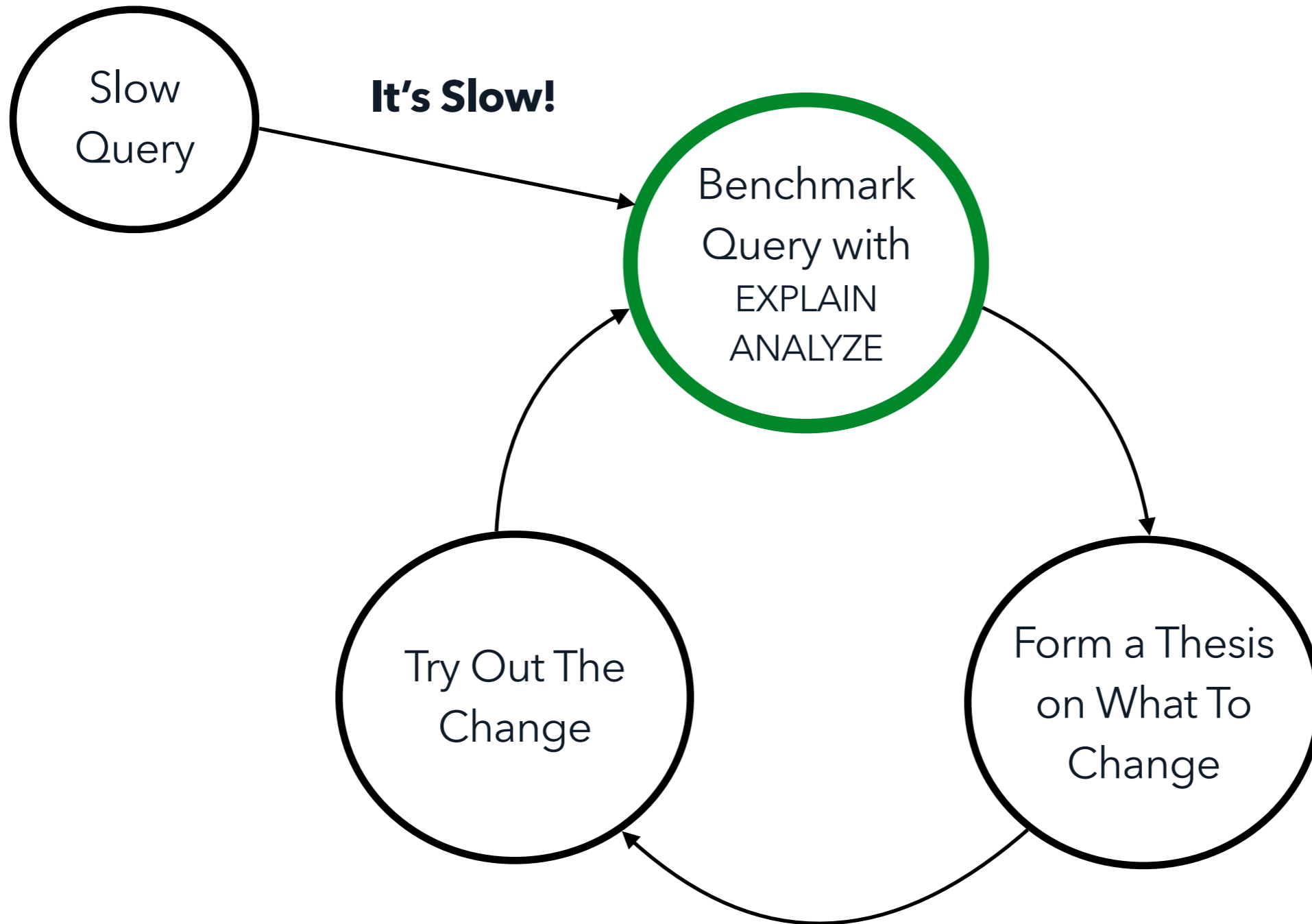
How to debug why a query is slow











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
```

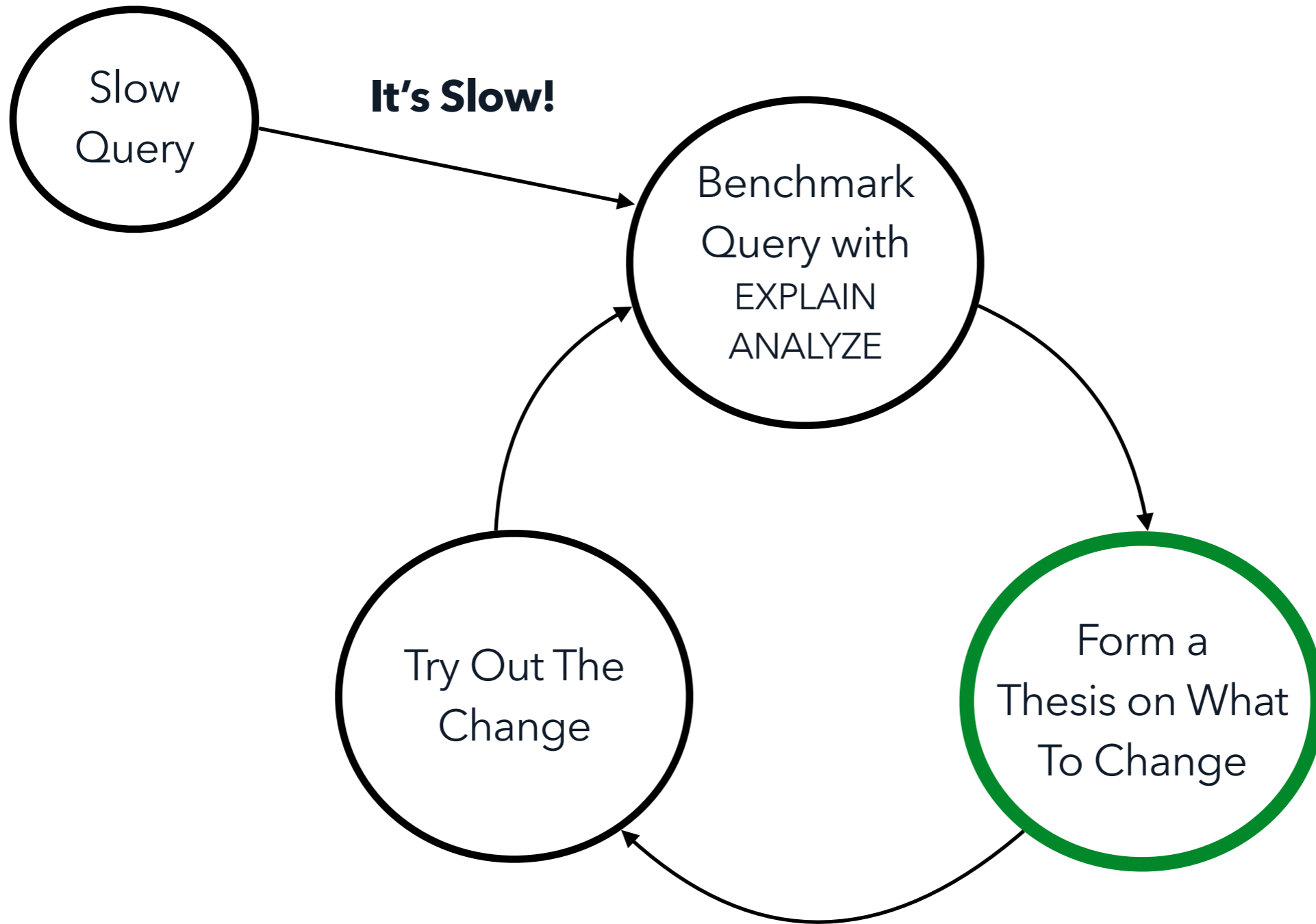
```
(12 rows)
```



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)





“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)



↻ **Nested Loop** inefficient nested loop 5

Actual Time: 3,375ms
I/O Time: 2,354ms
Est. Cost: 182
Actual Rows: 1,007 · est. 1

f(x) **Function Scan** expensive i/o-heavy mis-estimate 6

Actual Time: 1,592ms
I/O Time: 1,209ms
Est. Cost: 175
Actual Rows: 1,007 · est. 1

☰ **CTE Scan** 7

index_sizes

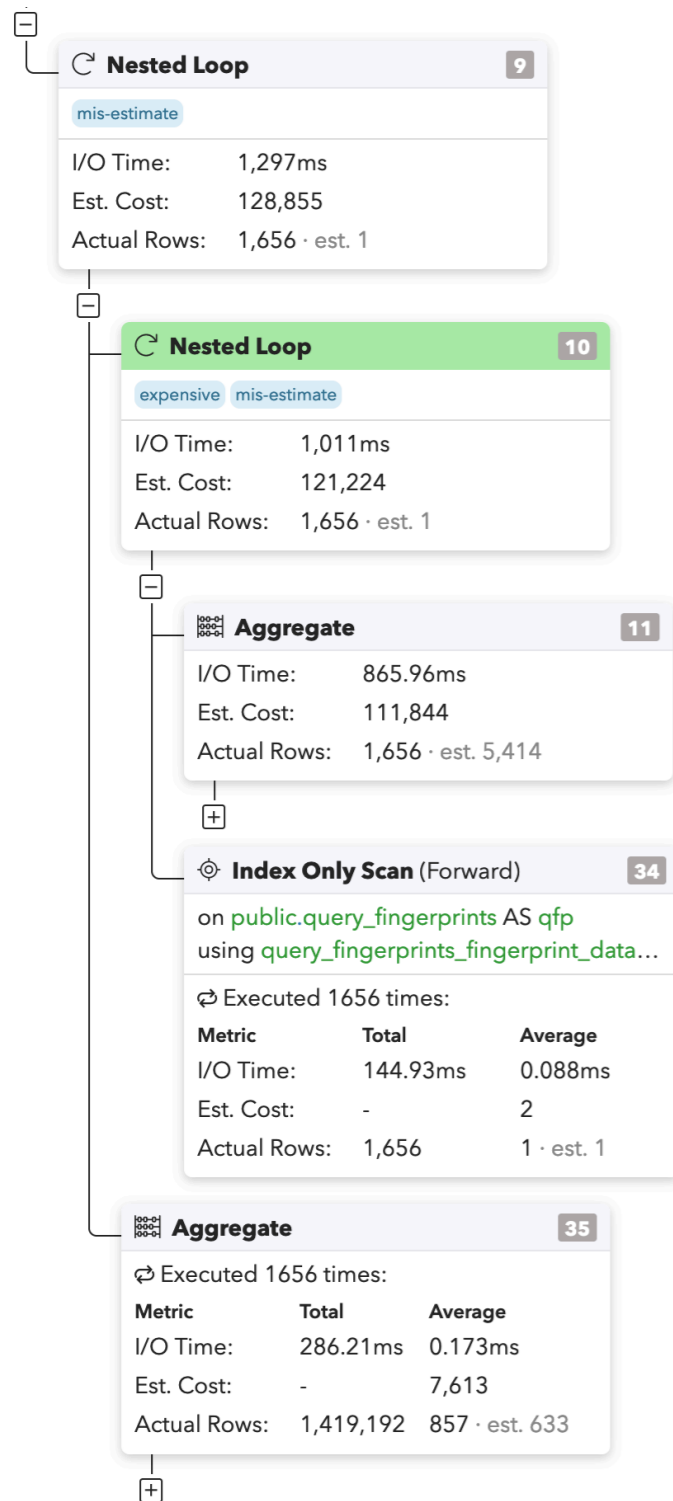
↻ Executed 1007 times:

Metric	Total	Average
Actual Time:	1,705ms	1.69ms
I/O Time:	1,145ms	1.14ms
Est. Cost:	-	4
Actual Rows:	1,014,049	1,007 · est. 200

New pganalyze EXPLAIN Insight: Inefficient Nested Loop

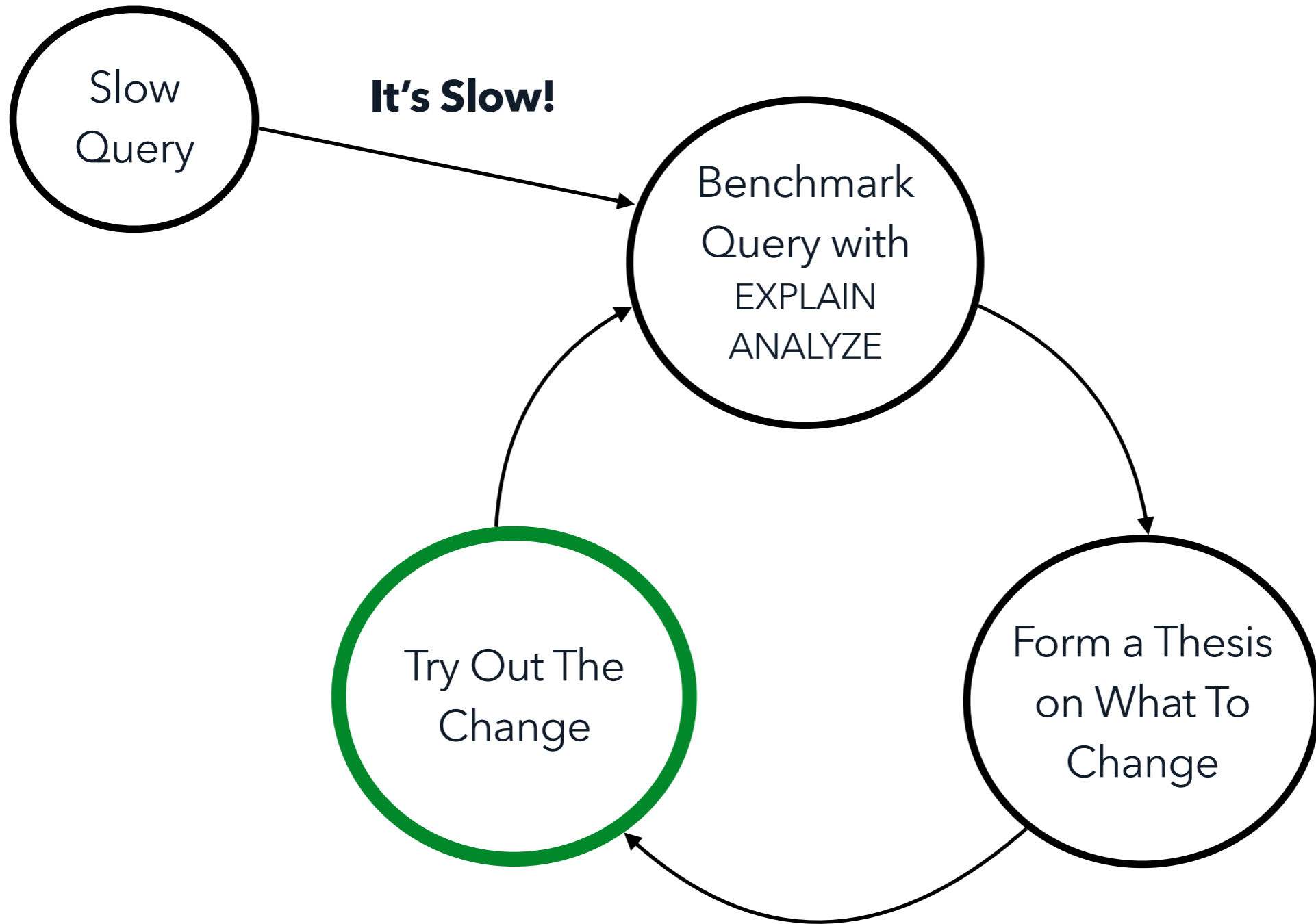
-> Nested Loop (cost=0.25..181.76 **rows=1** width=152)
(**actual rows=1007**)





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.



To Understand
Why A "Bad" Plan Was Chosen
Start By Forcing The Good Plan





Benchmarking alternate plans with settings & hints

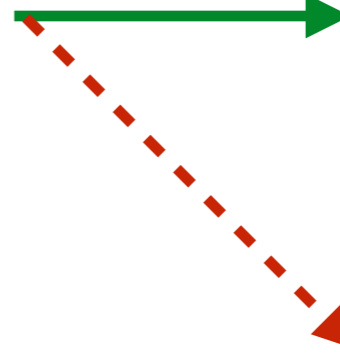
```
SELECT * FROM test  
WHERE object_id = 123
```



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



Cost=250



Cost=300

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



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



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



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



Cost=500



Cost=300

The easiest test:

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





Cost=300



Cost=500



SET enable_seqscan = off



Cost=100000000000.00



Cost=500

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 width=8)
    -> Index Scan using schema_tables_database_id_seq on schema_tables_database_id_seq
        (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 width=8)
    -> Index Only Scan using index_schema_column_statistics on index_schema_column_statistics
        (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 width=8)
    -> Index Only Scan using index_schema_column_statistics on index_schema_column_statistics
        (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

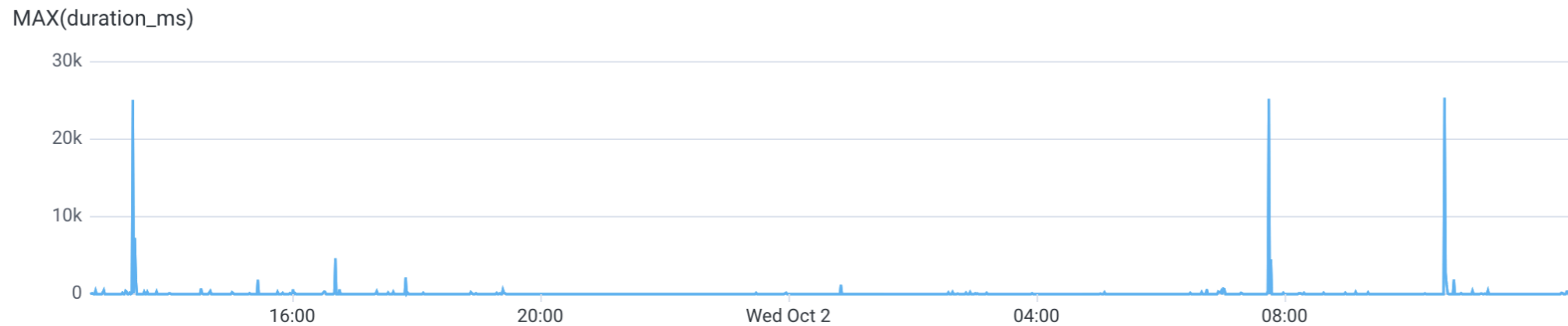
You're receiving notifications because you're watching this repository.





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

← Query in pganalyze-app

Reload Trace

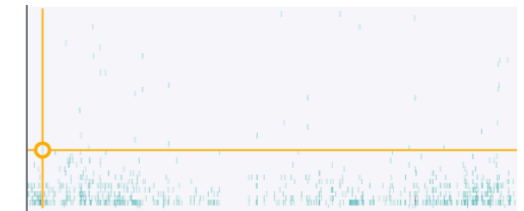
Trace ef312a415e8f9a77623e53cc18a8705e

Trace summary 34 spans at Oct 1 2024 13:28:19 UTC-04:00 (7.308s)

name	Service Name	0s	1s	2s	3s	4s	5s	6s	7.308s
1 SchemaTable.find_by_sql	pganalyze-app	2.281ms							
pgaweb	pganalyze-app	0.9960ms							
1 PostgresSetting.find_by_s...	pganalyze-app	2.517ms							
pgaweb	pganalyze-app	1.084ms							
1 PostgresRole.find_by_sql	pganalyze-app	2.363ms							
pgaweb	pganalyze-app	0.9996ms							
1 PostgresSetting.find_by_s...	pganalyze-app	2.787ms							
pgaweb	pganalyze-app	0.9922ms							
1 SchemaAggregateInfo.fin...	pganalyze-app	2.035ms							
pgaweb	pganalyze-app	0.9203ms							
3 Dataload.select_rows	pganalyze-app	7.134s							
EXPLAIN Plan	Postgres (pganalyze)	7.120s							
EXPLAIN Plan	Postgres (pganalyze)	7.120s							
pgaweb	pganalyze-app	7.133s							
Database.find_by_sql	pganalyze-app							0.8299ms	
Server.find_by_sql	pganalyze-app							0.4473ms	
pgaweb	pganalyze-app							6.024ms	
SELECT pgaweb	pganalyze-app							0.9575ms	
2 HTTP POST	pganalyze-app							46.53ms	
connect	pganalyze-app							12.46ms	
HTTP POST	pganalyze-app							33.51ms	

Postgres (pganalyze) > EXPLAIN Plan

Distribution of span duration



Fields Span events (0) Links (0)

Filter fields and values in span

Timestamp	...
2024-10-01T17:28:19.9492574Z	
db.postgresql.plan	...
https://app.pganalyze.com/servers/edzxhla46zfcjlvpyyl36oivq/databases/pgaweb/queries/b33bede238bc4de1/samples/1727803707?role=pgaweb_app	
db.system	...
postgresql	
duration_ms	...
7120	
library.name	...
go.opentelemetry.io/otel/sdk/tracer	
library.version	...
0.58.0	
meta.signal_type	...
trace	
name	...
EXPLAIN Plan	

Multiple Mis-Estimates of Nested Loops

The screenshot displays three execution plan nodes from PostgreSQL:

- Node 6: Nested Loop (inefficient nested loop)**
 - CTE fingerprints
 - Actual Time: 659.24ms
 - I/O Time: 0.00ms
 - Est. Cost: 1,140
 - Actual Rows: 26,241 · est. 1
- Node 7: Nested Loop (expensive, inefficient nested loop)**
 - Actual Time: 571.86ms
 - I/O Time: 0.00ms
 - Est. Cost: 1,137
 - Actual Rows: 26,241 · est. 1
- Node 8: Index Scan (Forward) (expensive, mis-estimate)**
 - on public.query_table_associations AS qta
 - using index_query_table_associations_on_databas...
 - Actual Time: 170.45ms
 - I/O Time: 0.00ms
 - Est. Cost: 327
 - Actual Rows: 129,405 · est. 290

Under Estimate



Under Estimate



Under Estimate

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

WITH total_times AS (...), fingerprints AS (...), raw_query_data AS (...), query_data AS (...), q...

Avg Time 18.46ms Calls Per Minute 1.69 / min

fingerprint b33bede238bc4de1 role pgaweb_app controller graphql action graphql line /app/services/dataload/queries/query_stats_for_tab... View all query tags

Compare to 7 days ago

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

Node Tree Text JSON Compare Plans

Summary Node Details Node Source

CTE Scan mis-estimate 42

raw_query_data

Actual Time: 1,058ms
I/O Time: 19.44ms
Est. Cost: 2
Actual Rows: 35,431 · est. 101

Hash 43

Actual Time: 675.98ms
I/O Time: 0.00ms
Est. Cost: 0
Actual Rows: 26,241 · est. 1

CTE Scan mis-estimate 44

fingerprints

Actual Time: 668.50ms
I/O Time: 0.00ms
Est. Cost: 0
Actual Rows: 26,241 · est. 1

Index Scan (Forward) i/o-heavy 45

on public.queries AS q
using queries_pkey

Executed 19764 times:

Metric	Total	Average
Actual Time:	5,178ms	0.262ms
I/O Time:	2,772ms	0.140ms
Est. Cost:	-	3
Actual Rows:	19,764	1 · est. 1

Index Scan (Forward)

on public.queries AS q
using queries_pkey

Scans through the index to fetch a single value or a range of values in index order from the table. [Learn more](#)

Index Cond
(q.id = qfp.query_id)

Rows Removed by Index Recheck
0

Scan Direction
Forward

Insights (1)

i/o-heavy took 99% of total I/O time

I/O & Buffers




	Shared ⓘ	Local ⓘ	Temp ⓘ
Hit ⓘ	724.9 MB	0 B	-
Read ⓘ	47.9 MB	0 B	0 B
Dirtied ⓘ	6.3 MB	0 B	-
Written ⓘ	0 B	0 B	0 B

I/O Read Time 2,772.08ms
I/O Write Time 0.00ms



Let's Tune The Query!

```
WITH total_times AS (...), fingerprints AS (...), raw_query_data AS (...), q
```

 fingerprint b33bede238bc4de1  role pgaweb_app  line /app/services/dataload/queries/query_stats_for_tab... cc

[Overview](#) [Index Advisor](#)  [Query Samples](#) **5+** [EXPLAIN Plans](#) **5+** [Query Tags](#) **5+** [Log](#)

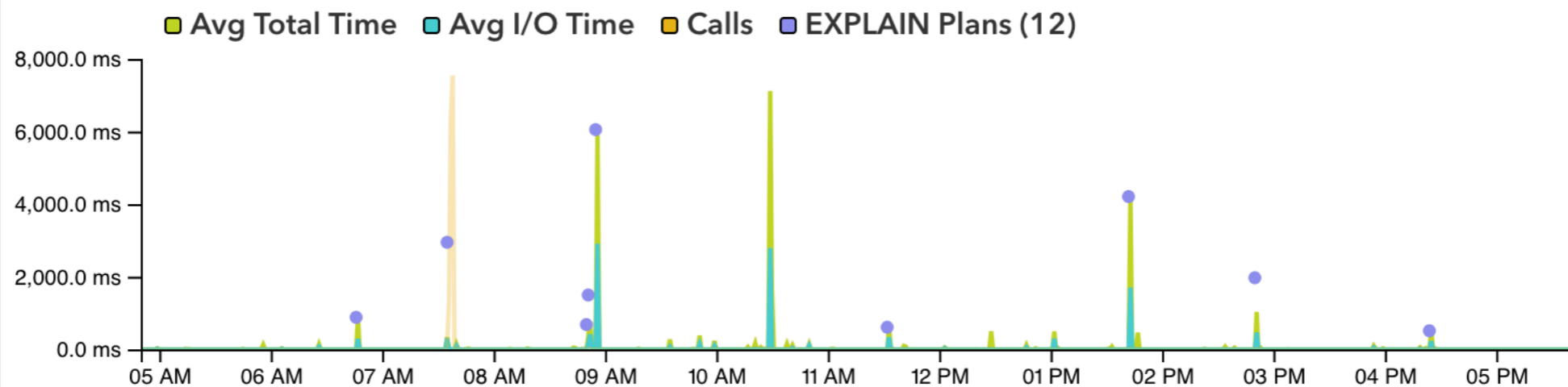
SQL Statement

```
/*controller:graphql,action:graphql,line:/app/services/dataload/queries/query_stats_for_tabl  
d,traceparent:00-c196797a...
```

[Show full query text](#)

 [Tune query in workbook](#)

Avg Time & Calls



Let's Tune The Query!

Server
● prod-db-r

WITH total
fingerprint

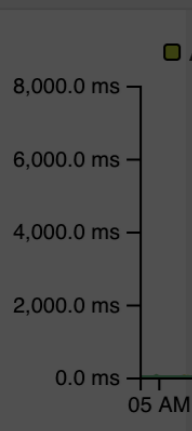
Overview

SQL State

/*control
d,tracepar
Show full qu

Tune quer

Avg Time



New workbook ✕

Create variants of a query and track progress towards improving query time.

Name
Tune Query #43900342

Description (Optional)
Review/Optimize Nested Loops

Cancel Next

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

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

QUERY SAMPLE	PLAN FINGERPRINT	RUNTIME ▾
<input type="checkbox"/> \$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
<input checked="" type="checkbox"/> \$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
<input checked="" type="checkbox"/> \$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...](#)



Benchmark the same query, with different parameters

Run EXPLAIN ANALYZE

[Switch to Collector workflow](#)

EXPLAIN for Param Set 1

Command

```
EXPLAIN (ANALYZE, VERBOSE, BUFFERS, FORMAT JSON)
/*controller:graphql,action:graphql,line:/app/services/dataload/queries/query_stats_for_table.rb: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 = ██████████ AND qos.collected_at BETWEEN '2024-09-30 17:28:19' AND '2024-10-01 17:28:19'
```

[copy](#)

EXPLAIN output

[Text or JSON format supported](#)

Paste EXPLAIN output here...

We've recorded the Baseline

Tune Query #43900342

[Overview](#) [Compare Plans](#) [Parameter Sets](#)

All Query Plans


Baseline

[+ Add Query Variant](#)

Query

#43900342

Query tags

 fingerprint b33bede238bc4de1

 role pgaweb_app





Baseline

With parameter aliases ▼

```
/*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:0  
0-c196797a...
```

[Show full query text](#)

Query Plans

VARIANT	PLAN	PARAMETER SET	EST. COST	RUNTIME
Baseline	 a392842	Param Set 1	3,017	1,738.88ms
Baseline	 a3ed913	Param Set 2	986	382.88ms
Baseline	 a3aa4a4	Param Set 3	1,874	 49.00ms



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	-> Index Scan	0	1
-> Index Scan	-> Index Scan	1	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	-> Result	26,504	0
-> Nested Loop	-> Result	26,504	0
-> CTE Scan	-> Result	31,973	0
-> Index Scan	-> Result	1	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_table',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



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

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"





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](https://pganalyze.com)

Get free pganalyze eBooks and Postgres blog posts

[PGANALYZE.COM/RESOURCES](https://pganalyze.com/resources)

[PGANALYZE.COM/BLOG](https://pganalyze.com/blog)

[PGANALYZE.COM/NEWSLETTER](https://pganalyze.com/newsletter)

