pganalyze in Action

The Latest Features for Tuning Postgres

lukas@pganalyze.com

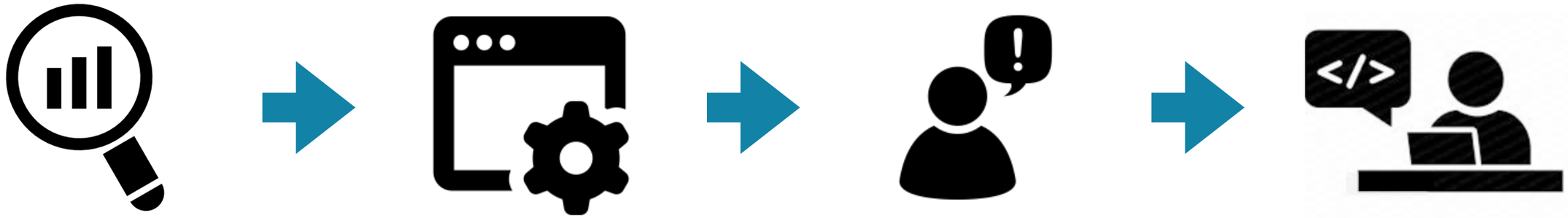
pganalyze

Lukas Fittl

CEO & Founder

Why pganalyze?

When Postgres Slows Down, You Feel It Everywhere



Slow Query

App Latency

User Complaints

Developer Scramble

pganalyze

Engineering Teams Succeed With an End-to-End Solution Built for Postgres

Generic tools:
fragmented & hard to use

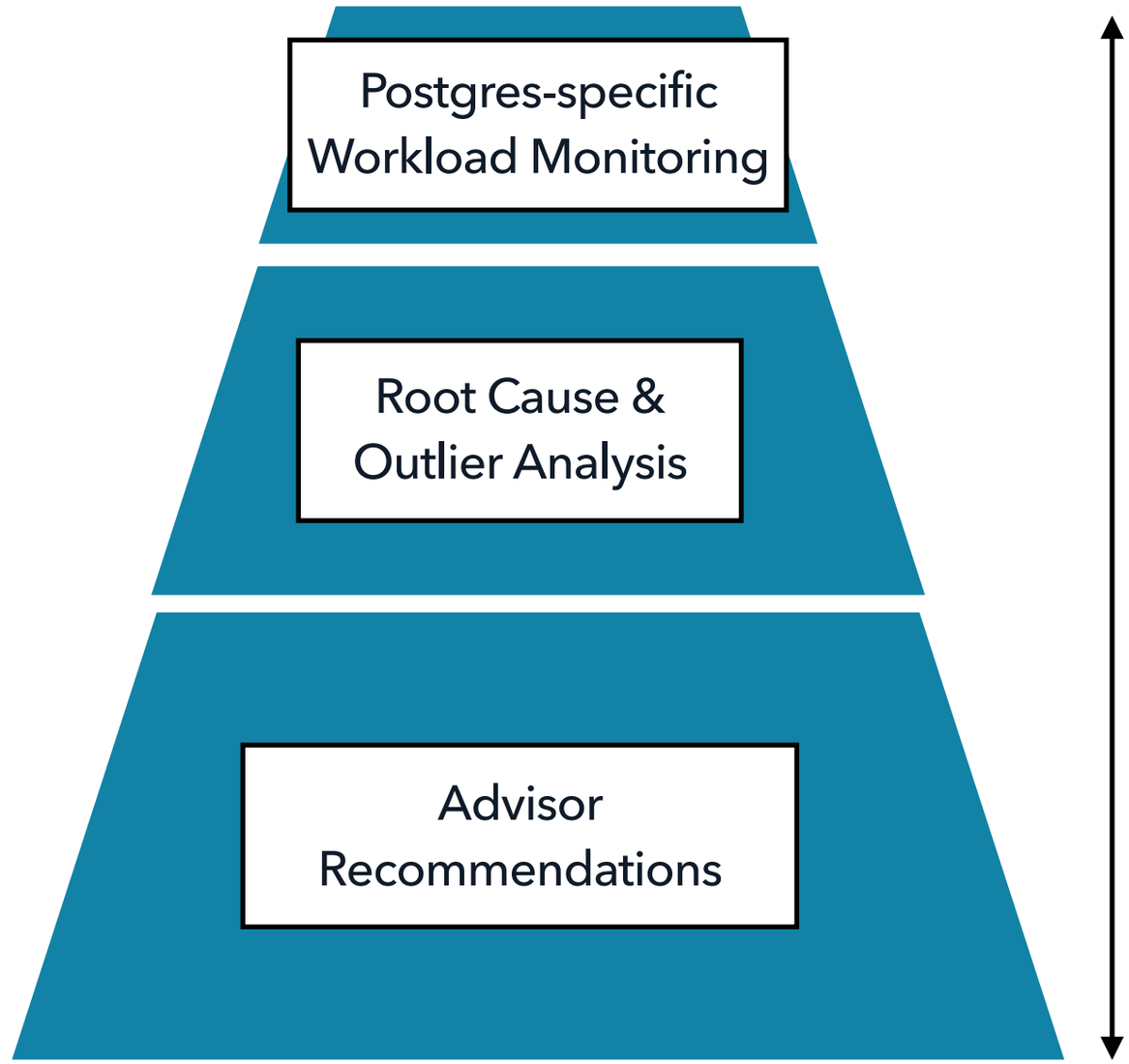
APM
High Level Stats

Custom Scripts

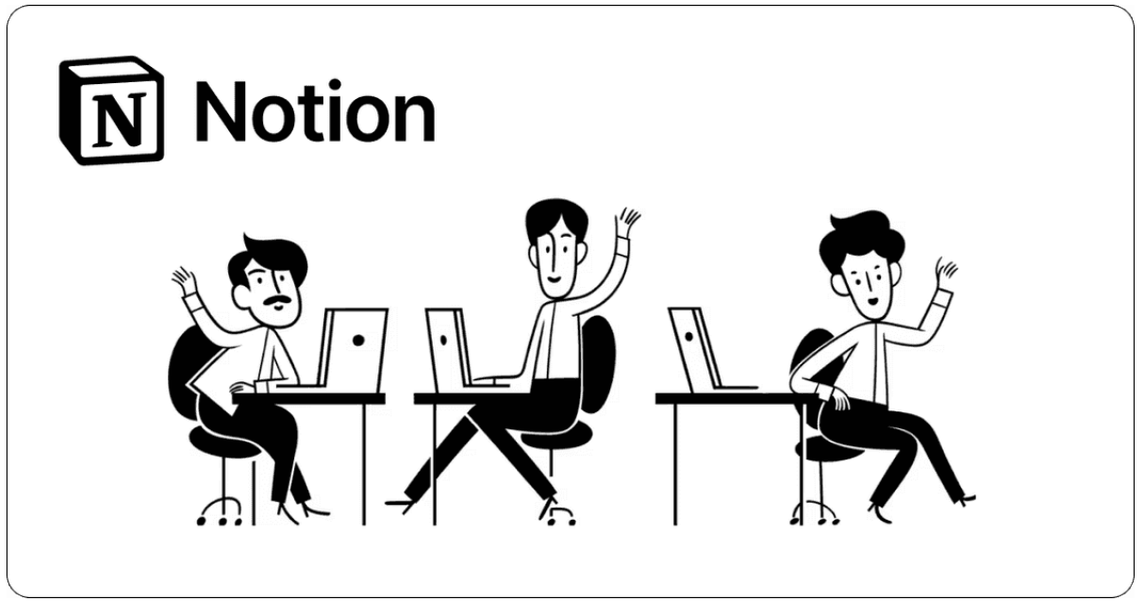
Manual EXPLAIN plan guesswork

Combing through logs

pganalyze: unified & purpose-built for Postgres



How Notion Runs Postgres at Scale with pganalyze



* 733% performance improvement
* 8x query runtime reduction
* Prevents production incidents

> “During incidents, pganalyze gives us the visibility to pinpoint what’s happening. It’s especially helpful for understanding spikes in CPU or query load and distinguishing between root causes and symptoms.”

– Arka Ganguli, Engineering Manager, Notion

pganalyze

Use pganalyze in the cloud, and on-premise

Amazon Web Services

RDS and Aurora

Microsoft Azure

Flexible Server

Google Cloud

Cloud SQL and AlloyDB

Self-Managed

VM, Container, On-Premise

Heroku

Heroku Postgres

Crunchy Data

Crunchy Bridge

Aiven

Managed PostgreSQL

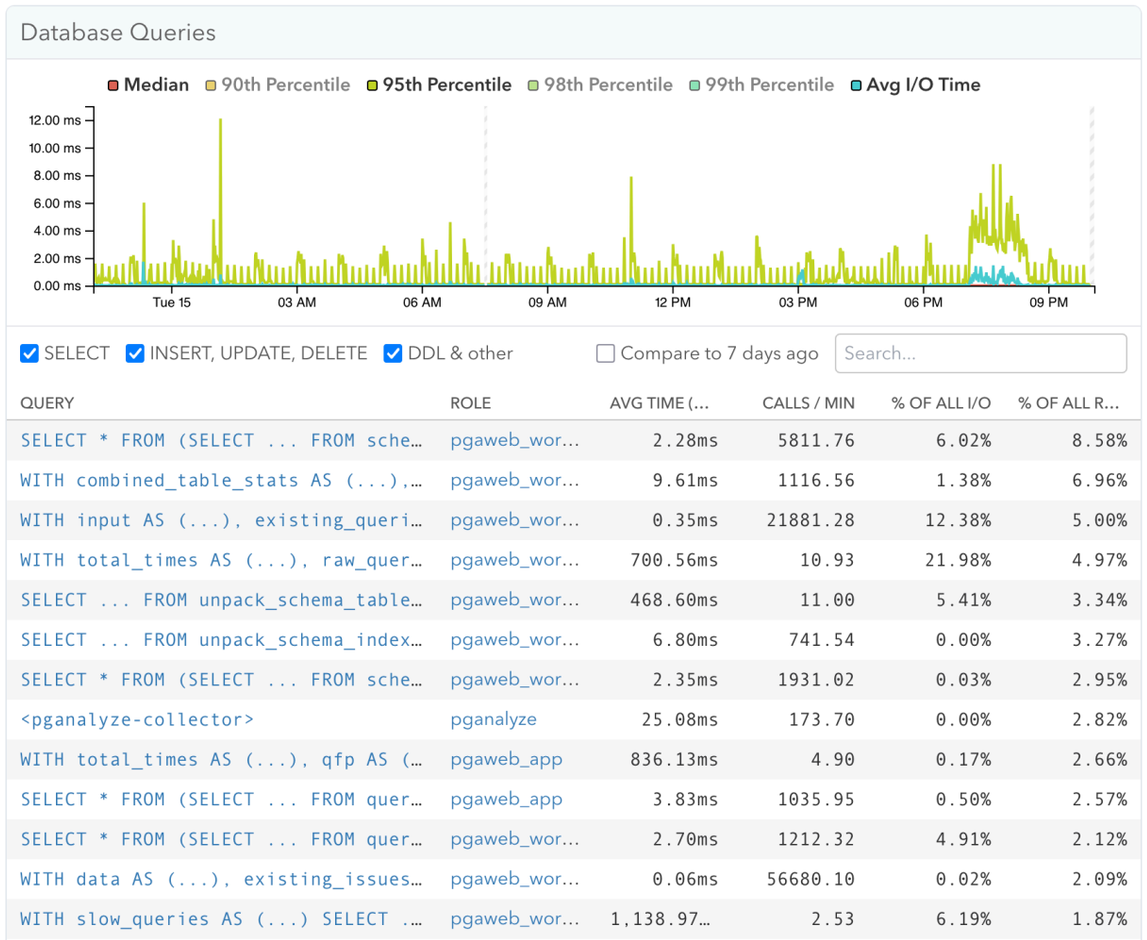
pganalyze

pganalyze

Feature Overview

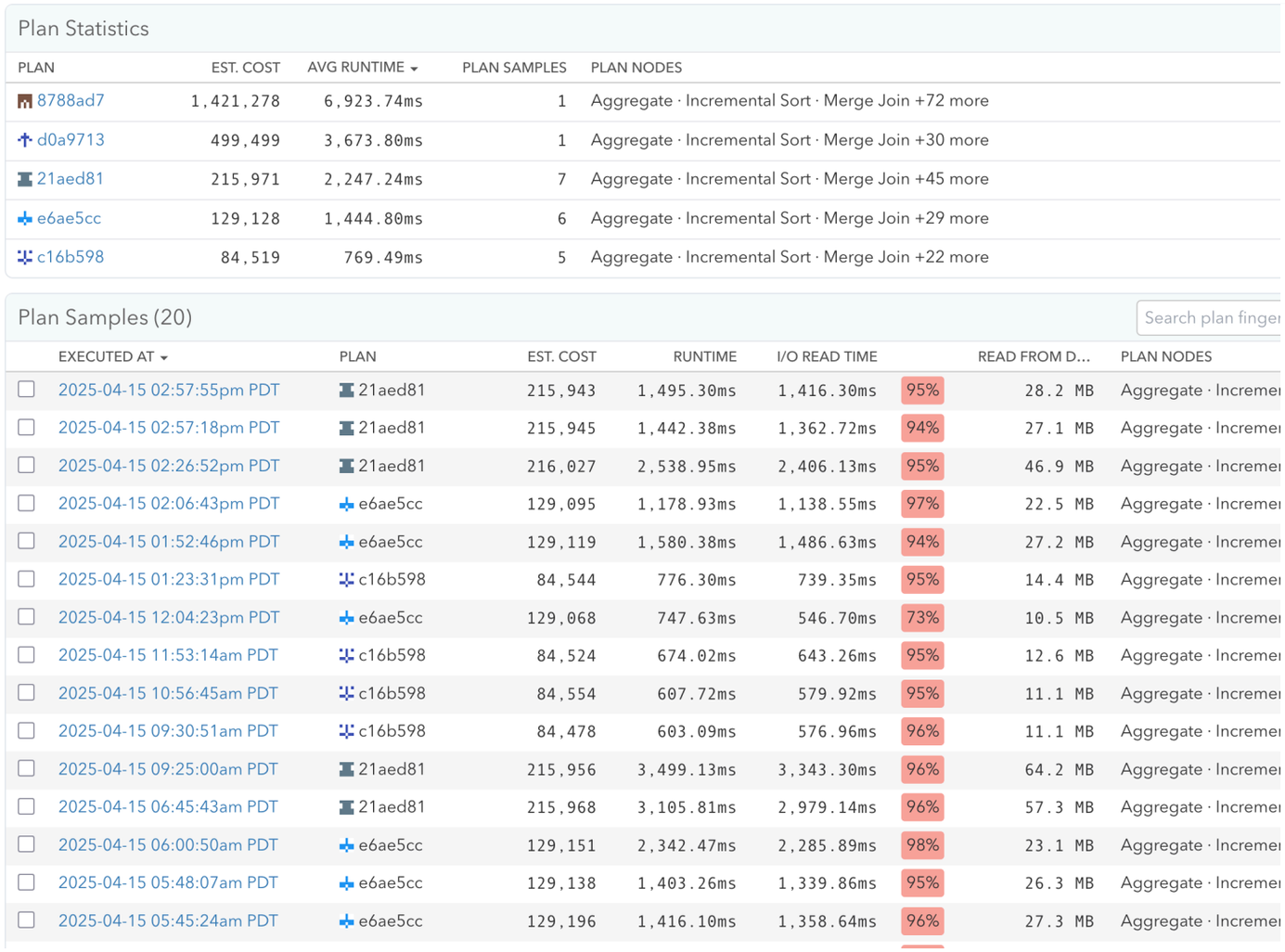
Query Performance Monitoring

- Captures all queries executed, fast and slow.
- Detailed information for each query, over time.
- Queries are grouped by query fingerprint (powered by pg_query)



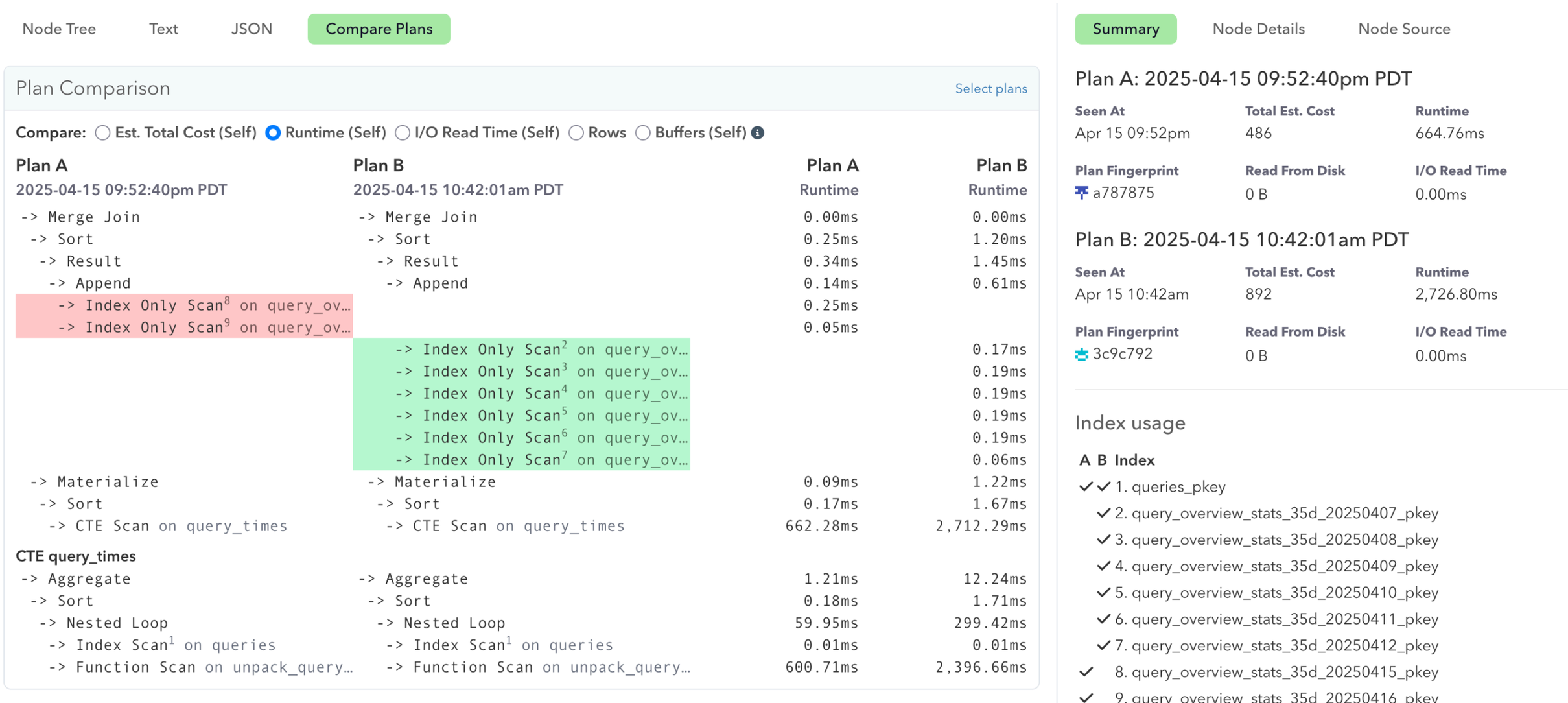
Automated EXPLAIN

- Automatically gather query plans for slow executions
- Utilizes auto_explain or slow query log (log_min_duration_statement)
- Best data source for knowing the root cause of a slow outlier



Plan Comparison (new!)

- Purpose-built diff mechanism for EXPLAIN plans
- Shows structural differences in the plan shape
- Allows comparison between different per-node metrics across plans

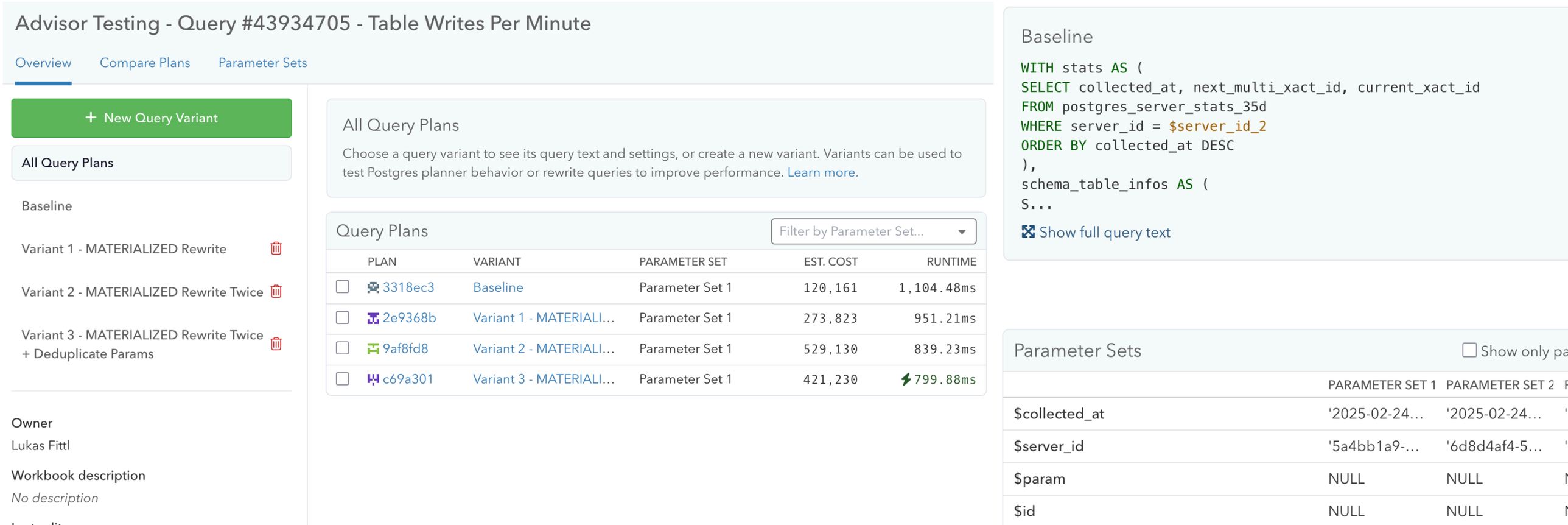


Plan Statistics (new!)

- Allows tracking of different plans used for a query over time
- Aimed at tracking plan shapes, best used in combination with auto_explain for tracking outliers and their execution statistics
- Currently requires AWS Aurora's aurora_stat_plans extension
- pganalyze will release a new pg_stat_plans extension later this year

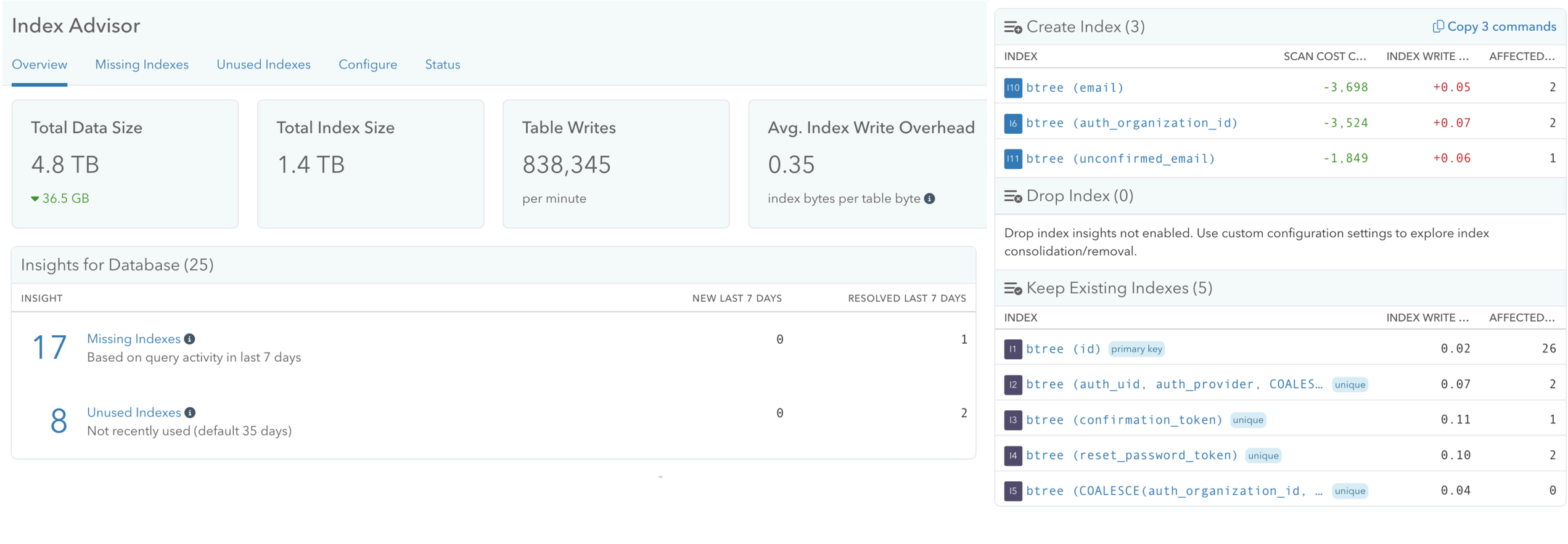
Query Tuning (new!)

- Allows running EXPLAIN ANALYZE benchmarks through pganalyze
- Explicit testing of different parameter values for the same query, to ensure fixing the query for one customer doesn't regress for others
- Query Tuning Advisor coming later this year that makes recommendations on query rewrites or other ways to fix bad plans



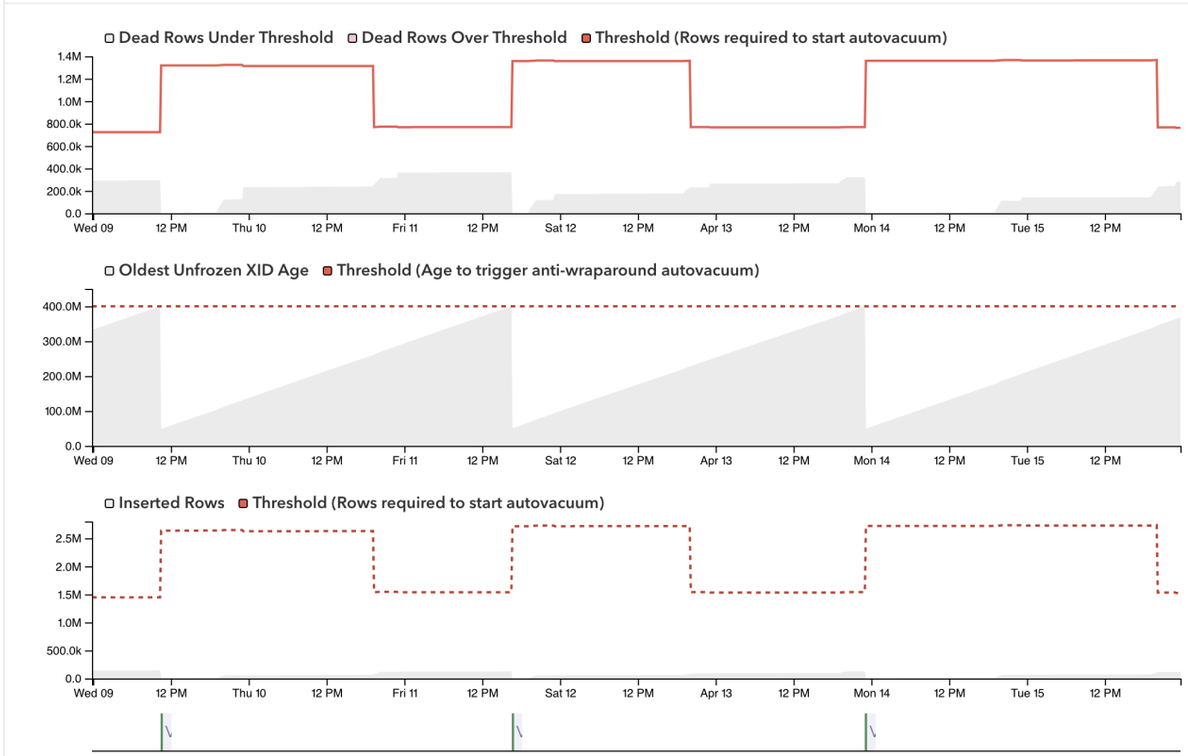
Index Advisor

- Detects missing and unused indexes in your database
- Missing index detection is based on a Constraint Programming model that analyzes all queries on a given table for their WHERE/JOIN conditions and finds the optimal set of indexes
- Currently supports B-Tree and GIST indexes, other types planned



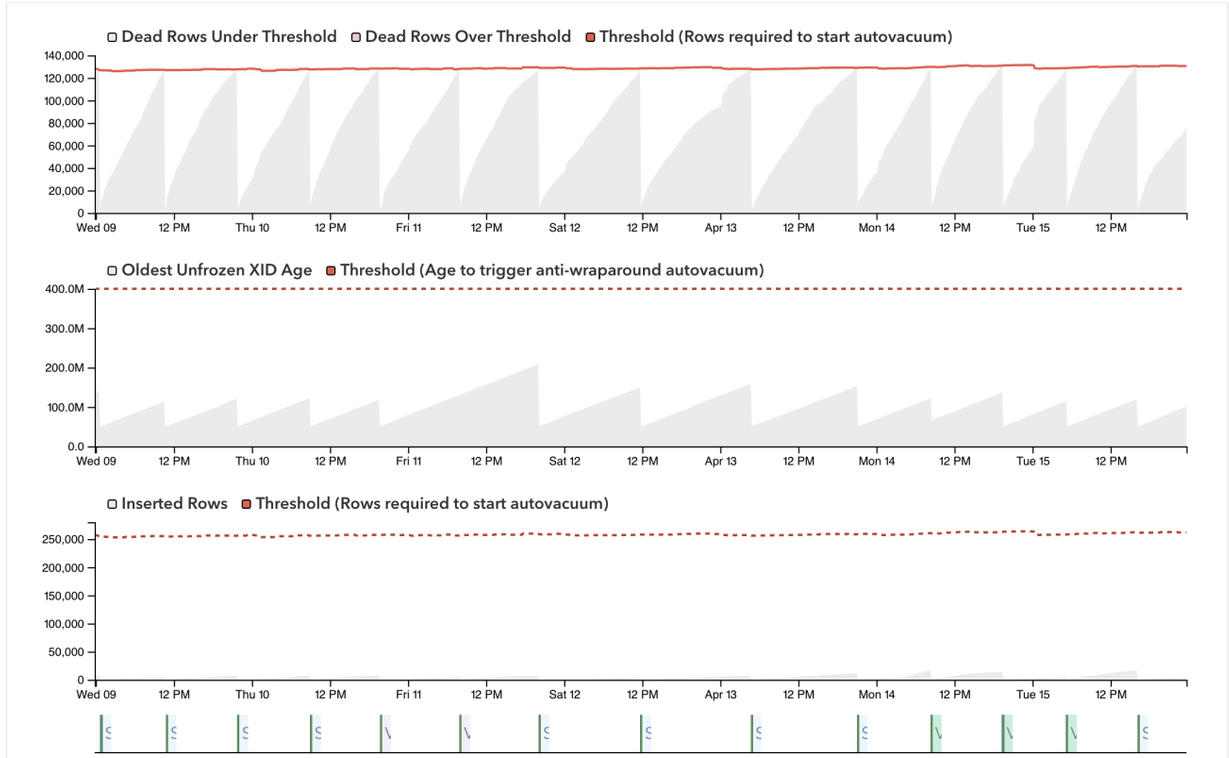
VACUUM Monitoring

- Tracks detailed VACUUM information on a per-table basis, over time
- Can clearly show why Postgres started an autovacuum on a table (whether it was based on dead rows, inserted rows, or TXIDs)
- Helps identify problematic VACUUM behavior, e.g. too few VACUUMs



Show TOAST VACUUMs

ID	STARTED BY	TOAST	TIME STARTED	TIME FINISHED	FREEZING PROGRESS	DEAD TUPLE PROGRESS
17446105440020171	autovacuum	No	Apr 13 11:02:23pm PDT	Apr 13 11:03:23pm PDT	+351,626,387	-321,897
17444142450014302	autovacuum	No	Apr 11 04:30:45pm PDT	Apr 11 04:31:50pm PDT	+350,502,160	-367,024
17442192180003707	autovacuum	No	Apr 09 10:20:17am PDT	Apr 09 10:20:59am PDT	+352,550,892	-295,451



Show TOAST VACUUMs

ID	STARTED BY	TOAST	TIME STARTED	TIME FINISHED	FREEZING PROGRESS	DEAD TUPLE PROGRESS
17447586340020837	autovacuum	No	Apr 15 04:10:33pm PDT	Apr 15 04:10:46pm PDT	+0	+6,556
17447190790020673	autovacuum	No	Apr 15 05:11:18am PDT	Apr 15 05:11:40am PDT	+65,294,379	-125,085
17446836160014869	autovacuum	No	Apr 14 07:20:15pm PDT	Apr 14 07:21:01pm PDT	+87,823,865	-126,203

VACUUM Advisor

- Detects common configuration problems with autovacuum
- Estimates bloat on tables, and notifies you when new bloat occurred and how to tune autovacuum settings to avoid it
- Notifies of blocked VACUUMs & identifies the root cause (e.g. idle TX)

VACUUM Advisor

Overview | Bloat | Freezing | Performance | Activity

Avg. Autovacuum Workers
0.15
out of 4 max workers

Total Autovacuum Count
1,160
in the last 24 hours

Total Anti-Wraparound Autovacuums
417 out of 1,160
in the last 24 hours

Insights for Server (0) — Based on table activity in last 7 days

INSIGHT		NEW LAST 7 DAYS	RESOLVED LAST 7 DAYS
0	Bloat Insufficient VACUUM Frequency VACUUM Blocked By Xmin Horizon Learn more in documentation	0	0
0	Freezing Approaching TXID Wraparound Approaching MXID Wraparound Learn more in documentation	0	0
0	Performance Inefficient Index Phase Learn more in documentation	0	0

Issue #21019: VACUUM: Bloat - Insufficient VACUUM Frequency

Overview

Severity	Check Frequency	Last Updated	State
Info	Daily	2025-04-08 07:00:51pm PDT	Res

Description
Insufficient vacuuming is causing avoidable growth on table public.issue_references

Guidance

```
ALTER TABLE public.issue_references SET (autovacuum_vacuum_scale_factor = 0.01);
```
Copy ALTER TABLE command

Avoidable table growth due to insufficient VACUUM frequency was detected. You can adjust the frequency of autovacuum by changing relevant config settings. Lowering `autovacuum_vacuum_threshold` or `autovacuum_vacuum_scale_factor` will increase the frequency of autovacuum in general.

› How to apply changes

› What to check following ALTER TABLE

To confirm the impact of changes, it is also recommended to run pg_repack to reclaim disk space. You can check out a graph of estimated bloat over time on the VACUUM/ANALYZE Activity page.

Recommendation

	Current	Recomme
autovacuum_vacuum_threshold	50	
autovacuum_vacuum_scale_factor	0.10	

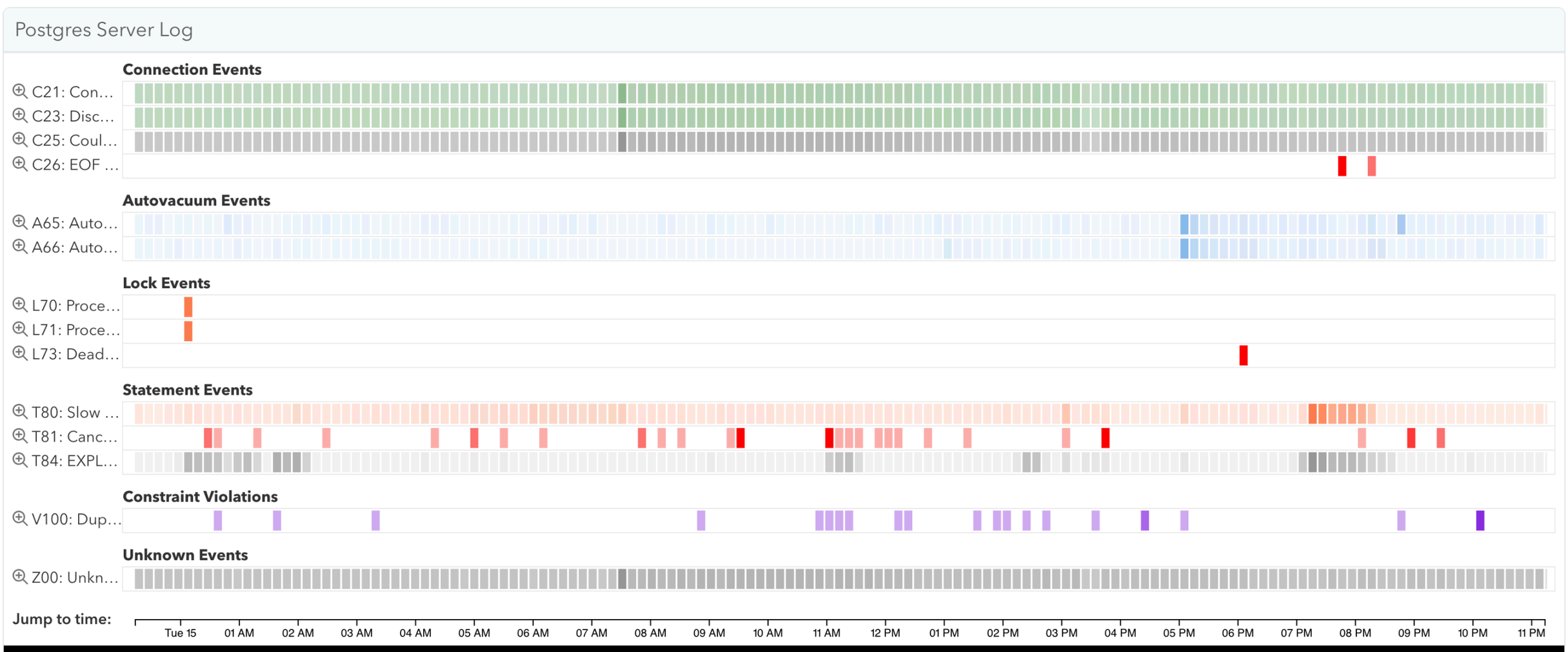
Estimated Improvement

	Current	Esti
Total autovacuum count	0	
Avoidable growth		
Rows	535,638	28(
Percentage	40.1%	2
Bytes	113.9 MB	59.

Insufficient VACUUM frequency typically results in table bloat
Table bloat can cause slow queries, increased disk space, memory usage, and increased I/O.
Learn more in documentation

Log Insights

- Tracks 100+ common Postgres log events with predefined filters
- Can filter out sensitive information from the logs and redact it
- Streams logs continuously from the database for immediate insights on deadlocks, timed out queries, checkpoints and more



HOT (Heap-Only Tuples) Analysis

- HOT = Optimization that makes UPDATEs in Postgres significantly cheaper by avoiding VACUUM overhead
- Not possible if any indexes are present on columns (except for BRIN)
- Considered for Index Advisor, and also analyzed for each column

Table: public.issue_references

Statistics | Partitions | Queries | Columns | Indexes | Index Advisor | Constraints | VACUUM/ANALYZE Activity | VACUUM Simulator

Row Statistics

Average Row Size	217 B	**Last Analyzed**	Apr 15 08:04:57pm PDT · 3 hours ago

Columns

NAME	TYPE	NULL %	AVG. SIZE	NUM. DISTINCT	UPDATES / MIN	INDEXED?	HOT?	MODIFIERS
id	uuid	0.00%	16	7,623,002*	-	Yes	-	NOT NULL DEFAULT public.gen_random_uu…
created_at	timestamp …	0.00%	8	36,530	-	No	-	NOT NULL DEFAULT now()
resolved_at	timestamp …	37.96%	8	22,166	0.45	Yes	No ⚠	
organization_id	uuid	0.00%	16	913	-	No	-	NOT NULL
server_id	uuid	0.00%	16	2,880	-	No	-	NOT NULL
issue_id	integer	0.00%	4	19,037	-	Yes	-	NOT NULL
referent_type	text	0.00%	8	9	-	Yes	-	NOT NULL
referent_id	text	0.00%	12	1,110,375*	-	Yes	-	NOT NULL
details	jsonb	0.00%	105	18,500	0.00	No	Yes ✓	NOT NULL DEFAULT '{}'::jsonb

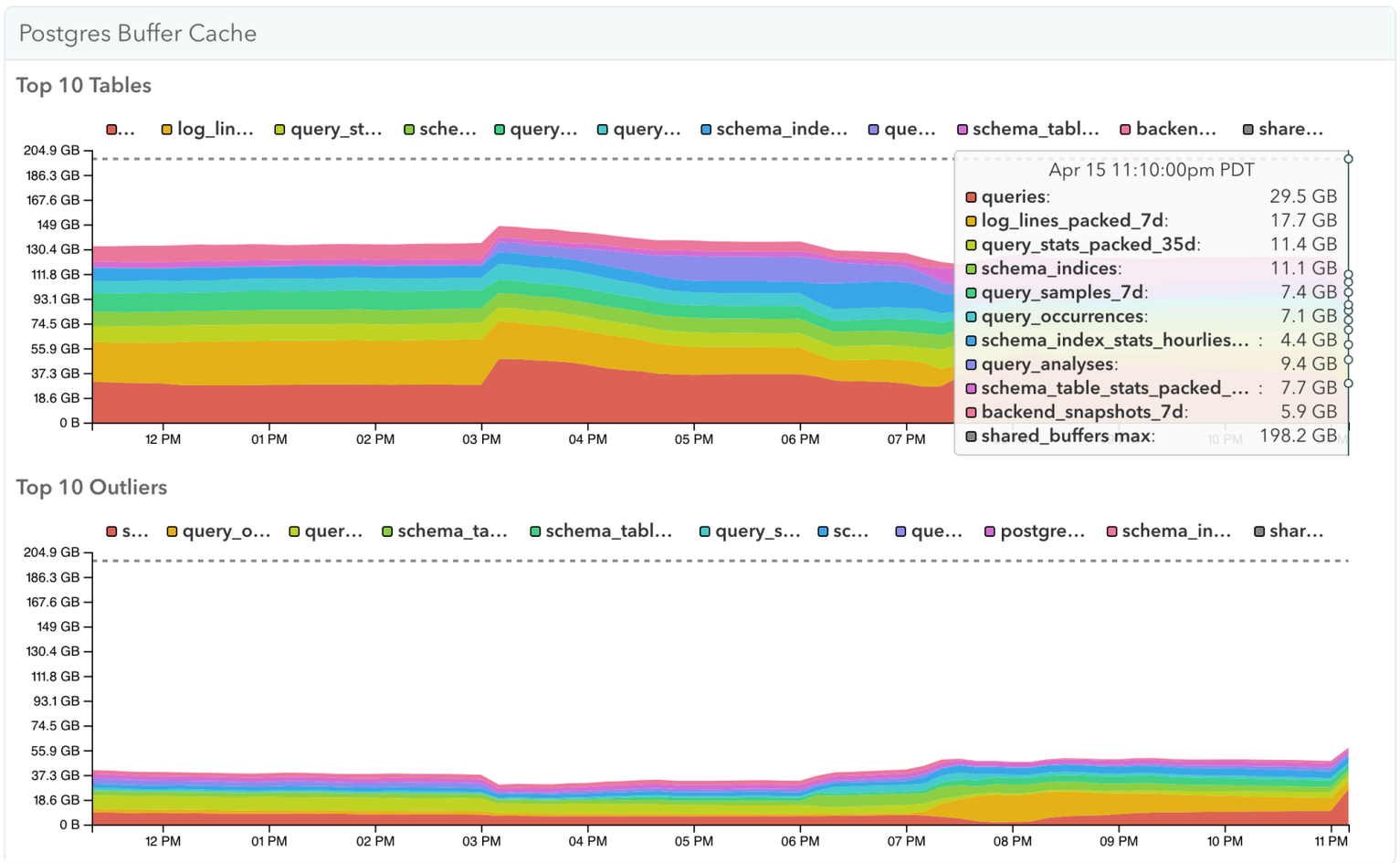
Lock Monitoring

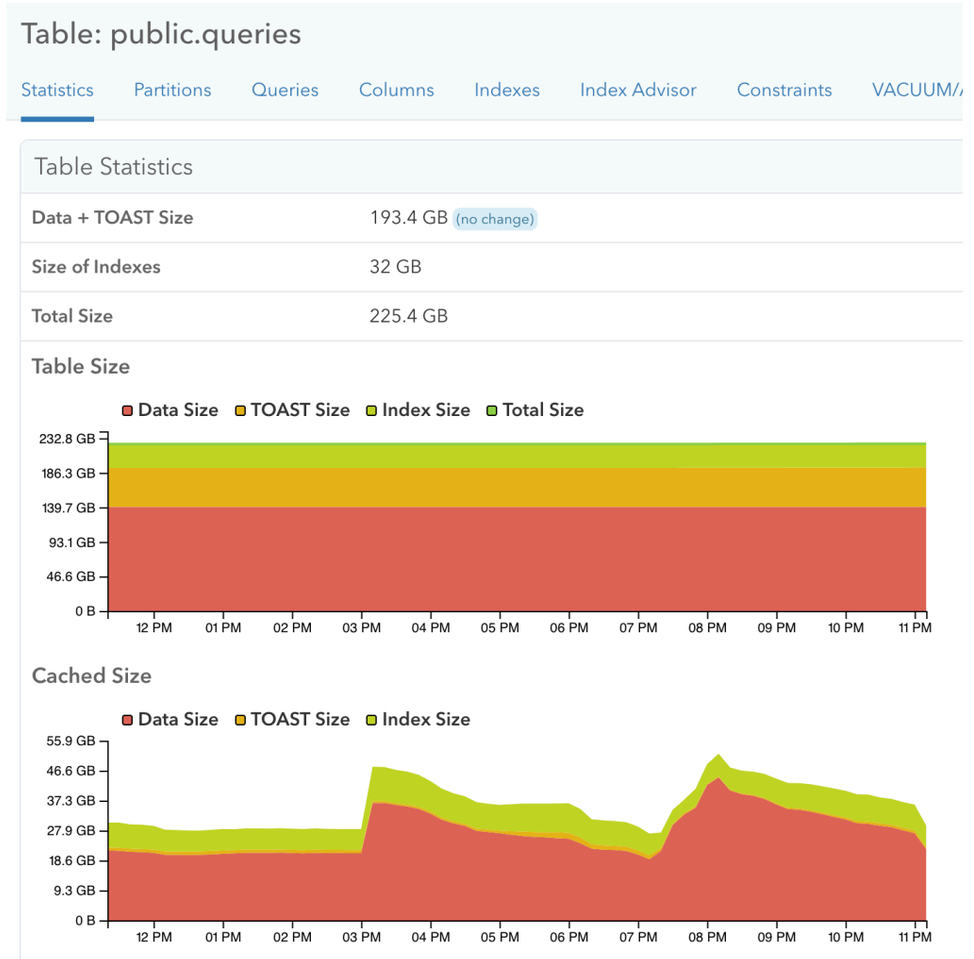
- Part of the Connection Tracing functionality ("Connections" page)
- Automatically analyzes lock graphs in Postgres to find the root query that is blocking activity
- Built-in alerting that notifies if a certain # of queries are blocked

19189	pgaweb_wor…	active (waiting)	<5 seconds	3 INSERT INTO backend_snapshots_7d (...) VALUES (...)	Lock / extend
8836	pgaweb_wor…	active	<5 seconds	SELECT query_analyses.* FROM query_analyses WHERE query_analyses.query_id = $1 LIMIT $2	Client / ClientRead
12488	pgaweb_wor…	active	<5 seconds	WITH data AS (...), existing_issues AS (...), stale_issues AS (...), insert_issues AS (...), upda...	Client / ClientRead
6708	pgaweb_wor…	active	<5 seconds	WITH child_tables AS (...), child_table_size AS (...) SELECT ... FROM unpack_schema_table_stats (d...	-
12561	pgaweb_wor…	active	<5 seconds	3 INSERT INTO backend_snapshots_7d (...) VALUES (...) Blocking 12359, 12489, 19189	IO / DataFileExtend
12489	pgaweb_wor…	active (waiting)	<5 seconds	2 INSERT INTO backend_snapshots_7d (...) VALUES (...)	Lock / extend
12359	**pgaweb_wo…**	**active (waiti…**	**<5 seconds**	1 INSERT INTO backend_snapshots_7d (...) VALUES (...)	**Lock / extend**

Buffer Cache Monitoring (new!)

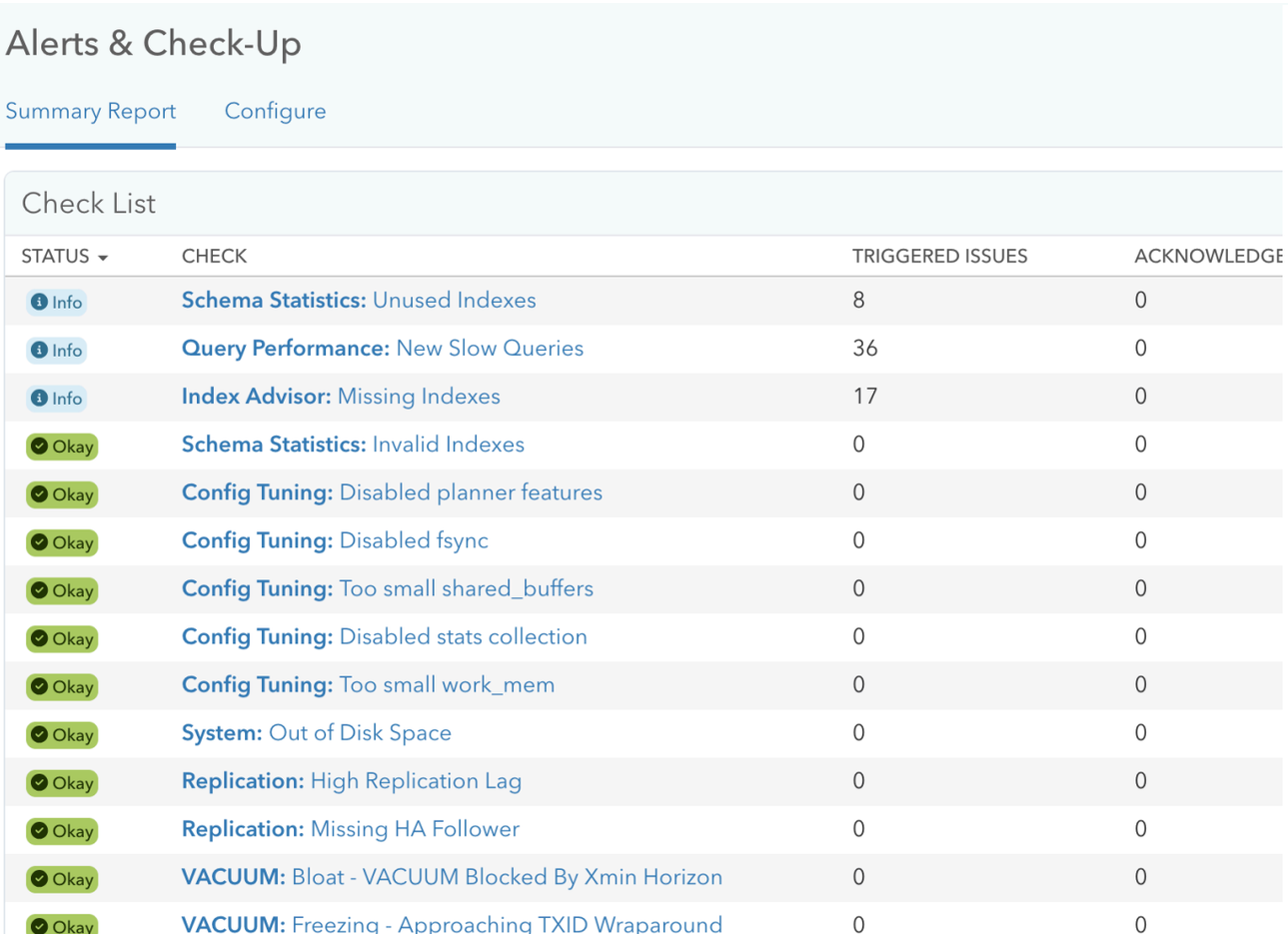
- Requires the pg_buffercache extension for Postgres
- Automatically analyzes cache contents over time to help identify performance regressions due to cache changes
- Shows breakdown of how much of a table and its indexes are cached

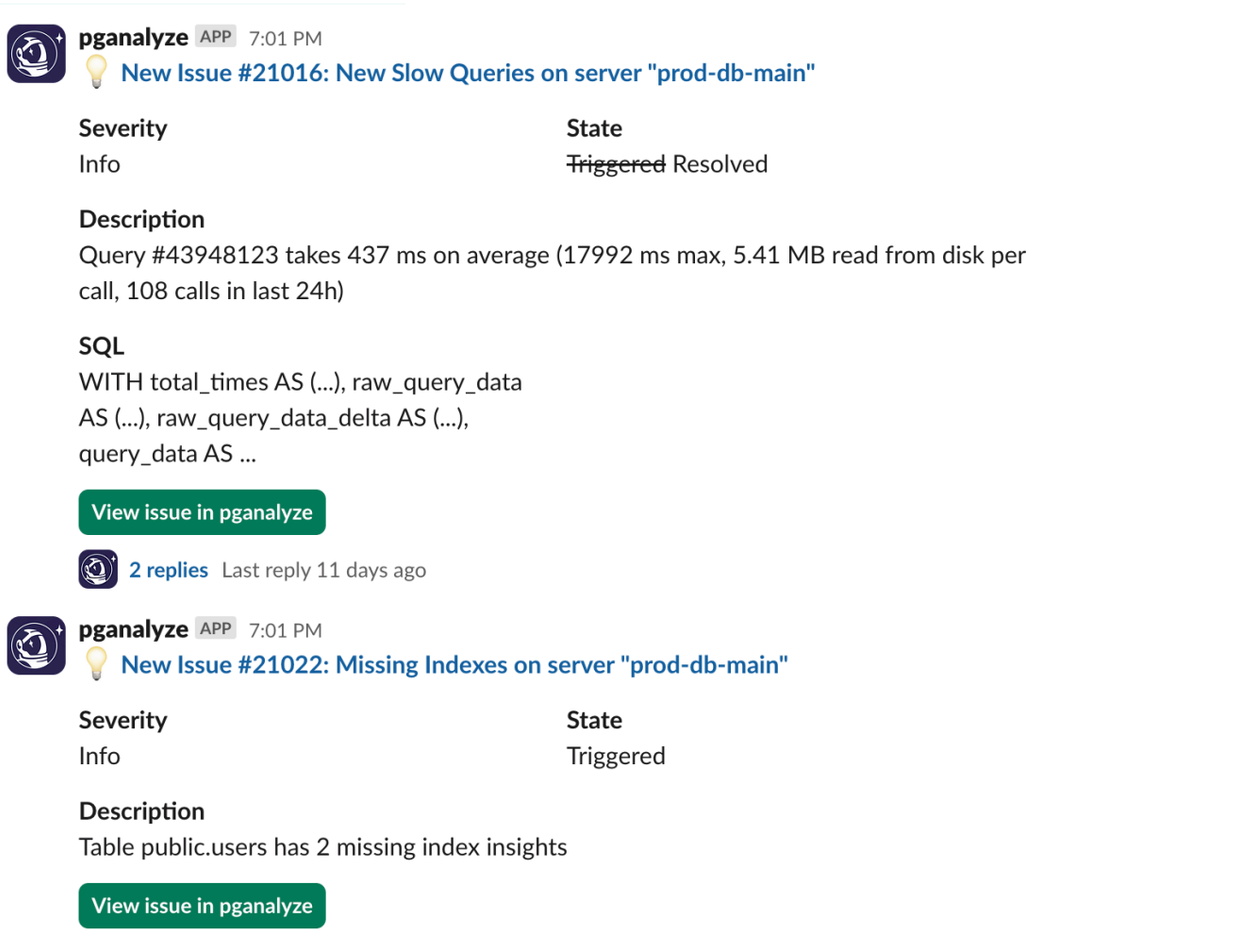




Alerting

- Built-in alerting for common Postgres problems
- Supports email notifications, Slack and PagerDuty
- API available for integrating with other systems
- Additional customization options coming later this year





pganalyze

Demo

Thank you!

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
