Fixing & Detecting
Slow Postgres Queries with
pganalyze Query Advisor

lukas@pganalyze.com

1. Re-introducing pganalyze Workbooks
2. What's new with Workbooks?
3. Introducing pganalyze Query Advisor (Demo!)
4. Common Problems detected by Query Advisor
5. Behind the Scenes of Query Advisor
6. Early Access & Looking Ahead

pganalyze

Re-introducing pganalyze Workbooks

Introducing Query Tuning Workbooks: Safely Tune Postgres Queries on Production with pganalyze


Keiko Oda
Engineering


Maciek Sakrejda
Engineering


Sean Linsley
Engineering


Jens Nikolaus
Design


Lukas Fittl
Product

Published on:
January 08, 2025

In this article:

At some point, every engineering team finds itself grappling with the complexity of query optimization. One query may run perfectly well for a particular customer's parameters, yet degrade performance for another's dataset. A small tweak that improves latency in staging might have unforeseen consequences in production. Until now, the standard approach to experimentation—perhaps running `EXPLAIN ANALYZE` in a sandbox environment, copying and pasting results into an editor—hasn't given developers or DBAs the full, real-world picture they need.

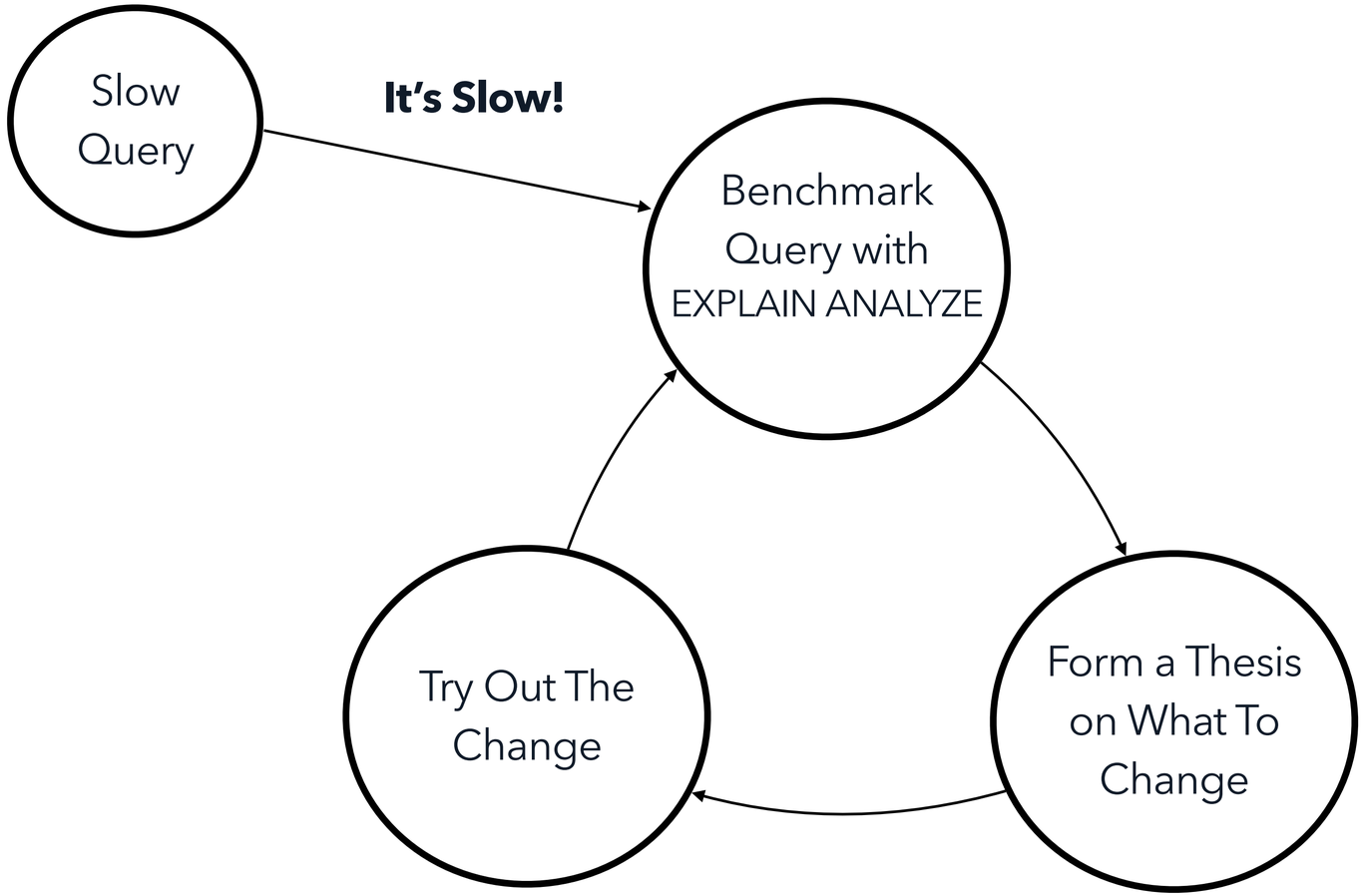
Today, **we're excited to introduce the new Query Tuning Workbooks feature in pganalyze**. Currently in beta, Query Tuning Workbooks let you benchmark a query that is slow, experiment with rewrites, planner settings, or planner hints, and compare the resulting EXPLAIN plans.

You can either upload `EXPLAIN ANALYZE` results to a workbook, or you can opt-in to running `EXPLAIN ANALYZE` through the pganalyze collector, making it easier to test different query variants on production, benchmark the same query with multiple different input parameters, and share the results with your team.

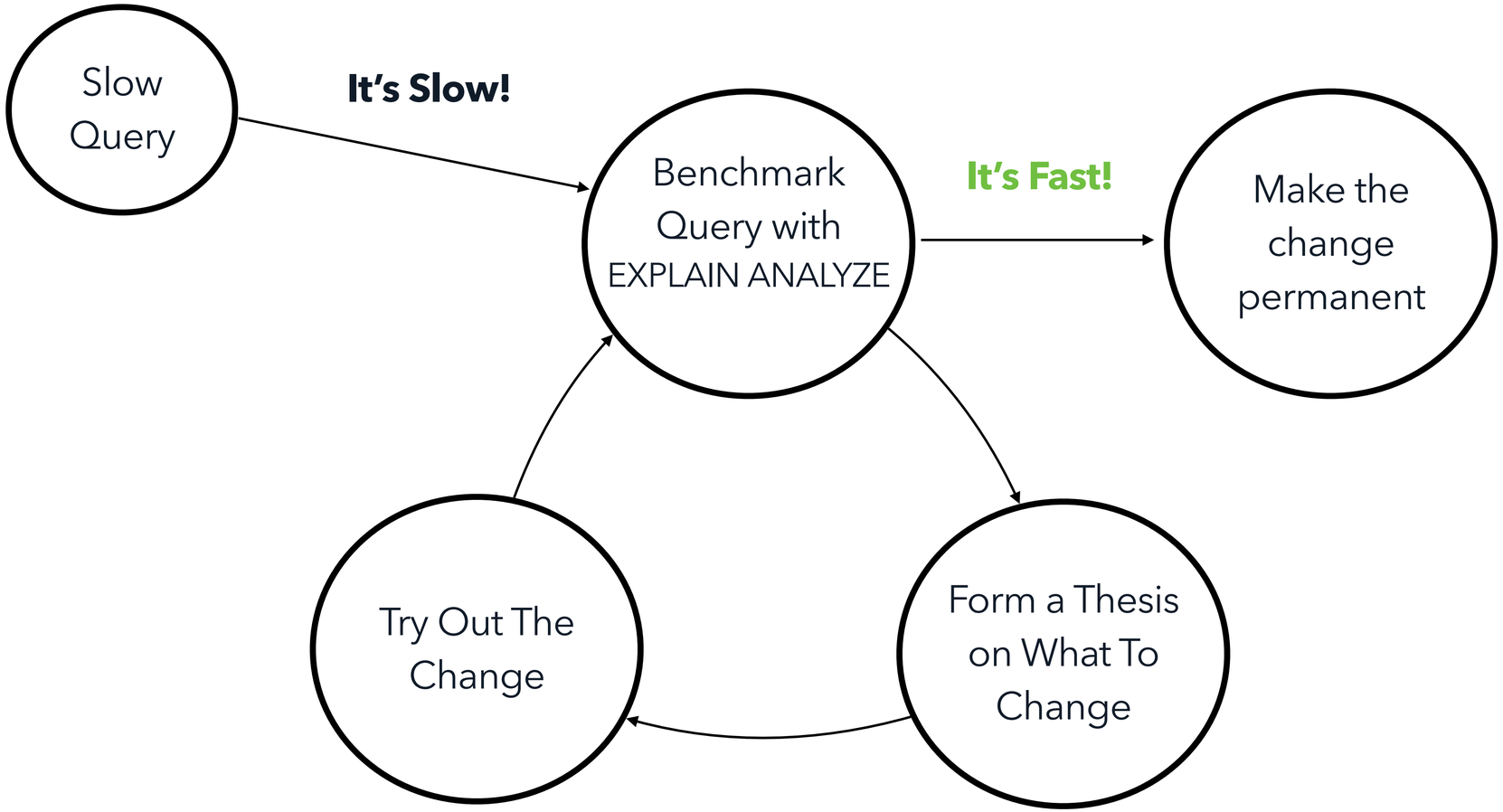
A dedicated environment for tuning Postgres queries

At a high level, the workflow for tuning Postgres queries can be split into three steps:

a. Record details of the execution of a particular query with a given set of input parameters using `EXPLAIN (ANALYZE, BUFFERS)`
b. Form a hypothesis on which parts of the query are slower than necessary, and how to tweak them
c. Try out a change, such as rewriting the query, or turning of a planner setting to see a different plan


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

Speed up role filter in query_stats_for_database #4963

Edit | Code

Merged **seanlinsley** merged 1 commit into `main` from `query-stats-for-databases-role-filter-speedup` on Jun 20

Conversation 1 | Commits 1 | Checks 7 | Files changed 1 | +5 −2

seanlinsley commented on Jun 18 • edited

https://pganalyze.zendesk.com/agent/tickets/6770 reported a query timeout when using the filter on the Query Performance list view. This regression was introduced as part of #4807

Example Ruby code:

```
Dataload::Queries.query_stats_for_database(Database.find(-663636933),
  start_ts: Time.at(1750253727), end_ts: Time.at(1750275327), statement_types: ["read", "write", "other"],
  filter: "test", offset: 0, limit: 100, sort_by: "pctOfTotal", sort_direction: "desc", compare: true
).count
```

The problem is this part of the explain plan, where a nested loop join has to filter out 74 million rows:

```
  ->  Sort  (cost=756.23..756.24 rows=1 width=258) (actual time=49170.982..49171.001 rows=100 loops=1)
        Sort Key: (sum(qd.pct_of_total)) DESC NULLS LAST
        Sort Method: top-N heapsort  Memory: 376kB
        Buffers: shared hit=96303
        ->  Nested Loop  (cost=177.69..756.22 rows=1 width=258) (actual time=1260.248..49170.092 rows=513 loo
              Join Filter: ((qd.fingerprint = query_data_early_sort_limit.fingerprint) AND ((q.normalized_que
              Rows Removed by Join Filter: 74097721
              Buffers: shared hit=96300
              ->  Nested Loop  (cost=108.09..677.00 rows=1 width=550) (actual time=1072.749..1157.073 rows=87
                    Buffers: shared hit=88734
                    ->  HashAggregate  (cost=107.50..112.00 rows=200 width=112) (actual time=1072.693..1083.4
                          Group Key: qd.fingerprint
                          Batches: 1  Memory Usage: 8609kB
                          Buffers: shared hit=45019
                          ->  CTE Scan on query_data_early_sort_limit qd  (cost=0.00..43.00 rows=2150 width=1
                                Buffers: shared hit=45019
                    ->  Index Scan using index_queries_on_database_id_and_fingerprint on queries q  (cost=0.5
                          Index Cond: ((database_id = '-663636933'::integer) AND (fingerprint = (("right"((qd
                          Filter: ((statement_types && '{SelectStmt}'::text[]) OR (statement_types && '{Inser
                          Buffers: shared hit=43713
              ->  GroupAggregate  (cost=69.60..73.72 rows=200 width=64) (actual time=0.002..4.844 rows=8477 l
                    Group Key: query_data_early_sort_limit.fingerprint
                    Buffers: shared hit=6
```

Reviewers
lfittl ✓

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

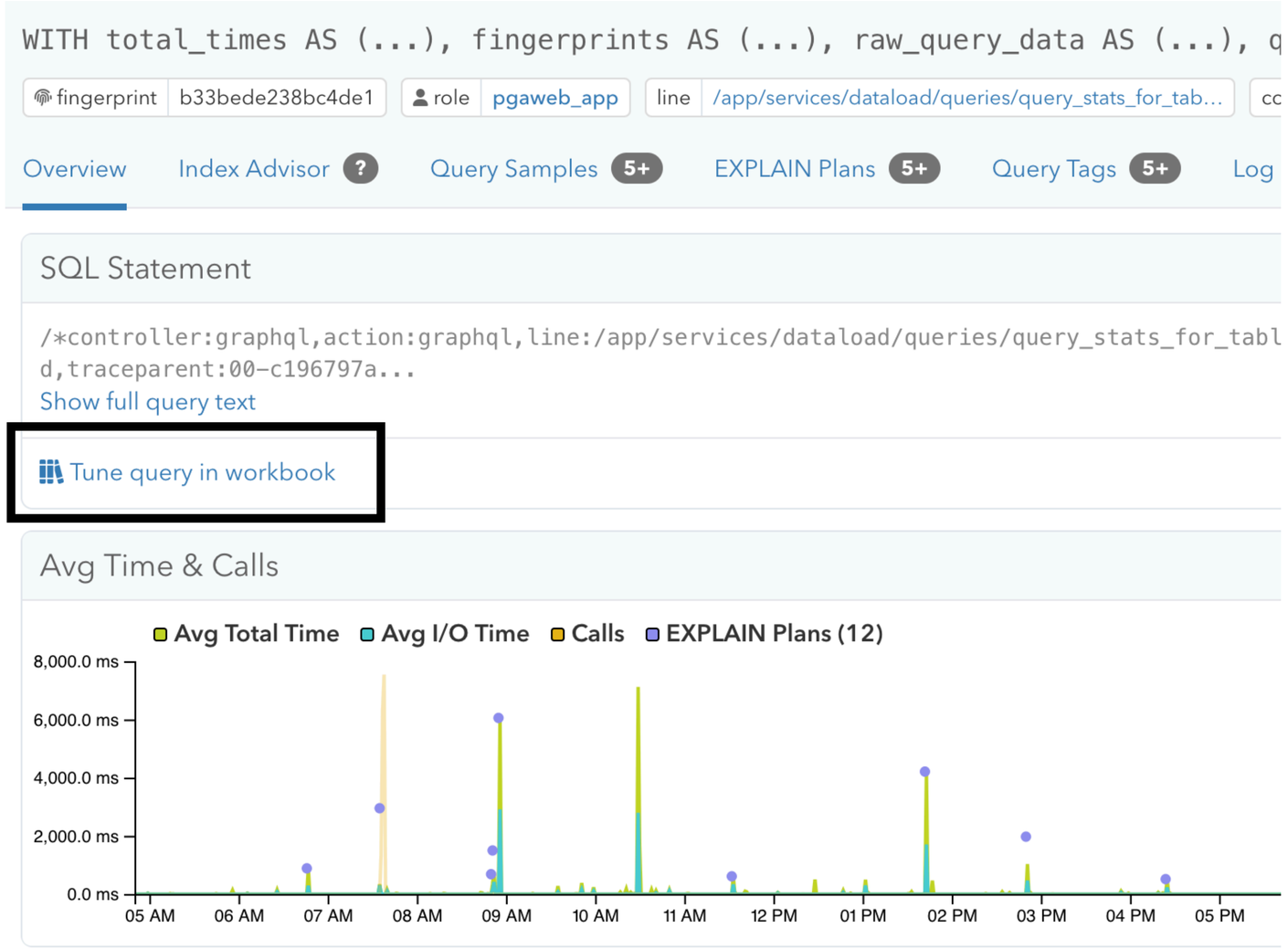
Development
Successfully merging this pull request may close these issues.
None yet

Notifications Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

3 participants

pganalyze

Let's Tune The Query!



Automatic Naming of Parameters

1 Review query 2 Choose parameters 3 Run EXPLAIN

The query was reformatted, normalized, and positional parameters (`$1`) were turned into named parameters (`$param`). Names are auto-assigned and suffixed with consecutive numbers–for example, `param` , `param_2` , and `param_3` . Parameters can be merged by renaming to an existing name.

```
/*job:Storage::RecheckMissingIndexWorker,line:/app/services/dataload/queries/query_stats_for_index_advisor.rb:65:in `query_stats_for_index_advisor',sentry_trace_id:fbb91171cbe64102a85bad6585d7a95c,tracestate:pganalyze=t:1753833646.5048623*/
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
  WHERE
    database_id = $database_id_2
    AND collected_at BETWEEN $collected_at_3 AND $collected_at_4
), raw_query_data AS (
  (
    (
      (
        SELECT
          fingerprint, sum(calls) AS calls, sum(ROWS) AS ROWS,
          sum(total_time) AS total_time,
          sum(shared_blks_hit) AS shared_blks_hit,
          sum(shared_blks_read) AS shared_blks_read,
          sum(blk_read_time) AS blk_read_time,
          sum(blk_write_time) AS blk_write_time
        FROM unpack_query_stats(database_ids := ARRAY[$param_10], start_ts := $start_ts_2, end_ts := $end_ts_2)
        WHERE
          collected_at >= $collected_at_5
          AND CASE
            WHEN $param_4 >= $param_5 THEN collected_at < $collected_at_13
            ELSE collected_at <= $collected_at_8
          END
```

Choose Parameters

36 parameters detected

Rename Parameters

- array_to_string_arg
- database_id
- right_arg
- database_id_2
- collected_at_3
- collected_at_4
- param_10
- start_ts_2
- end_ts_2
- collected_at_5
- param_4
- param_5
- collected_at_13
- collected_at_8
- database_id_5
- collected_at_11

pganalyze

PR 4963 Dataload::Queries.query_stats_for_database optimization

Overview | Compare Plans | Parameter Sets

+ New Query Variant

All Query Plans

- Baseline
- Variant 1
- Variant 2
- Variant 4
- Variant 5 - materialize roles

Query Advisor
No insights found

Query
No match with existing queries

Query tags
No query tags

Baseline

With parameter names

```
1  WITH total_times AS (
2    SELECT
3      sum(query_stats_blk_read_time_sum + query_stats_blk_write_time_sum) AS total_iotime,
4      sum(query_stats_total_time_sum) AS total_runtime
5    FROM query_overview_stats_35d qos
6    WHERE
7      qos.database_id = $database_id_2
```

Show all code

Query Plans

PLAN	VARIANT	PARAMETER SET	EST. COST	RUNTIME
9d7d557	Baseline	Parameter Set 1	79,656	50,499.50ms

pganalyze

PR 4963 Dataload::Queries.query_stats_for_database optimization

Variant: Baseline | Parameter Set: Parameter Set 1

Overview | Compare Plans | Parameter Sets

Node Tree | Grid | Text | JSON

SQL Statement

```
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...
```

Show full query text

Show: ○ Est. Cost ◉ Runtime ○ Rows ○ Buffers ○ Reads ○ Writes

All metrics exclude children, except Rows. Learn more about reading EXPLAIN plans.

Plan	Runtime
1 Limit mis-estimate	0.03ms
47 └ Sort mis-estimate	1.10ms
48 └ Nested Loop mis-estimate	5,812.04ms
49 ├ Nested Loop inefficient nested loop mis-estimate	16.91ms
50 │ ├ Aggregate	18.77ms
51 │ │ └ CTE Scan on query_data_early_sort_limit	1,071.16ms
52 │ └ Index Scan (Forward) on queries	69.93ms ⟲8.7k
53 └ Aggregate	37,463.93ms ⟲8.7k
Group Key: query_data_early_sort_limit.fingerprint Partial Mode: Simple Strategy: Sorted	
54 └ Sort	6,034.93ms ⟲8.7k
55 └ Aggregate	4.32ms
56 └ CTE Scan on query_data_early_sort_limit	0.78ms
CTE total_times	
2 Aggregate	0.03ms
3 └ Index Scan (Forward) on query_overview_stats_35d_20250618	0.17ms
CTE raw_query_data	
4 Append	1.84ms
5 ├ Subquery Scan mis-estimate	2.19ms

Summary | Node Details | Node Source

Aggregate

Performs grouped or ungrouped aggregation on output of its child.

Group Key
`query_data_early_sort_limit.fingerprint`

Partial Mode
`Simple`

Strategy
`Sorted`

I/O & Buffers

☑ Include cumulative costs of node children

	Shared	Local	Temp
Hit	0 B	0 B	-
Read	0 B	0 B	0 B
Dirtied	0 B	0 B	-
Written	0 B	0 B	0 B

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

Output Columns

- query_data_early_sort_limit.fingerprint
- array_agg(query_data_early_sort_limit.postgres_role_id ORDER BY (sum(query_data_early_sort_limit.to

Get Help

pganalyze

Untitled Variant

Edit name and description

Cancel workflow

1 Rewrite Query and Edit Planner 2 Run EXPLAIN

Edit | Unified Diff | Split Diff

```
1  WITH total_times AS (
2    SELECT
3      sum(query_stats_blk_read_time_sum + query_stats_blk_write_time_sum) AS total_iotime,
4      sum(query_stats_total_time_sum) AS total_runtime
5    FROM query_overview_stats_35d qos
6    WHERE
7      qos.database_id = $database_id_2
8      AND qos.collected_at BETWEEN $collected_at_5 AND $collected_at_6
9  ), raw_query_data AS (
10   (
11     (
12       (
13         SELECT
14           fingerprint, postgres_role_id, sum(calls) AS calls,
15           sum(total_time) AS total_time,
16           sum(shared_blks_hit) AS shared_blks_hit,
17           sum(shared_blks_read) AS shared_blks_read,
18           sum(blk_read_time) AS blk_read_time,
19           sum(blk_write_time) AS blk_write_time
20         FROM unpack_query_stats(database_ids := ARRAY[$param_30], start_ts := $start_ts_3, end_ts :=
   $end_ts_3) qs
21         WHERE
22           collected_at >= $collected_at_7
23           AND CASE
24             WHEN $param_18 >= $param_19 THEN collected_at < $collected_at_23
25             ELSE collected_at <= $collected_at_13
26           END
27         GROUP BY 1, 2
28         UNION ALL
29         SELECT
30           fingerprint, postgres_role_id, sum(calls) AS calls,
```

Verify Query

Query Advisor

No insights found

Customize planner behavior by changing the settings and hints. Settings are only applied to the current session.

Edit Planner Settings

Edit Planner Hints

No custom planner settings

Get Help

pganalyze

Use query variants to test hypothesis

All Query Plans

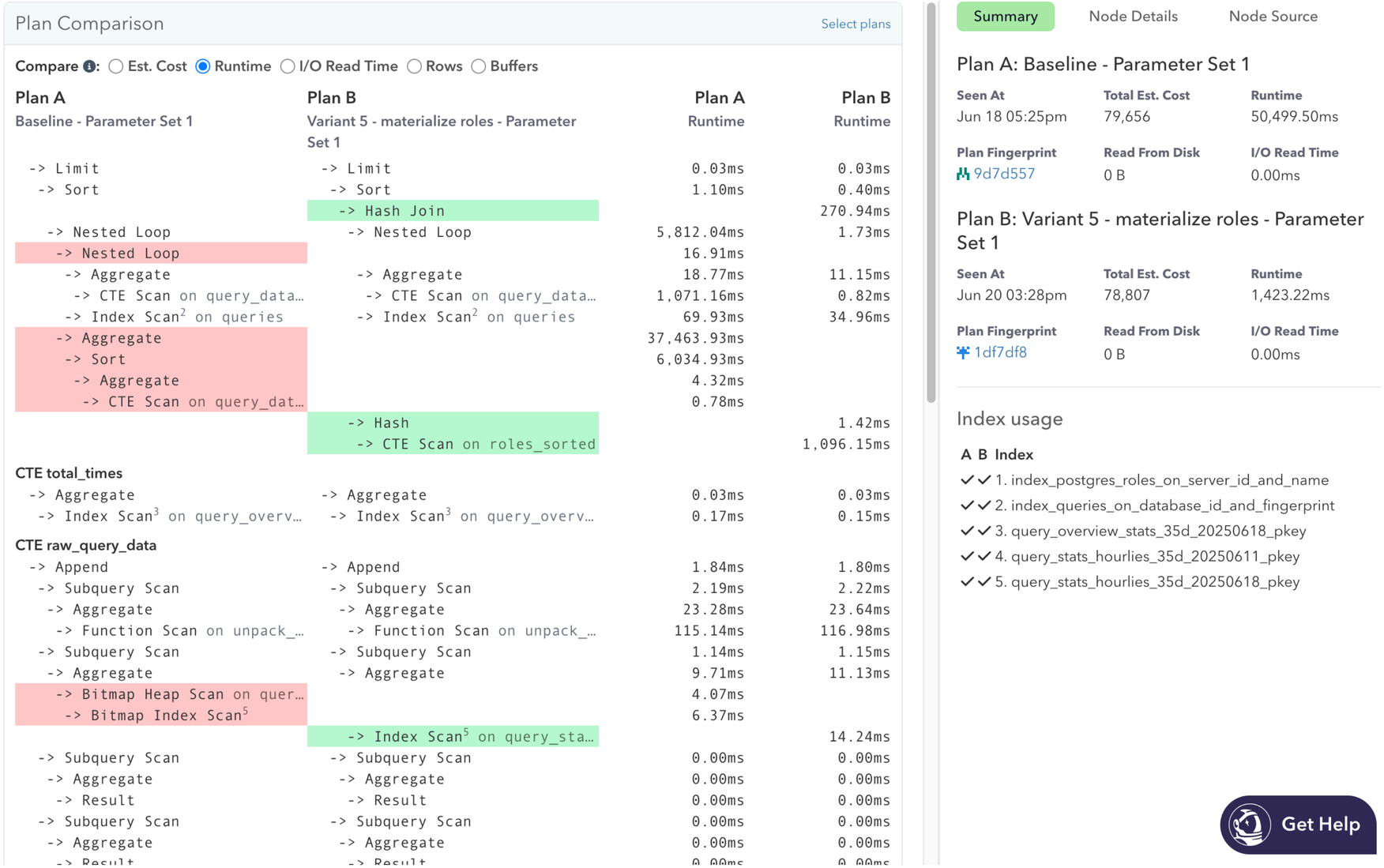
Choose a query variant to see its query text and settings, or create a new variant. Variants can be used to test Postgres planner behavior or rewrite queries to improve performance. Learn more.

Query Plans

Filter by Parameter Set...

PLAN	VARIANT	PARAMETER SET	EST. COST	RUNTIME
9d7d557	Baseline	Parameter Set 1	79,656	50,499.50ms
661d018	Variant 1	Parameter Set 1	79,671	2,868.94ms
d73907f	Variant 2	Parameter Set 1	78,822	2,844.51ms
3212a67	Variant 4	Parameter Set 1	78,824	2,128.92ms
1df7df8	Variant 5 - materialize roles	Parameter Set 1	78,807	1,423.22ms

See the impact of plan changes



pganalyze

What's new with Workbooks?

New "Grid" EXPLAIN View

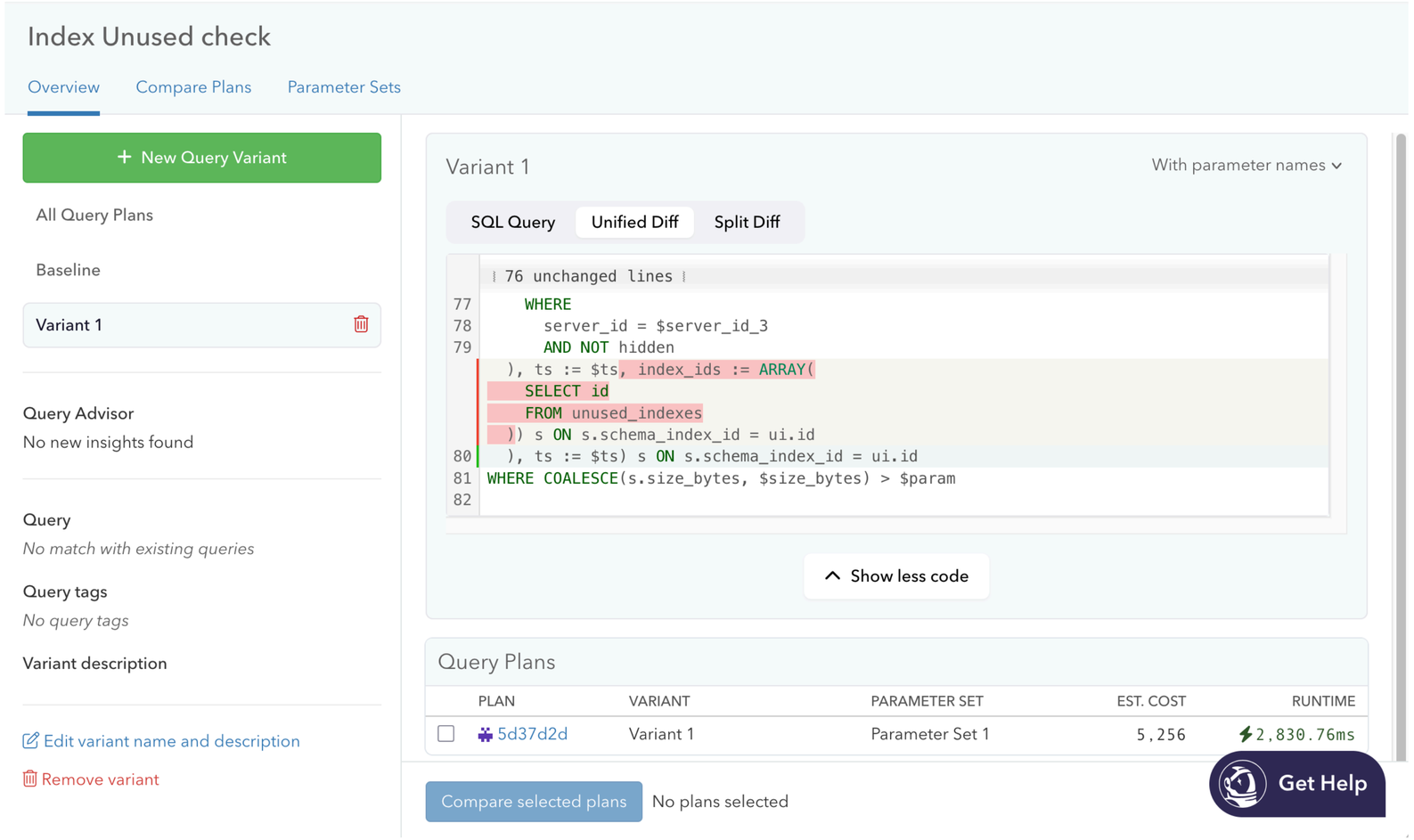
Node Tree | **Grid** | Text | JSON

Show: ○ Est. Cost ◉ Runtime ○ Rows ○ Buffers ○ Reads ○ Writes

All metrics exclude children, except Rows. Learn more about reading EXPLAIN plans.

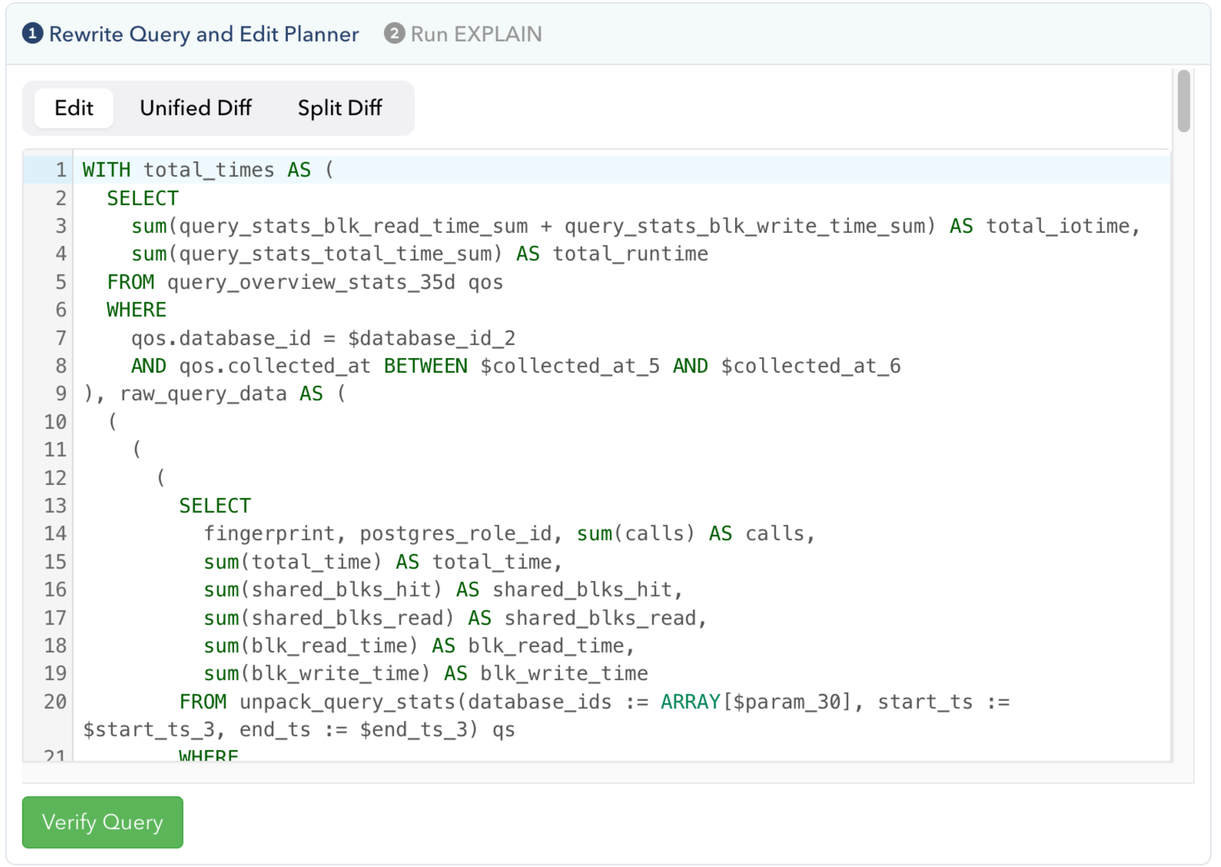
	Plan	Runtime	
1	Limit mis-estimate	0.03ms	
47	└ Sort mis-estimate	1.10ms	
48	└ Nested Loop mis-estimate	5,812.04ms	
49	├ Nested Loop inefficient nested loop mis-estimate	16.91ms	
50	│ ├ Aggregate	18.77ms	
51	│ │ └ CTE Scan on query_data_early_sort_limit	1,071.16ms	
52	│ └ Index Scan (Forward) on queries	69.93ms	⟲8.7k
53	└ Aggregate	37,463.93ms	⟲8.7k
54	└ Sort	6,034.93ms	⟲8.7k
55	└ Aggregate	4.32ms	
56	└ CTE Scan on query_data_early_sort_limit	0.78ms	
	CTE total_times		
2	Aggregate	0.03ms	
3	└ Index Scan (Forward) on query_overview_stats_35d_20250618	0.17ms	
	CTE raw_query_data		
4	Append	1.84ms	
5	├ Subquery Scan mis-estimate	2.19ms	
6	│ └ Aggregate mis-estimate	23.28ms	
7	│ └ f(x) Function Scan on unpack_query_stats(…) mis-estimate	115.14ms	
8	├ Subquery Scan	1.14ms	
9	│ └ Aggregate	9.71ms	

Built-in Diff to compare against Baseline Query Text



pganalyze

Built-in SQL formatting



pganalyze

Better IN list handling in Parameter Sets

```
column1 IN ($1, $2, $3, $4, $5)
```

=>

```
column1 = ANY($1)
```

Planner Settings

Edit Planner Settings

Custom planner settings

enable_incremental_sort	off (was on)
enable_memoize	off (was on)

Select a planner setting to edit

work_mem | 204800 | (unit: kB) | Save setting

work_mem (integer)

Sets the base maximum amount of memory to be used by a query operation (such as a sort or hash table) before writing to temporary disk files. If this value is specified without units, it is taken as kilobytes. The default value is four megabytes (4MB). Note that a complex query might perform several sort and hash operations at the same time, with each operation generally being allowed to use as much memory as this value specifies before it starts to write data into temporary files. Also, several running sessions could be doing such operations concurrently. Therefore, the total memory used could be many times the value of work_mem; it is necessary to keep this fact in mind when choosing the value. Sort operations are used for ORDER BY, DISTINCT, and merge joins. Hash tables are used in hash joins, hash-based aggregation, memoize nodes and hash-based processing of IN subqueries.

Hash-based operations are generally more sensitive to memory availability than equivalent sort-based operations. The memory limit for a hash table is computed by multiplying work_mem by **hash_mem_multiplier**. This makes it possible for hash-based operations to use an amount of memory that exceeds the usual work_mem base amount.

Source: PostgreSQL 16 documentation

pganalyze

Planner Hints

pganalyze

pganalyze Query Tuning Workbooks

pganalyze ~~Query Tuning~~ Workbooks

pganalyze Workbooks

pganalyze

Introducing pganalyze Query Advisor

[Live Demo]

pganalyze

Common Problems detected by Query Advisor

Inefficient Nested Loop

EXPLAIN Plan

Show: ○ Est. Cost ○ Runtime ◉ Rows ○ Buffers ○ Reads ○ Writes

All metrics exclude children, except Rows. Learn more about reading EXPLAIN plans.

Plan	Rows	Est. Rows
1 Nested Loop nested loop	0	1
2 Index Scan (Forward) on issues outer mis-estimate	175	1
on public.issues using index_issues_on_server_id_and_severity		
Filter: ((issues.state = ANY ($1::integer[])) AND ((issues.database_id = ANY ($2::bigint[…		
Index Cond: ((issues.server_id = ANY ($1::uuid[])) AND (issues.server_id = $2::uuid))		
Rows Removed by Filter: 0		
Rows Removed by Index Recheck: 0		
Scan Direction: Forward		
3 Index Scan (Forward) on issue_references inner	0	1 ⟲175

Query Summary

Runtime	I/O Read Time
667.00ms	665.24ms
Read From Disk	**Total Est. Cost**
648 kB	15

Jul 25 06:03:20am PDT · Plan Fingerprint: db5ccf2

Insights (2)

Query Advisor

Inefficient Nested Loop

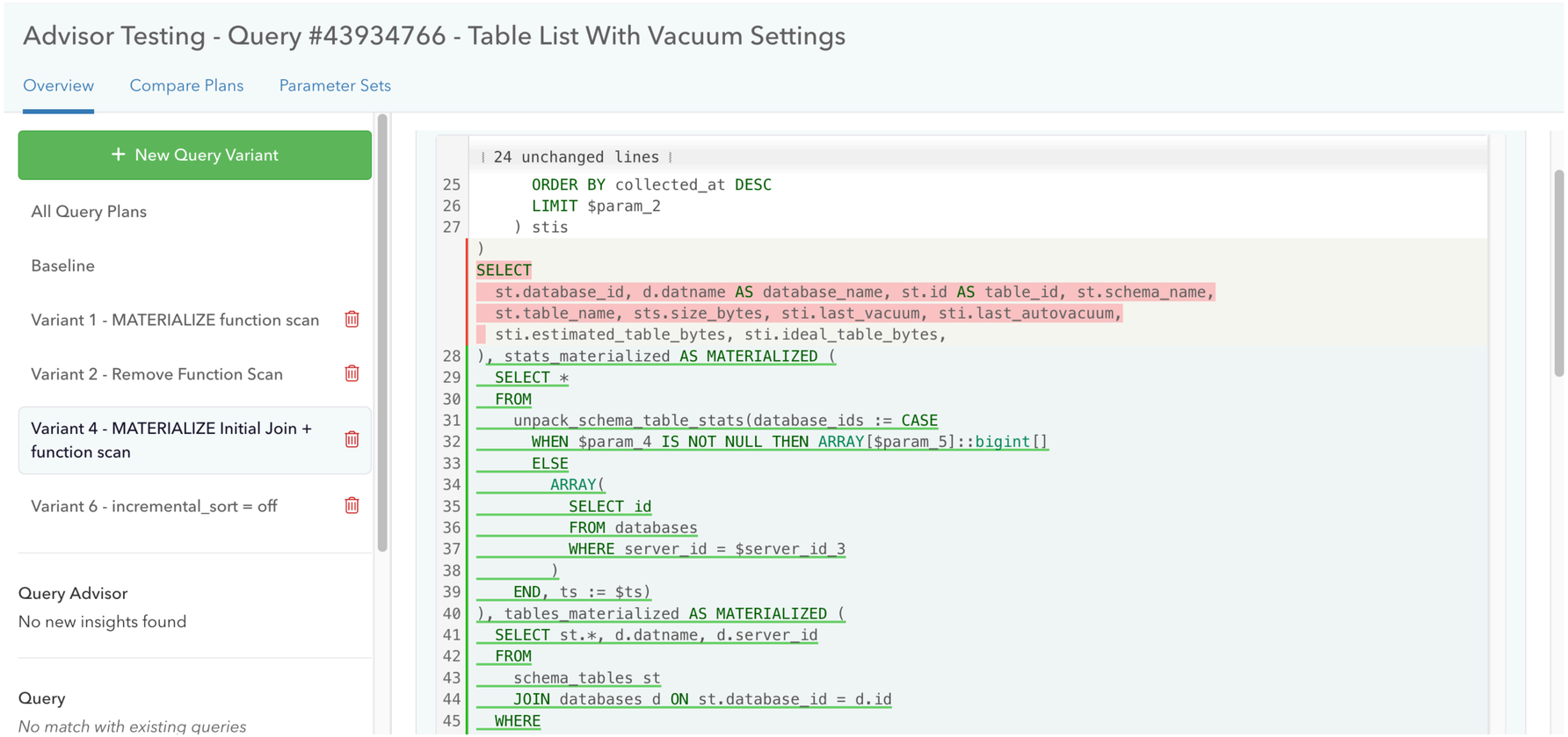
- Estimated 1 loops, but 175 were executed.
- Inner side of this nested loop takes 666.40ms (100% of total)

Create Workbook

Other ›

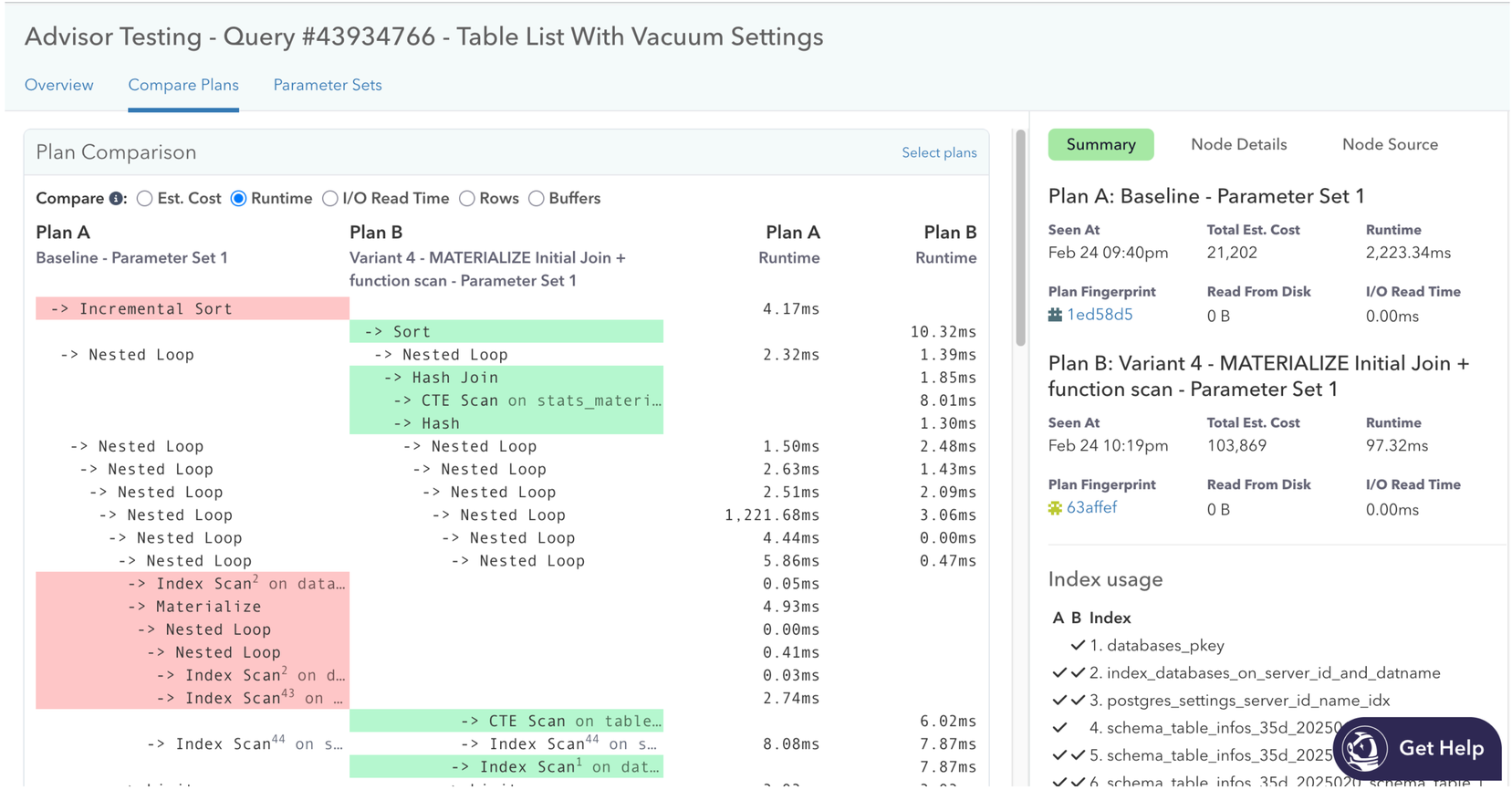
pganalyze

Inefficient Nested Loop



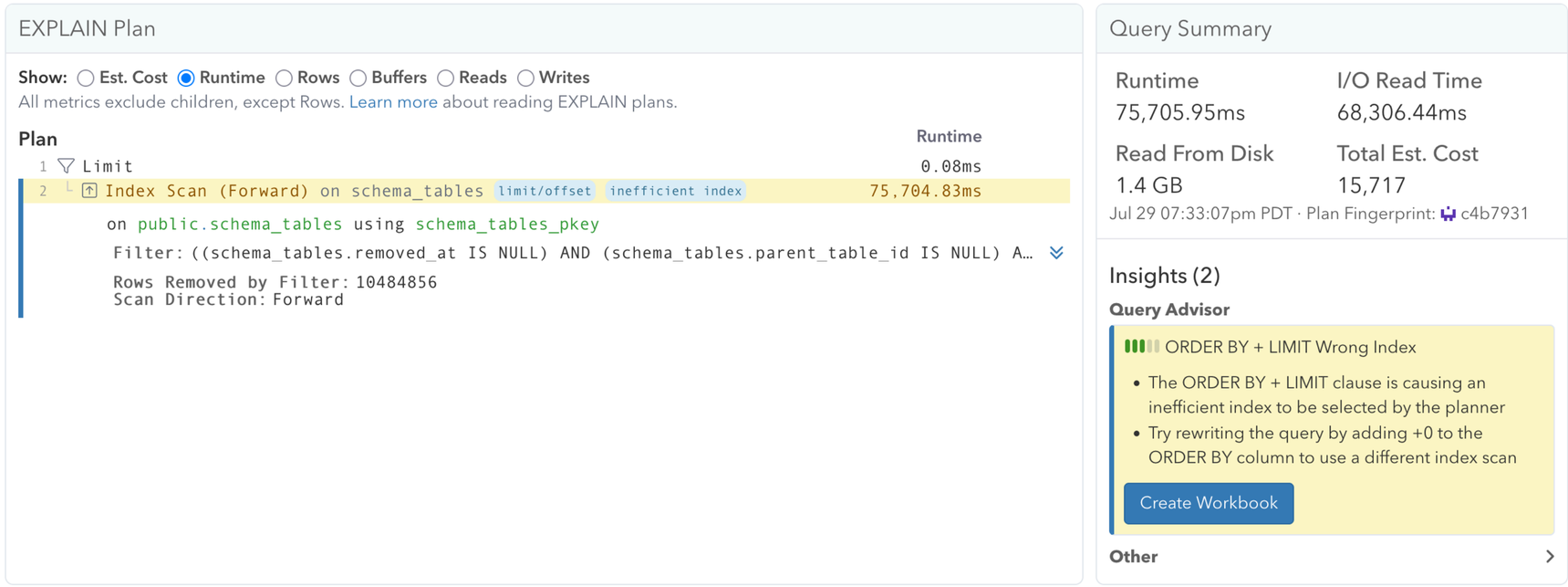
pganalyze

Inefficient Nested Loop



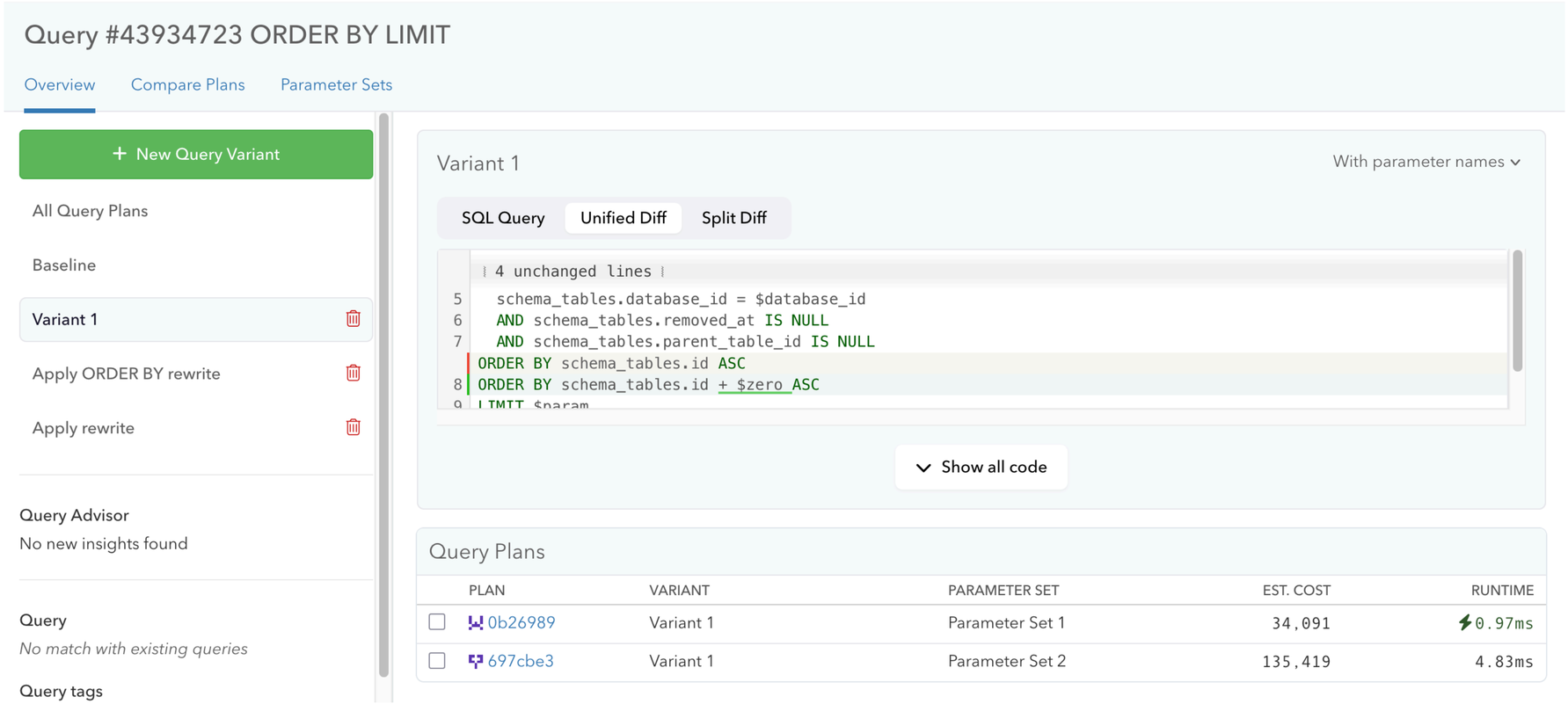
pganalyze

ORDER BY + LIMIT Wrong Index Use



pganalyze

ORDER BY + LIMIT Wrong Index Use



pganalyze

ORDER BY + LIMIT Wrong Index Use

Plan Comparison — Select plans

Compare: ○ Est. Cost ◉ Runtime ○ I/O Read Time ○ Rows ○ Buffers

Plan A Baseline - Parameter Set 2	Plan B Variant 1 - Parameter Set 2	Plan A Runtime	Plan B Runtime
-> Limit	-> Limit	0.00ms	0.00ms
-> Index Scan[2] on schema_tables		6,683.13ms	
	-> Gather Merge		4.56ms
	-> Sort		0.03ms
	-> Bitmap Heap Scan on sche…		0.16ms
	-> Bitmap Index Scan[1]		0.07ms

Summary | Node Details | Node Source

Plan A: Baseline - Parameter Set 2

Seen At	Total Est. Cost	Runtime
Jul 24 12:31am	8,577	6,683.23ms

Plan Fingerprint	Read From Disk	I/O Read Time
c4b7931	0 B	0.00ms

Plan B: Variant 1 - Parameter Set 2

Seen At	Total Est. Cost	Runtime
Jul 24 12:37am	135,419	4.83ms

Plan Fingerprint	Read From Disk	I/O Read Time
697cbe3	0 B	0.00ms

Index usage

A	B	Index
	✓	1. schema_tables_database_id_schema_name_table_…
✓		2. schema_tables_pkey

pganalyze

Additional insights in the works:

- OR => UNION transformation
- Planner hint suggestions
- Memoize mis-estimates
- GROUP BY column ordering
- Settings changes
  - work_mem
  - random_page_cost
  - etc.

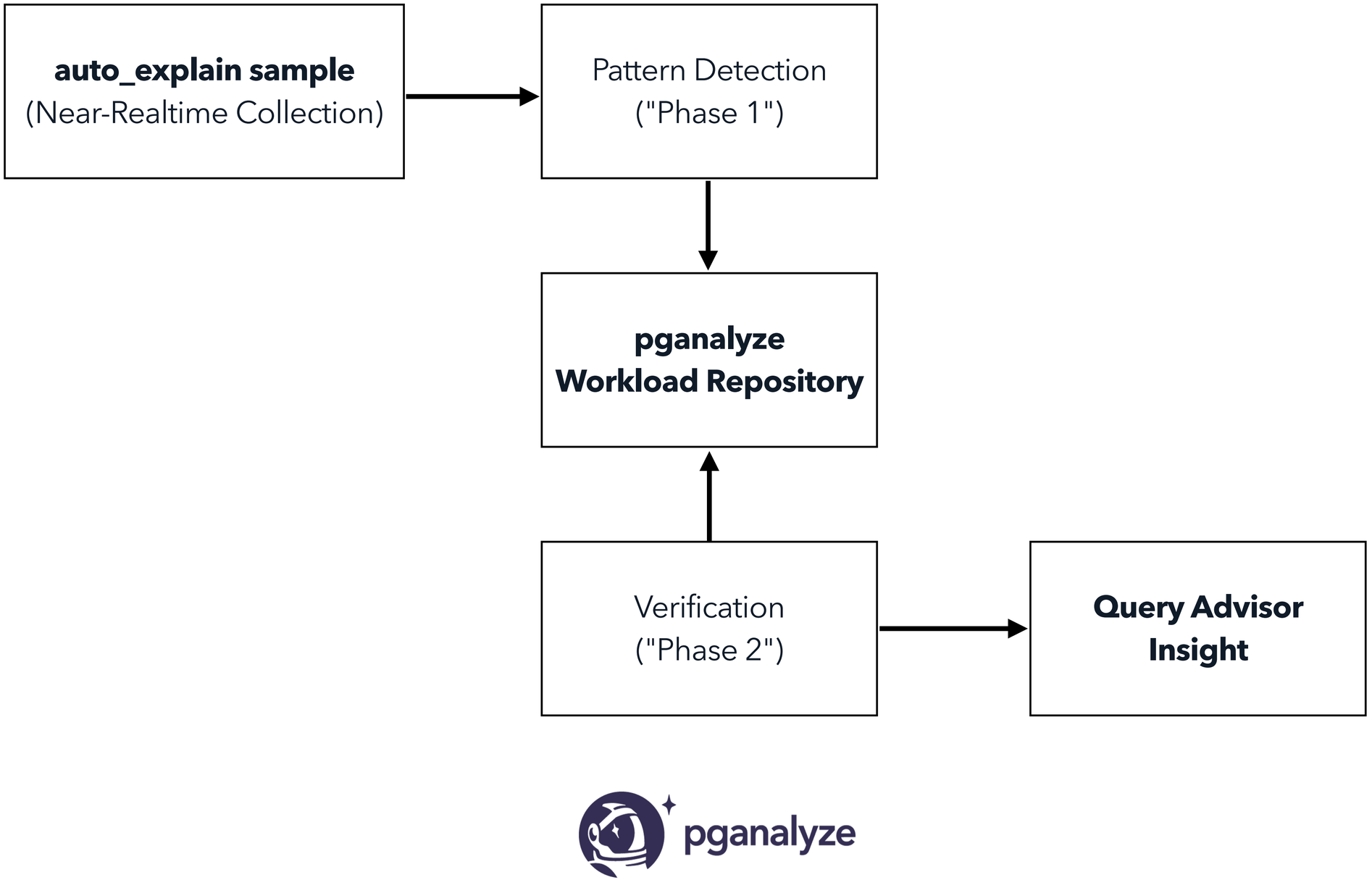
pganalyze

Behind the scenes of pganalyze Query Advisor

Query Tuning Advisor at a High Level

- **Built for scale, running behind the scenes**
- **Currently processing 7 million samples per day**
- **Custom code, not using LLM-driven analysis**

(LLMs don't scale for automated analysis)

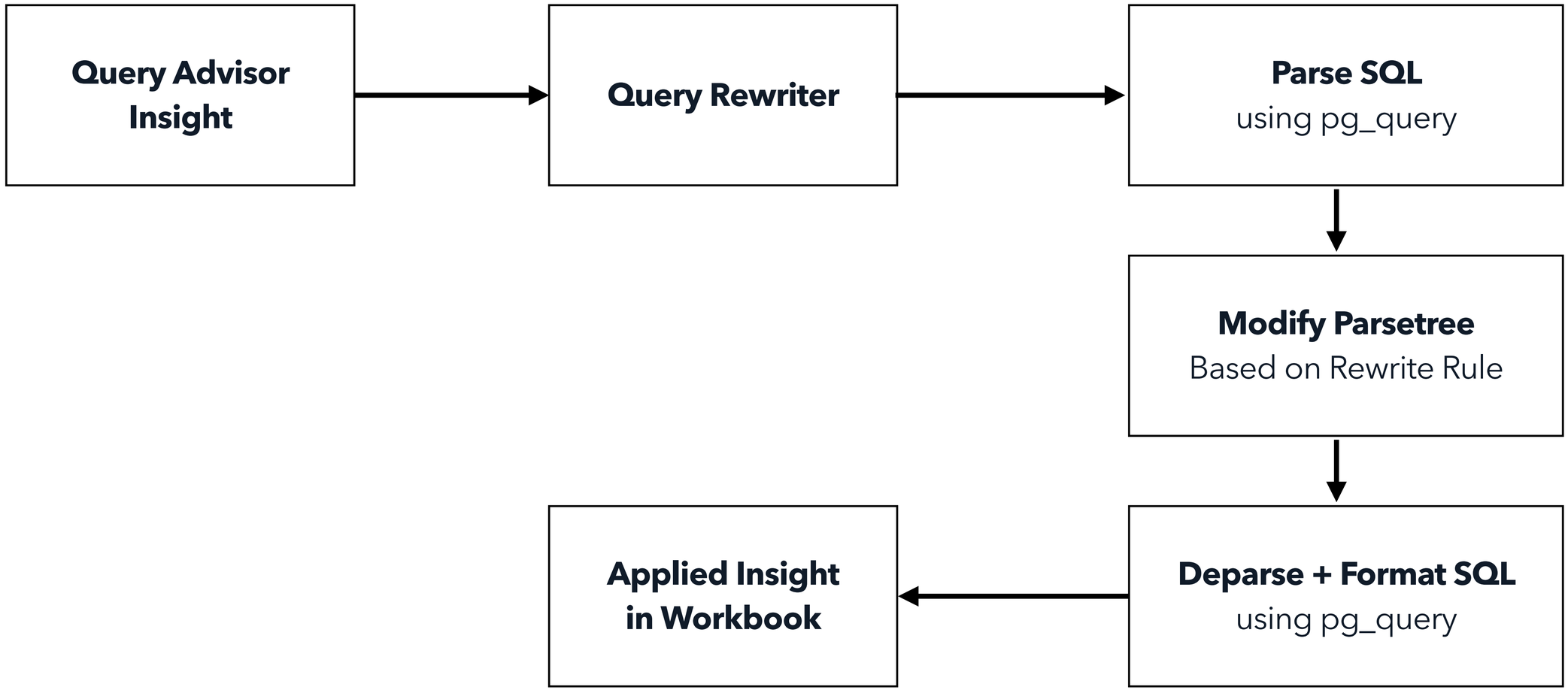

auto_explain sample
(Near-Realtime Collection)
Pattern Detection
("Phase 1")
pganalyze
Workload Repository
Verification
("Phase 2")
Query Advisor
Insight

pganalyze


pganalyze Workbook
EXPLAIN Upload
Pattern Detection +
Verification
Query Advisor
Insight

pganalyze

"Codemods" for queries


Query Advisor Insight
Query Rewriter
Parse SQL
using pg_query
Modify Parsetree
Based on Rewrite Rule
Deparse + Format SQL
using pg_query
Applied Insight in Workbook

pganalyze

pganalyze

Early Access & Looking Ahead

For best results:

- Detecting problematic queries in the background
  requires auto_explain with log_analyze (but not log_timing)
- Workbook Collector workflow makes testing easy

Early Access Program starts August 11th

Let us know if you're interested through
the post-webinar survey, or by sending an email.

Looking Ahead:

- Expanded coverage of common Postgres problems
- Automatic testing of rewrites in the background (opt-in)
- Exploring how Workbooks and Query Advisor can be a tool that integrates into pre-production tests or LLM-based workflows

Thank you!

Try out pganalyze:

PGANALYZE.COM

Reach out for any questions:

lukas@pganalyze.com
