How to identify plan regressions using the new pg_stat_plans & **fix them with pganalyze**

pganalyze

lukas@pganalyze.com

1. **Query plans can change over time**
2. **How to capture plan statistics**
3. **The new pg_stat_plans extension**
4. **Behind the scenes: Low-overhead Plan IDs**
5. **Fixing bad query plans with pganalyze Query Advisor**

pganalyze

pganalyze

Query plans can change over time

Let's start with a simple query:

```
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $1
  AND databases.hidden = $2
ORDER BY databases.id ASC
LIMIT $3
```

Sometimes we get a good plan:

```
                                                                                 QUERY PLAN
----------------------------------------------------------------------------------------------------------
 Limit  (cost=137.54..137.55 rows=4 width=152) (actual time=0.029..0.029 rows=2 loops=1)
   ->  Sort  (cost=137.54..137.55 rows=4 width=152) (actual time=0.028..0.028 rows=2 loops=1)
         Sort Key: id
         Sort Method: quicksort  Memory: 25kB
         ->  Index Scan using index_databases_on_server_id_and_datname on databases  (cost=0.56..137.50
               Index Cond: (server_id = 'XXX'::uuid)
               Filter: (NOT hidden)
               Rows Removed by Filter: 3
 Planning Time: 0.096 ms
 Execution Time: 0.046 ms
```

And sometimes we get a bad plan:

```
                                                                          QUERY PLAN
--------------------------------------------------------------------------------------------------------
 Limit  (cost=1000.58..18048.33 rows=1000 width=152) (actual time=537.544..539.169 rows=1 loops=1)
   ->  Gather Merge  (cost=1000.58..162851.91 rows=9494 width=152) (actual time=537.543..539.167 rows=1
         Workers Planned: 2
         Workers Launched: 2
         ->  Parallel Index Scan using databases_pkey on databases  (cost=0.56..160756.04 rows=3956 wid
               Filter: ((NOT hidden) AND (server_id = 'YYY'::uuid))
               Rows Removed by Filter: 982622
 Planning Time: 0.088 ms
 Execution Time: 539.213 ms
```

Reasons for bad plans suddenly appearing:

- Different input values change selectivity
- ANALYZE changed the table statistics
- Table data changed
- Indexes changed
- Postgres version upgrades (rare, but it happens!)

We can't run (and look at) EXPLAIN on every single query.

Plan Statistics are about capturing what happens over time, so we can proactively identify bad plans.

pganalyze

How to capture plan statistics

Query ID

Differentiates by query structure.

Plan ID

Differentiates by plan shape.

Plan Shape

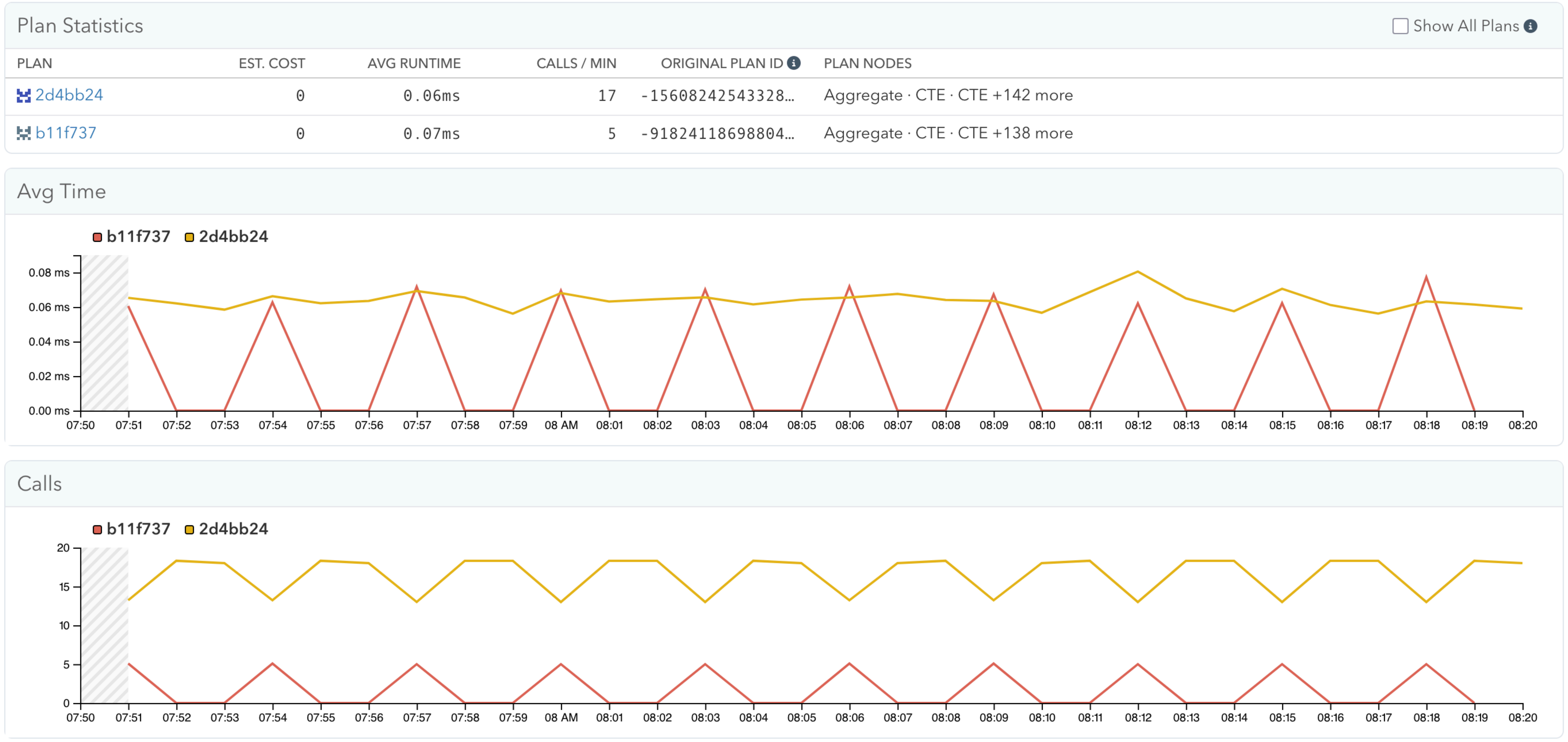
~ EXPLAIN (COSTS OFF)

```
Seq Scan on users
   Filter: (lower((email)::text) = '...'::text)
```

vs

```
 Bitmap Heap Scan on users
   Recheck Cond: (lower((email)::text) = '...'::text)
   ->  Bitmap Index Scan on index_users_lower_email
         Index Cond: (lower((email)::text) = '...'::text)
```

Plan IDs let us track plan usage over time



Plan Statistics

☐ Show All Plans

PLAN	EST. COST	AVG RUNTIME	CALLS / MIN	ORIGINAL PLAN ID	PLAN NODES
2d4bb24	0	0.06ms	17	-15608242543328…	Aggregate · CTE · CTE +142 more
b11f737	0	0.07ms	5	-91824118698804…	Aggregate · CTE · CTE +138 more

Avg Time

🟥 b11f737
🟨 2d4bb24
0.08 ms
0.06 ms
0.04 ms
0.02 ms
0.00 ms
07:50 07:51 07:52 07:53 07:54 07:55 07:56 07:57 07:58 07:59 08 AM 08:01 08:02 08:03 08:04 08:05 08:06 08:07 08:08 08:09 08:10 08:11 08:12 08:13 08:14 08:15 08:16 08:17 08:18 08:19 08:20

Calls

🟥 b11f737
🟨 2d4bb24
20
15
10
5
0
07:50 07:51 07:52 07:53 07:54 07:55 07:56 07:57 07:58 07:59 08 AM 08:01 08:02 08:03 08:04 08:05 08:06 08:07 08:08 08:09 08:10 08:11 08:12 08:13 08:14 08:15 08:16 08:17 08:18 08:19 08:20

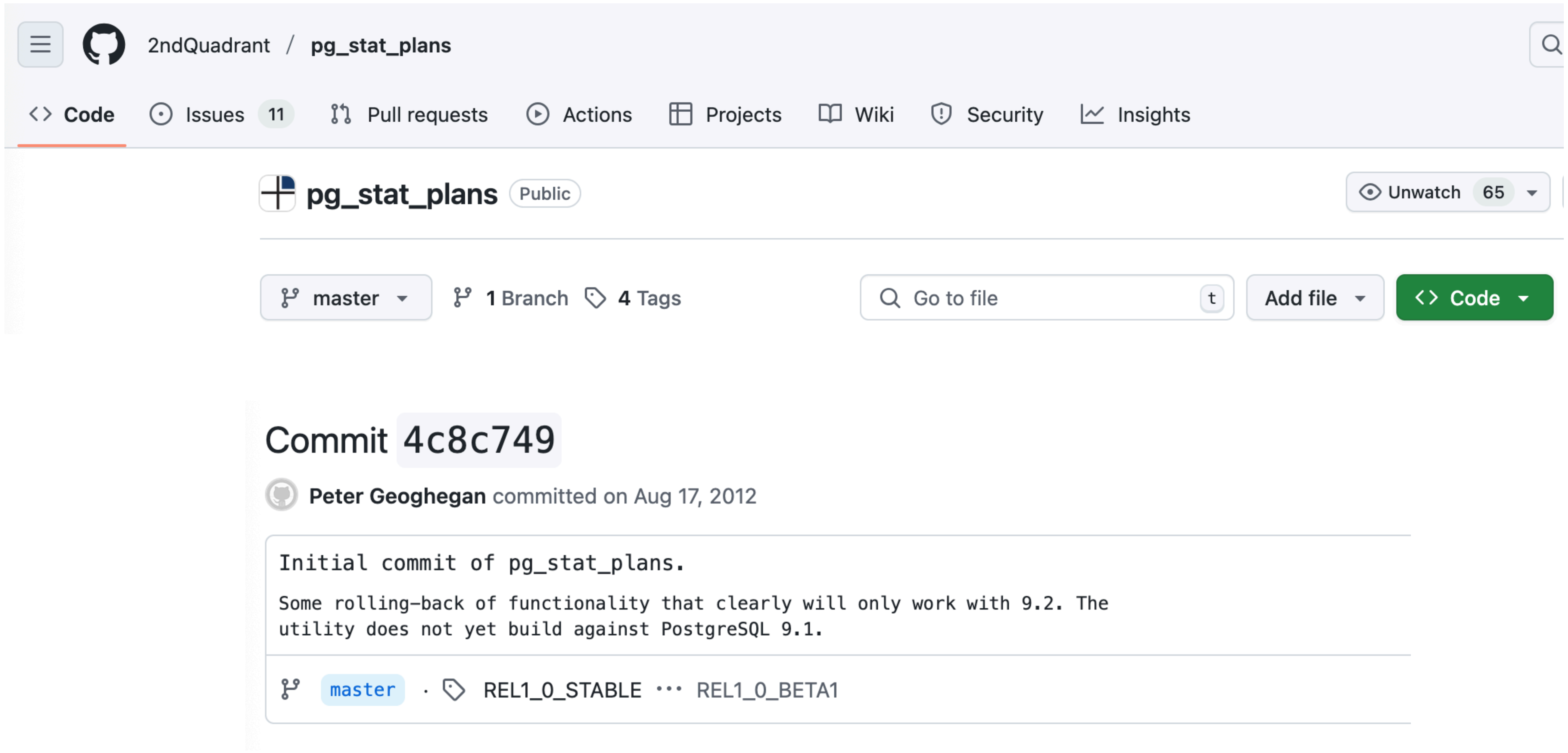
Plan IDs let us detect regressions, quickly

"I’m a huge fan of Postgres. This one is “user error”, but we still got bit pretty hard.

A query plan changed, on a frequently-run query (~1k/sec) on a large table (~2B rows) without warning. Went from sub-millisecond to multi-second.

The PG query planner is generally very good, but also very opaque."
- Scott Hardy on Hacker News (2021)

This is not a new idea.



Commit 4c8c749

Peter Geoghegan committed on Aug 17, 2012

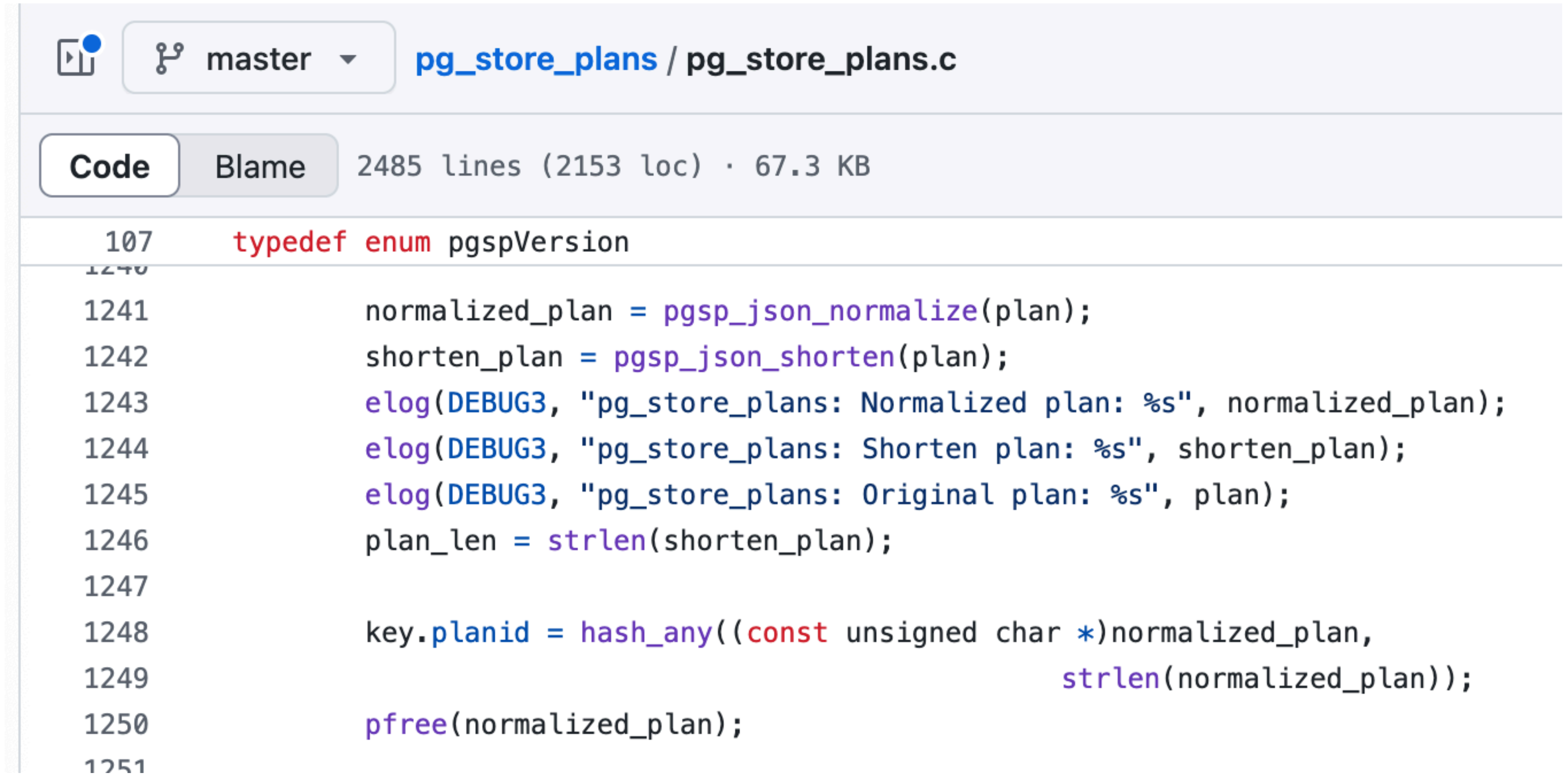
```
Initial commit of pg_stat_plans.

Some rolling-back of functionality that clearly will only work with 9.2. The
utility does not yet build against PostgreSQL 9.1.
```

master · REL1_0_STABLE ··· REL1_0_BETA1

The old pg_stat_plans is unmaintained.

There are open-source alternatives, but they have high overhead.

pg_store_plans

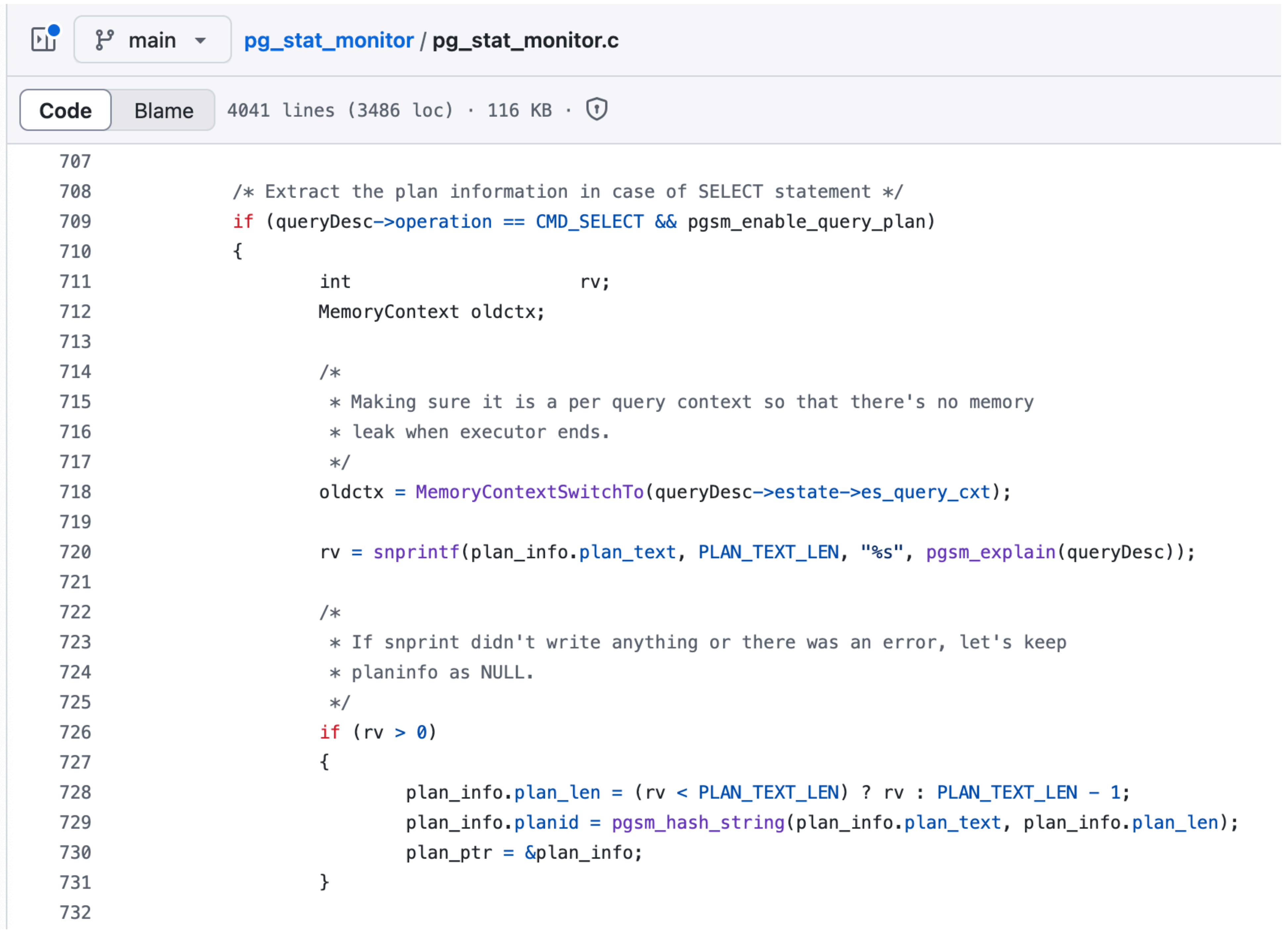

master
pg_store_plans / pg_store_plans.c
Code
Blame
2485 lines (2153 loc) · 67.3 KB

```
typedef enum pgspVersion
		normalized_plan = pgsp_json_normalize(plan);
		shorten_plan = pgsp_json_shorten(plan);
		elog(DEBUG3, "pg_store_plans: Normalized plan: %s", normalized_plan);
		elog(DEBUG3, "pg_store_plans: Shorten plan: %s", shorten_plan);
		elog(DEBUG3, "pg_store_plans: Original plan: %s", plan);
		plan_len = strlen(shorten_plan);

		key.planid = hash_any((const unsigned char *)normalized_plan,
											   strlen(normalized_plan));
		pfree(normalized_plan);
```

Calculates the EXPLAIN text for every execution to hash it for the plan ID

~20% overhead in some cases

pg_stat_monitor


main
pg_stat_monitor / pg_stat_monitor.c
Code
Blame
4041 lines (3486 loc) · 116 KB

```c

        /* Extract the plan information in case of SELECT statement */
        if (queryDesc->operation == CMD_SELECT && pgsm_enable_query_plan)
        {
                int                     rv;
                MemoryContext oldctx;

                /*
                 * Making sure it is a per query context so that there's no memory
                 * leak when executor ends.
                 */
                oldctx = MemoryContextSwitchTo(queryDesc->estate->es_query_cxt);

                rv = snprintf(plan_info.plan_text, PLAN_TEXT_LEN, "%s", pgsm_explain(queryDesc));

                /*
                 * If snprint didn't write anything or there was an error, let's keep
                 * planinfo as NULL.
                 */
                if (rv > 0)
                {
                        plan_info.plan_len = (rv < PLAN_TEXT_LEN) ? rv : PLAN_TEXT_LEN - 1;
                        plan_info.planid = pgsm_hash_string(plan_info.plan_text, plan_info.plan_len);
                        plan_ptr = &plan_info;
                }

```

Calculates the EXPLAIN text for every execution to hash it for the plan ID (if enabled)

In 2024, AWS launched
aurora_plan_stats for Aurora.

And Microsoft has plan IDs
in **Query Store for Azure Postgres.**

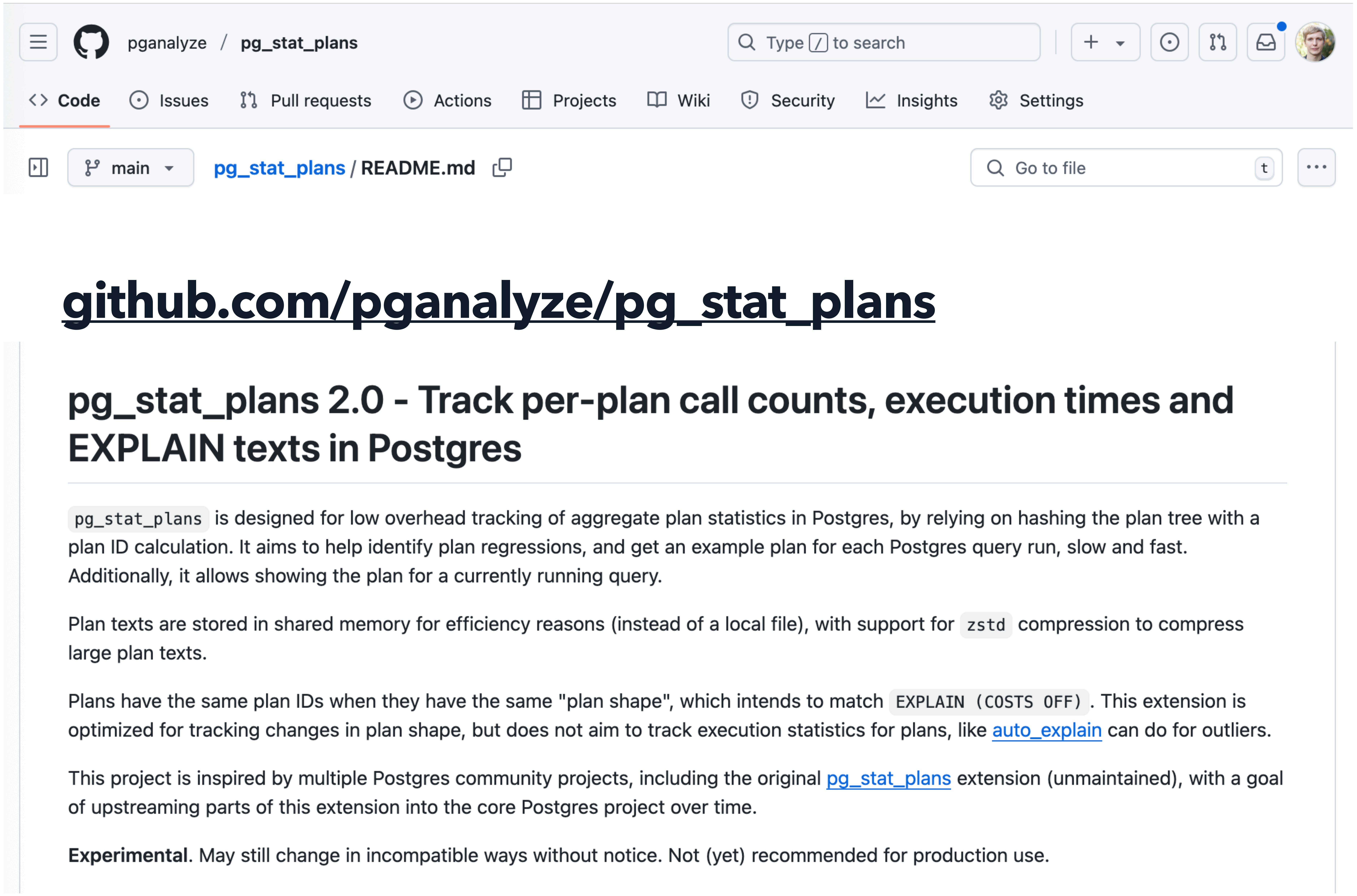
Can Postgres do better here?

pganalyze

A new pg_stat_plans

pganalyze

A new pg_stat_plans



github.com/pganalyze/pg_stat_plans

pg_stat_plans 2.0 - Track per-plan call counts, execution times and EXPLAIN texts in Postgres

`pg_stat_plans` is designed for low overhead tracking of aggregate plan statistics in Postgres, by relying on hashing the plan tree with a plan ID calculation. It aims to help identify plan regressions, and get an example plan for each Postgres query run, slow and fast. Additionally, it allows showing the plan for a currently running query.

Plan texts are stored in shared memory for efficiency reasons (instead of a local file), with support for `zstd` compression to compress large plan texts.

Plans have the same plan IDs when they have the same "plan shape", which intends to match `EXPLAIN (COSTS OFF)`. This extension is optimized for tracking changes in plan shape, but does not aim to track execution statistics for plans, like auto_explain can do for outliers.

This project is inspired by multiple Postgres community projects, including the original pg_stat_plans extension (unmaintained), with a goal of upstreaming parts of this extension into the core Postgres project over time.

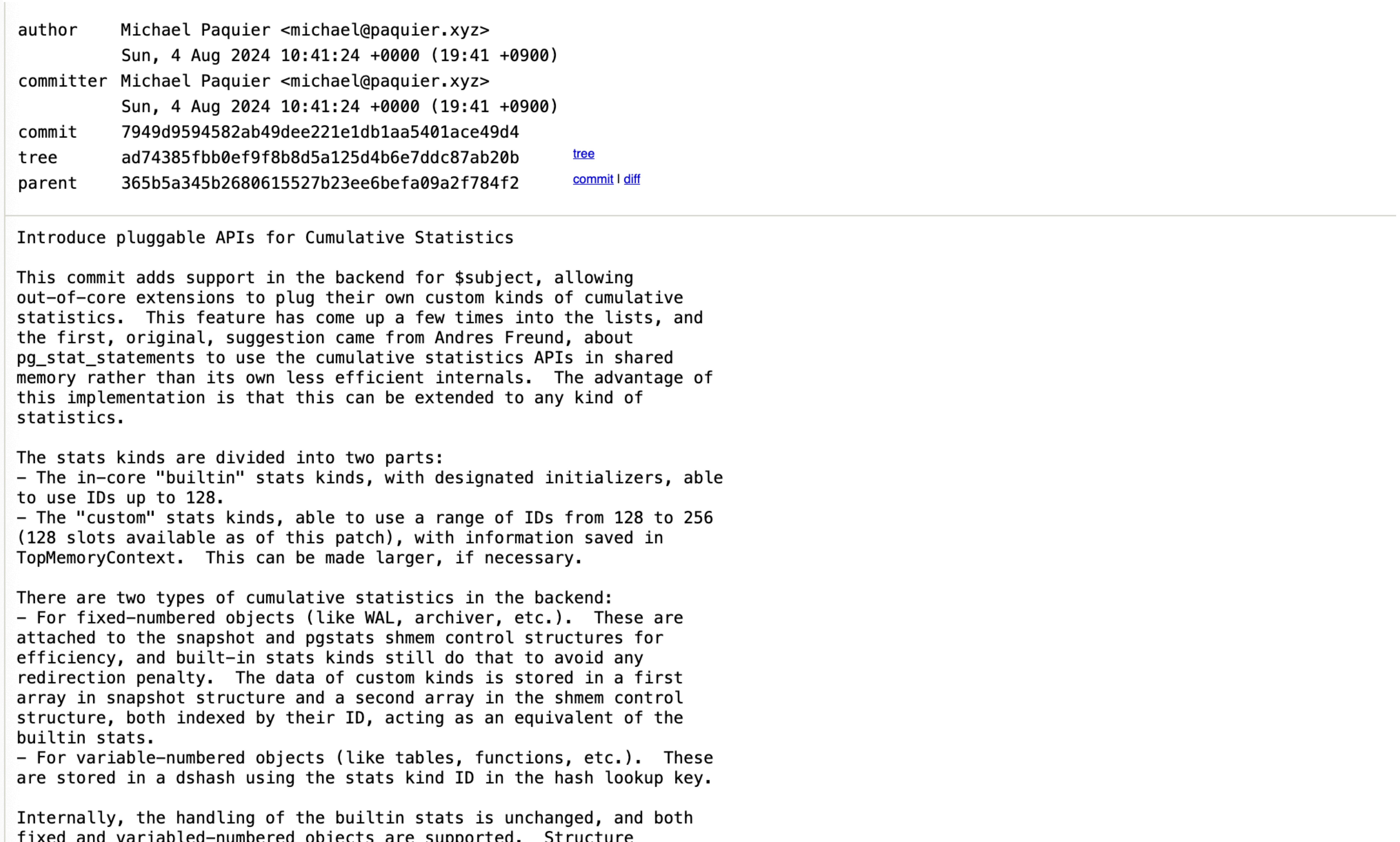
Experimental. May still change in incompatible ways without notice. Not (yet) recommended for production use.

```
SELECT * FROM pg_stat_plans;
```

```
-[ RECORD 1 ]---+------------------------------------------------------------------------------------------
userid          | 10
dbid            | 16391
toplevel        | t
queryid         | -2322344003805516737
planid          | -1865871893278385236
calls           | 1
total_exec_time | 0.047708
plan            | Limit
                |   ->  Sort
                |         Sort Key: database_stats_35d.frozenxid_age DESC
                |         ->  Bitmap Heap Scan on database_stats_35d_20250514 database_stats_35d
                |               Recheck Cond: (server_id = '00000000-0000-0000-0000-000000000000'::uuid)
                                                                                  d_at = '2025-05-14 14:30:0
                                                                                  14_server_id_idx
                                                                                  -0000-000000000000'::uuid)
```

Cumulative statistics on **which query ID used which plan, how often (calls)**, and **how long it took (total_exec_time).**

PG18: Introduce pluggable APIs for Cumulative Statistics



```
author    Michael Paquier <michael@paquier.xyz>
          Sun, 4 Aug 2024 10:41:24 +0000 (19:41 +0900)
committer Michael Paquier <michael@paquier.xyz>
          Sun, 4 Aug 2024 10:41:24 +0000 (19:41 +0900)
commit    7949d9594582ab49dee221e1db1aa5401ace49d4
tree      ad74385fbb0ef9f8b8d5a125d4b6e7ddc87ab20b     tree
parent    365b5a345b2680615527b23ee6befa09a2f784f2     commit | diff
```

```
Introduce pluggable APIs for Cumulative Statistics

This commit adds support in the backend for $subject, allowing
out-of-core extensions to plug their own custom kinds of cumulative
statistics.  This feature has come up a few times into the lists, and
the first, original, suggestion came from Andres Freund, about
pg_stat_statements to use the cumulative statistics APIs in shared
memory rather than its own less efficient internals.  The advantage of
this implementation is that this can be extended to any kind of
statistics.

The stats kinds are divided into two parts:
- The in-core "builtin" stats kinds, with designated initializers, able
to use IDs up to 128.
- The "custom" stats kinds, able to use a range of IDs from 128 to 256
(128 slots available as of this patch), with information saved in
TopMemoryContext.  This can be made larger, if necessary.

There are two types of cumulative statistics in the backend:
- For fixed-numbered objects (like WAL, archiver, etc.).  These are
attached to the snapshot and pgstats shmem control structures for
efficiency, and built-in stats kinds still do that to avoid any
redirection penalty.  The data of custom kinds is stored in a first
array in snapshot structure and a second array in the shmem control
structure, both indexed by their ID, acting as an equivalent of the
builtin stats.
- For variable-numbered objects (like tables, functions, etc.).  These
are stored in a dshash using the stats kind ID in the hash lookup key.

Internally, the handling of the builtin stats is unchanged, and both
fixed and variabled-numbered objects are supported.  Structure
```

```
SELECT * FROM pg_stat_plans;


-[ RECORD 1 ]---+------------------------------------------------------------------------
userid          | 10
dbid            | 16391
toplevel        | t
queryid         | -2322344003805516737
planid          | -1865871893278385236
calls           | 1
total_exec_time | 0.047708
plan            | Limit
                |   ->  Sort
                |         Sort Key: database_stats_35d.frozenxid_age DESC
                |         ->  Bitmap Heap Scan on database_stats_35d_20250514 database_stats_35d
```

Plan ID calculated with tree walk after planning
+ copying code from Postgres

```
SELECT * FROM pg_stat_plans;
```

Plan Text stored in **Dynamic Shared Memory,** not a file on disk. Optionally compressed with zstd.

```
plan               | Limit
                   |   ->  Sort
                   |         Sort Key: database_stats_35d.frozenxid_age DESC
                   |         ->  Bitmap Heap Scan on database_stats_35d_20250514 database_stats_35d
                   |               Recheck Cond: (server_id = '00000000-0000-0000-0000-000000000000'::uuid
                   |               Filter: ((frozenxid_age IS NOT NULL) AND (collected_at = '2025-05-14 14
                   |               ->  Bitmap Index Scan on database_stats_35d_20250514_server_id_idx
                   |                     Index Cond: (server_id = '00000000-0000-0000-0000-000000000000'::
```

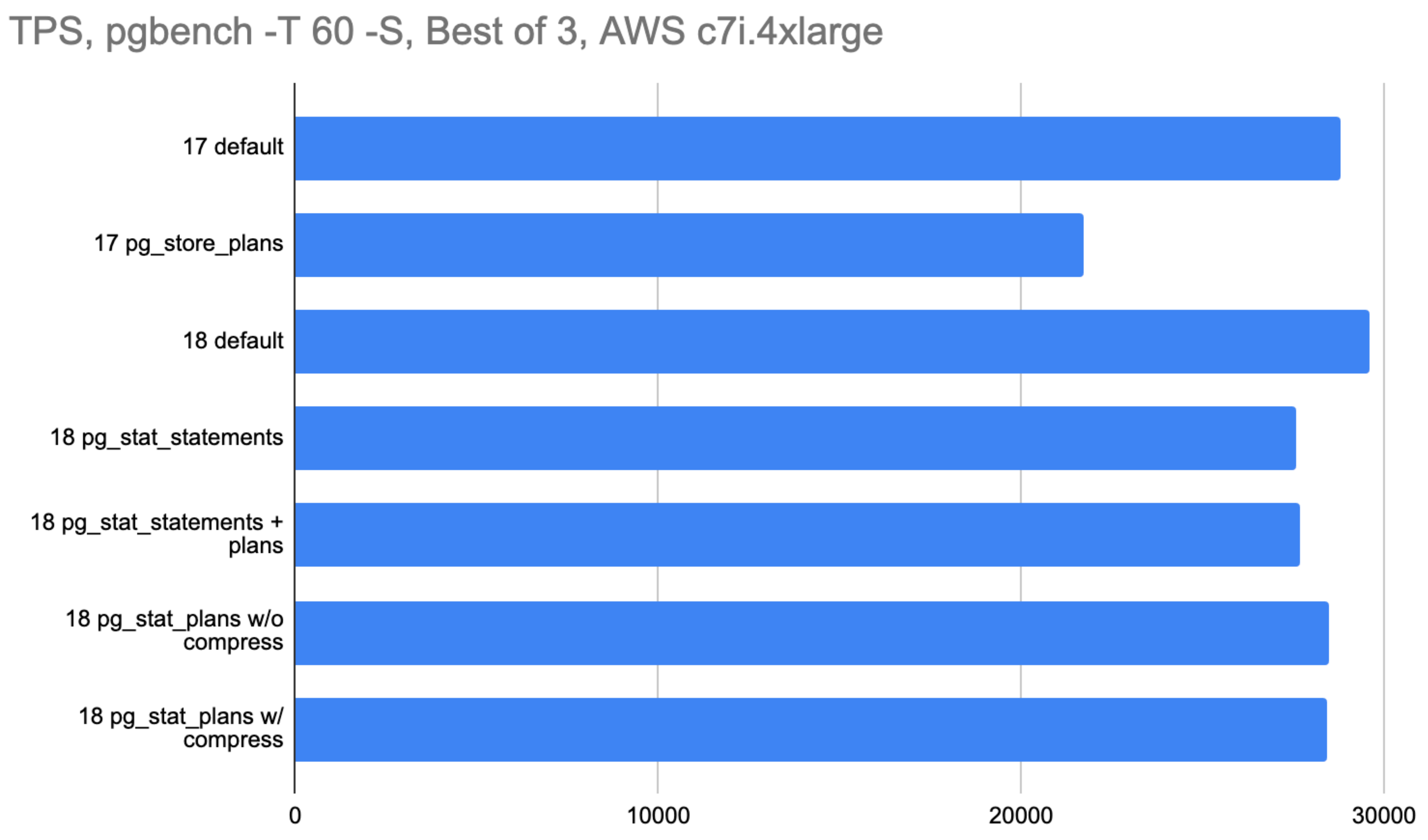
```
SELECT * FROM pg_stat_plans_activity;
```

```
  pid  |        plan_id        |                                         plan
-------+-----------------------+--------------------------------------------------------------------------------------------
 83994 | -5449095327982245076 | Merge Join
       |                       |   Merge Cond: ((a.datid = p.dbid) AND (a.usesysid = p.userid) AND (a.query_id = p.query
       |                       |   ->  Sort
       |                       |         Sort Key: a.datid, a.usesysid, a.query_id, a.plan_id
       |                       |         ->  Function Scan on pg_stat_plans_get_activity a
       |                       |   ->  Sort
       |                       |         Sort Key: p.dbid, p.userid, p.queryid, p.planid
       |                       |         ->  Function Scan on pg_stat_plans p
       |                       |               Filter: (toplevel IS TRUE)
 87168 |  4721228144609632390 | Sort
       |                       |   Sort Key: q.id
       |                       |   ->  Nested Loop
       |                       |         ->  Index Scan using index_query_runs_on_server_id on query_runs q
       |                       |               Index Cond: (server_id = '00000000-0000-0000-0000-000000000000'::uuid)
```

Get the plan for a currently running query

(no progress tracking, just the plan that's being used)

Overhead is noticeably lower than existing extensions (higher is better)



Next steps for pg_stat_plans 2.0

- Plan text compression improvements
- Stabilize extension (test/benchmark)
- Include in Postgres repositories
- Get cloud providers to adopt pg_stat_plans

Open questions

- How do we handle table partitioning (Append node) in plan IDs?
- What metrics should we capture per-plan?
- Worth supporting non-text EXPLAIN output?
- Should we normalize plan texts? (remove constants)

pganalyze

Behind the scenes:
Low-overhead Plan IDs

Plan ID calculation must be fast

It should happen with every planning cycle.

ExplainPrintPlan + hash(big text)

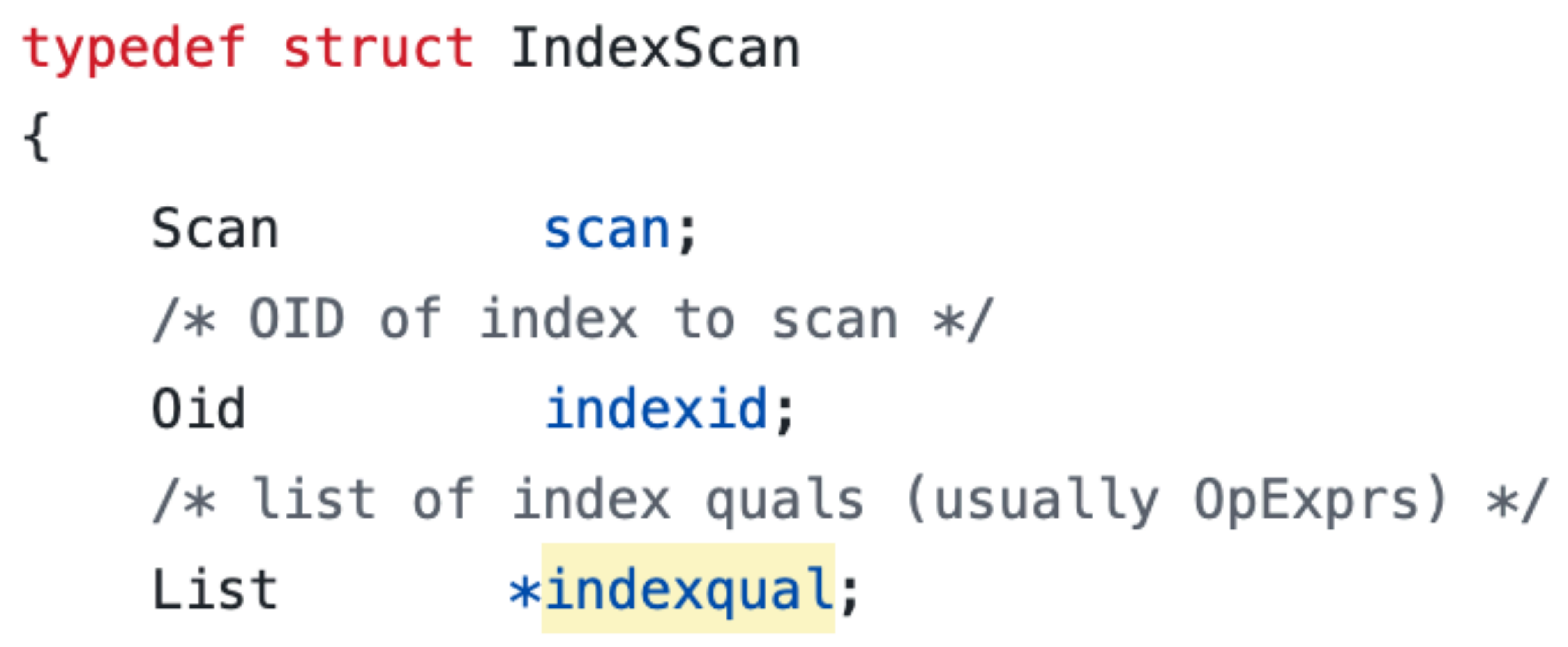
~~ExplainPrintPlan + hash(big text)~~

We need a tree walk + "jumble"

Query ID = Walk post parse-analysis trees

Plan ID = Walk plan tree

This is not trivial out-of-core.



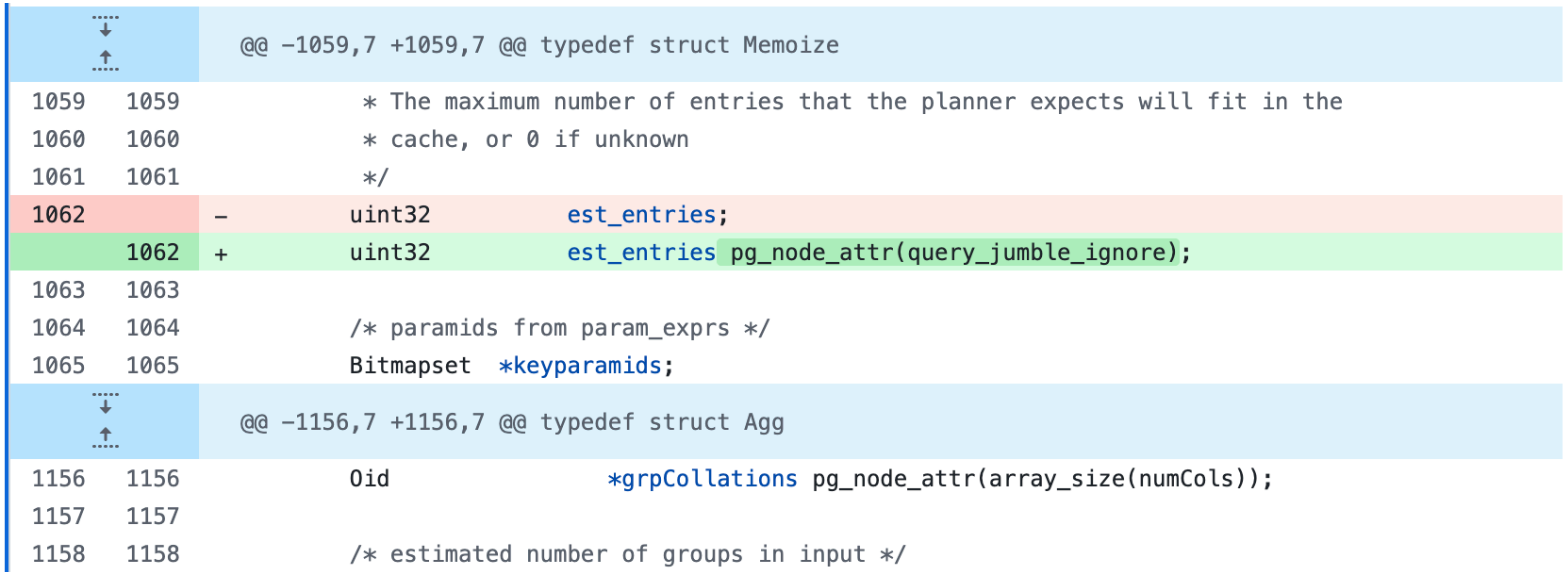
```
typedef struct IndexScan
{
    Scan        scan;
    /* OID of index to scan */
    Oid         indexid;
    /* list of index quals (usually OpExprs) */
    List       *indexqual;
```

e.g. Index Quals are "Usually" OpExpr

(but could be any node, and we want to make a hash of it)

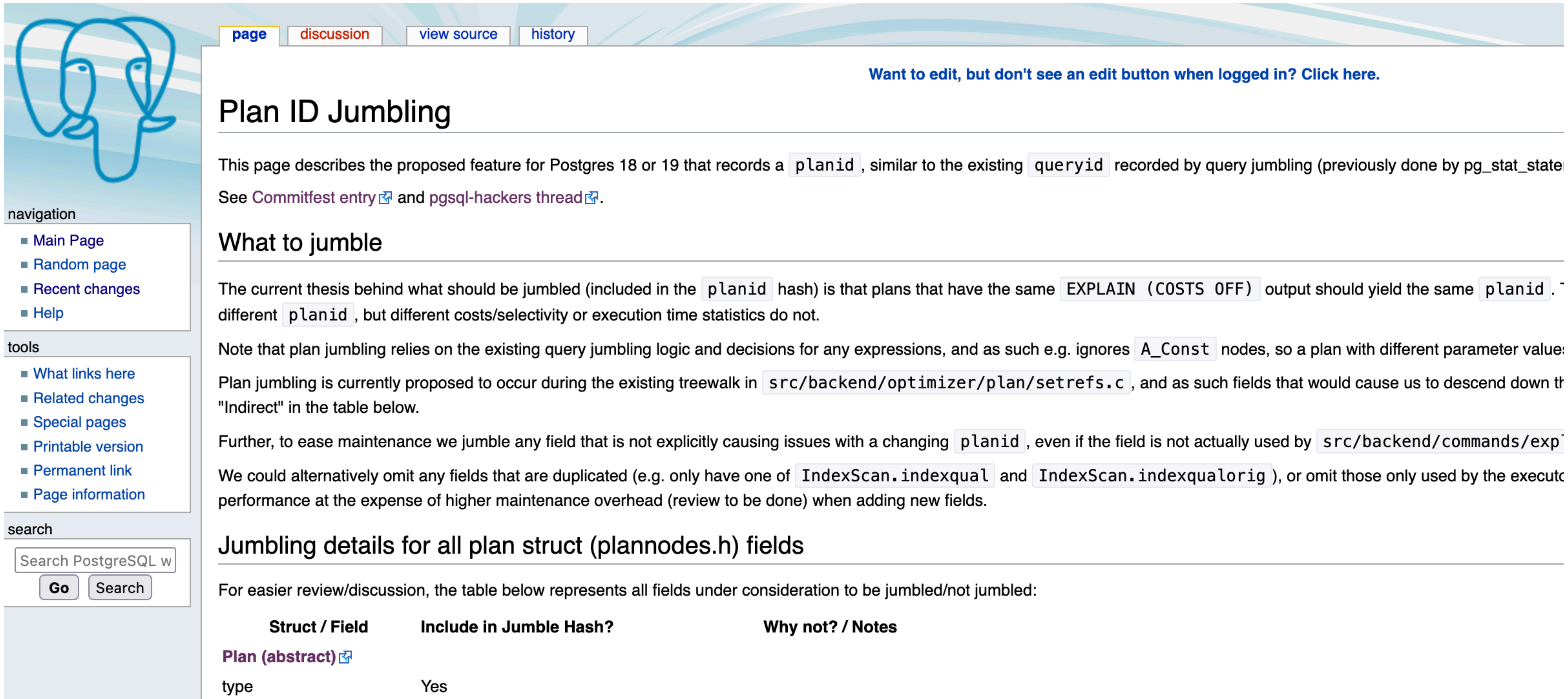
In core its easy to maintain "what is significant"

on the plannodes.h structs



```
@@ -1059,7 +1059,7 @@ typedef struct Memoize
         * The maximum number of entries that the planner expects will fit in the
         * cache, or 0 if unknown
         */
-       uint32          est_entries;
+       uint32          est_entries pg_node_attr(query_jumble_ignore);

        /* paramids from param_exprs */
        Bitmapset   *keyparamids;
@@ -1156,7 +1156,7 @@ typedef struct Agg
        Oid                 *grpCollations pg_node_attr(array_size(numCols));

        /* estimated number of groups in input */
```

In Postgres 18, started effort to define what a "Plan ID" could be in upstream Postgres



Want to edit, but don't see an edit button when logged in? Click here.

Plan ID Jumbling

This page describes the proposed feature for Postgres 18 or 19 that records a `planid`, similar to the existing `queryid` recorded by query jumbling (previously done by pg_stat_state

See Commitfest entry and pgsql-hackers thread.

What to jumble

The current thesis behind what should be jumbled (included in the `planid` hash) is that plans that have the same `EXPLAIN (COSTS OFF)` output should yield the same `planid`. different `planid`, but different costs/selectivity or execution time statistics do not.

Note that plan jumbling relies on the existing query jumbling logic and decisions for any expressions, and as such e.g. ignores `A_Const` nodes, so a plan with different parameter value

Plan jumbling is currently proposed to occur during the existing treewalk in `src/backend/optimizer/plan/setrefs.c`, and as such fields that would cause us to descend down th "Indirect" in the table below.

Further, to ease maintenance we jumble any field that is not explicitly causing issues with a changing `planid`, even if the field is not actually used by `src/backend/commands/exp`

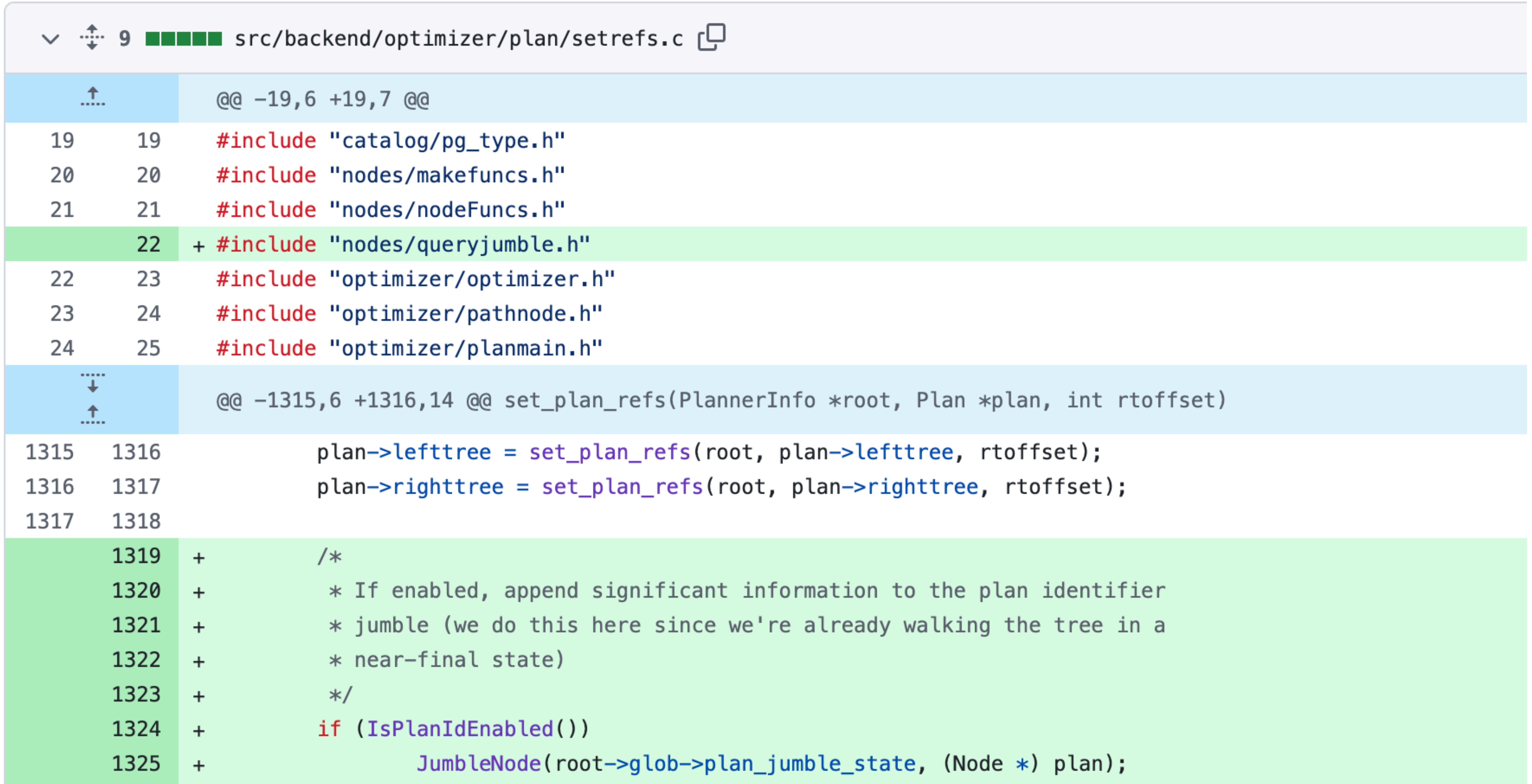
We could alternatively omit any fields that are duplicated (e.g. only have one of `IndexScan.indexqual` and `IndexScan.indexqualorig`), or omit those only used by the executo performance at the expense of higher maintenance overhead (review to be done) when adding new fields.

Jumbling details for all plan struct (plannodes.h) fields

For easier review/discussion, the table below represents all fields under consideration to be jumbled/not jumbled:

Struct / Field	Include in Jumble Hash?	Why not? / Notes
Plan (abstract)		
type	Yes	

In core we also have a tree walk we could re-use, in setrefs.c

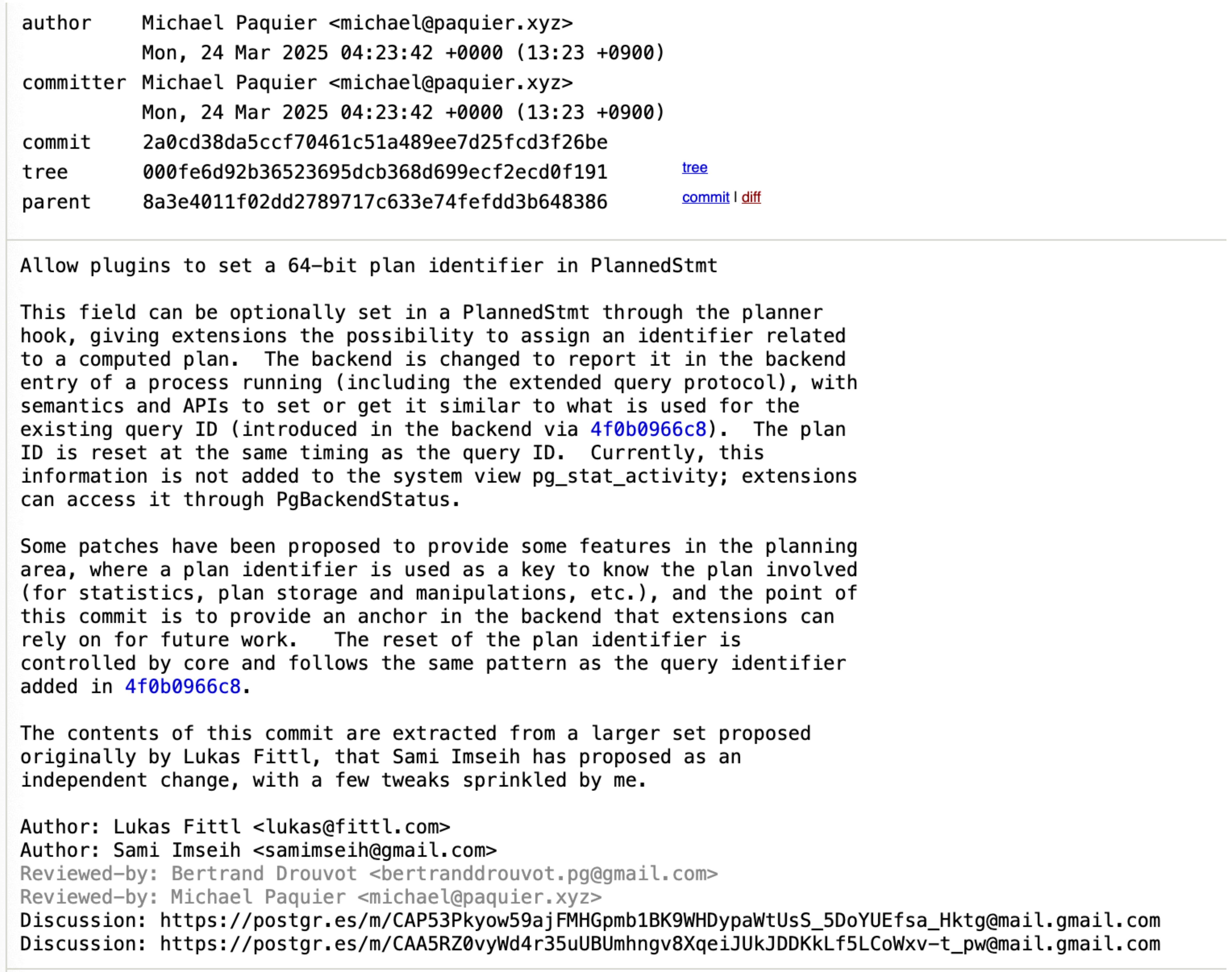


```
9 src/backend/optimizer/plan/setrefs.c

@@ -19,6 +19,7 @@
  #include "catalog/pg_type.h"
  #include "nodes/makefuncs.h"
  #include "nodes/nodeFuncs.h"
+ #include "nodes/queryjumble.h"
  #include "optimizer/optimizer.h"
  #include "optimizer/pathnode.h"
  #include "optimizer/planmain.h"
@@ -1315,6 +1316,14 @@ set_plan_refs(PlannerInfo *root, Plan *plan, int rtoffset)
          plan->lefttree = set_plan_refs(root, plan->lefttree, rtoffset);
          plan->righttree = set_plan_refs(root, plan->righttree, rtoffset);

+         /*
+          * If enabled, append significant information to the plan identifier
+          * jumble (we do this here since we're already walking the tree in a
+          * near-final state)
+          */
+         if (IsPlanIdEnabled())
+                 JumbleNode(root->glob->plan_jumble_state, (Node *) plan);
```

Most of this got pushed to PG19+.

But we did get a
key improvement in 18
we can build on.

PG18: Allow plugins to set a 64-bit plan identifier in PlannedStmt



```
author    Michael Paquier <michael@paquier.xyz>
          Mon, 24 Mar 2025 04:23:42 +0000 (13:23 +0900)
committer Michael Paquier <michael@paquier.xyz>
          Mon, 24 Mar 2025 04:23:42 +0000 (13:23 +0900)
commit    2a0cd38da5ccf70461c51a489ee7d25fcd3f26be
tree      000fe6d92b36523695dcb368d699ecf2ecd0f191    tree
parent    8a3e4011f02dd2789717c633e74fefdd3b648386    commit | diff
```

```
Allow plugins to set a 64-bit plan identifier in PlannedStmt

This field can be optionally set in a PlannedStmt through the planner
hook, giving extensions the possibility to assign an identifier related
to a computed plan.  The backend is changed to report it in the backend
entry of a process running (including the extended query protocol), with
semantics and APIs to set or get it similar to what is used for the
existing query ID (introduced in the backend via 4f0b0966c8).  The plan
ID is reset at the same timing as the query ID.  Currently, this
information is not added to the system view pg_stat_activity; extensions
can access it through PgBackendStatus.

Some patches have been proposed to provide some features in the planning
area, where a plan identifier is used as a key to know the plan involved
(for statistics, plan storage and manipulations, etc.), and the point of
this commit is to provide an anchor in the backend that extensions can
rely on for future work.   The reset of the plan identifier is
controlled by core and follows the same pattern as the query identifier
added in 4f0b0966c8.

The contents of this commit are extracted from a larger set proposed
originally by Lukas Fittl, that Sami Imseih has proposed as an
independent change, with a few tweaks sprinkled by me.

Author: Lukas Fittl <lukas@fittl.com>
Author: Sami Imseih <samimseih@gmail.com>
Reviewed-by: Bertrand Drouvot <bertranddrouvot.pg@gmail.com>
Reviewed-by: Michael Paquier <michael@paquier.xyz>
Discussion: https://postgr.es/m/CAP53Pkyow59ajFMHGpmb1BK9WHDypaWtUsS_5DoYUEfsa_Hktg@mail.gmail.com
Discussion: https://postgr.es/m/CAA5RZ0vyWd4r35uUBUmhngv8XqeiJUkJDDKkLf5LCoWxv-t_pw@mail.gmail.com
```

```
typedef struct PlannedStmt
{
    pg_node_attr(no_equal, no_query_jumble)

    NodeTag     type;

    /* select|insert|update|delete|merge|utility */
    CmdType     commandType;

    /* query identifier (copied from Query) */
    uint64      queryId;

    /* plan identifier (can be set by plugins) */
    uint64      planId;
```

In Postgres 18, you can now write an extension that sets **PlannedStmt.planId** in a **planner_hook,** and then uses it in **ExecutorFinish_hook** to track statistics.

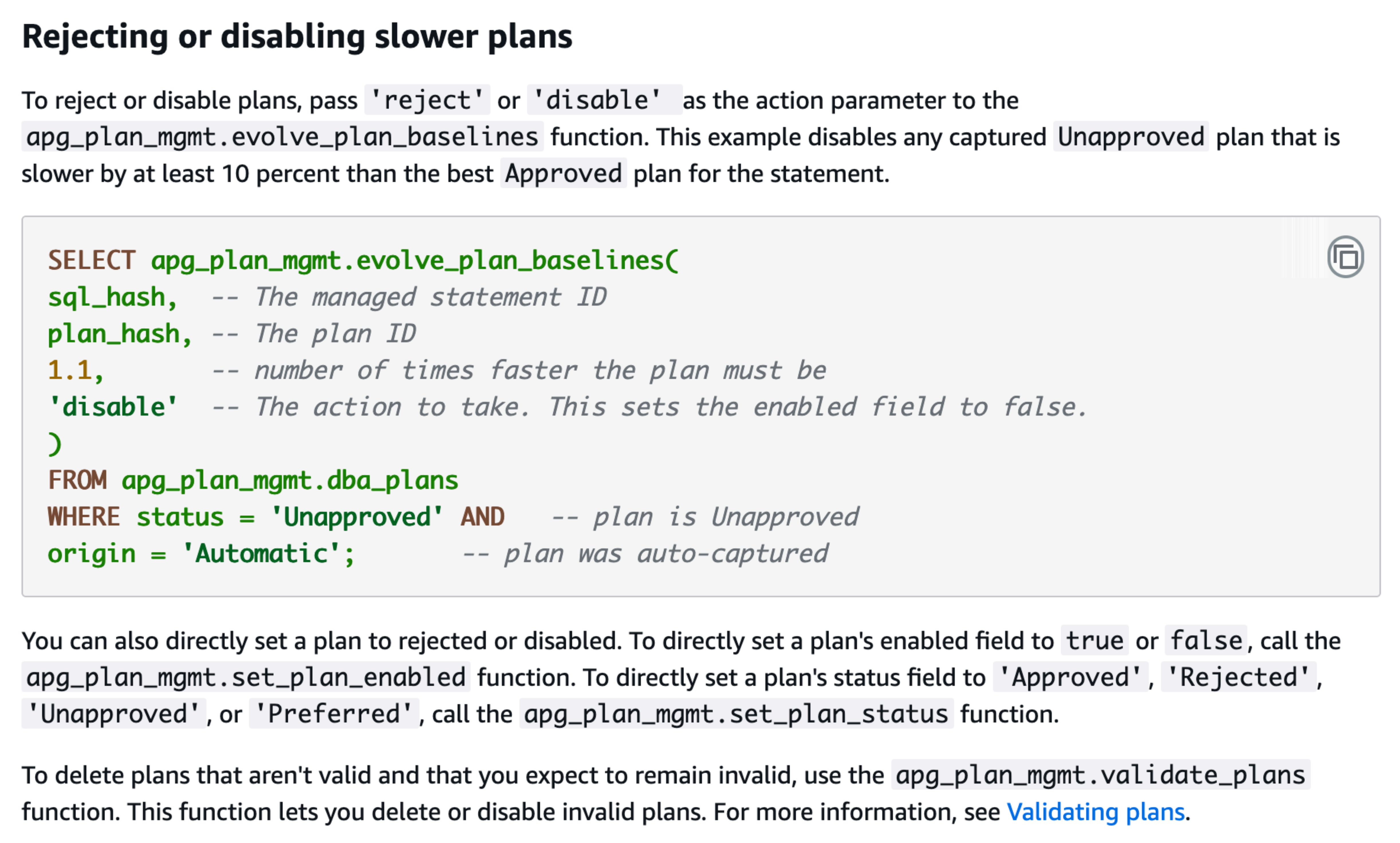
This enables **pg_stat_plans_activity** (plan for current queries).

pganalyze

Fixing bad query plans
with pganalyze Query Advisor

How could we fix a potential plan regression?

Aurora Query Plan Management is one solution:



Rejecting or disabling slower plans

To reject or disable plans, pass `'reject'` or `'disable'` as the action parameter to the `apg_plan_mgmt.evolve_plan_baselines` function. This example disables any captured `Unapproved` plan that is slower by at least 10 percent than the best `Approved` plan for the statement.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND    -- plan is Unapproved
origin = 'Automatic';        -- plan was auto-captured
```

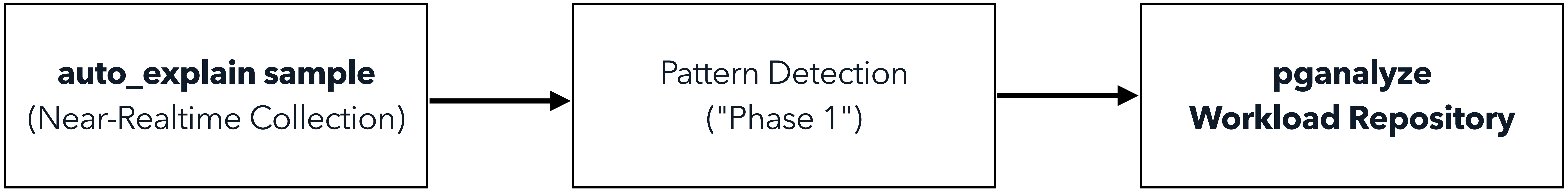
You can also directly set a plan to rejected or disabled. To directly set a plan's enabled field to `true` or `false`, call the `apg_plan_mgmt.set_plan_enabled` function. To directly set a plan's status field to `'Approved'`, `'Rejected'`, `'Unapproved'`, or `'Preferred'`, call the `apg_plan_mgmt.set_plan_status` function.

To delete plans that aren't valid and that you expect to remain invalid, use the `apg_plan_mgmt.validate_plans` function. This function lets you delete or disable invalid plans. For more information, see Validating plans.

But it has no open-source alternative. **And you need to produce the good plan.**

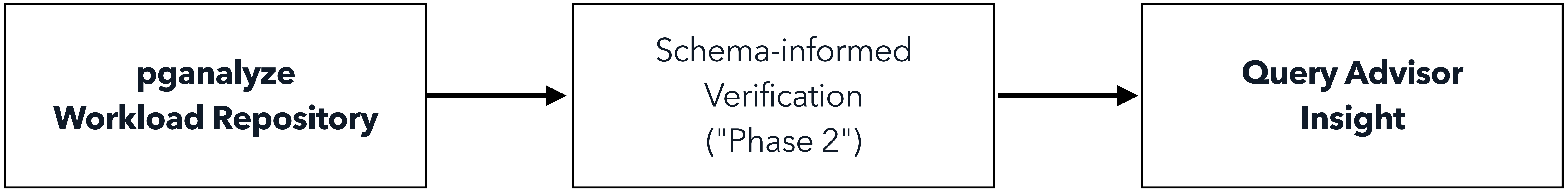
With pganalyze Query Advisor, we are introducing a different approach.

Utilizing EXPLAIN plan data from auto_explain, or manually uploaded plans, we detect pathological patterns like row mis-estimates, wrong index use & more.



(This uses auto_explain, since its more widely available than plan statistics today)

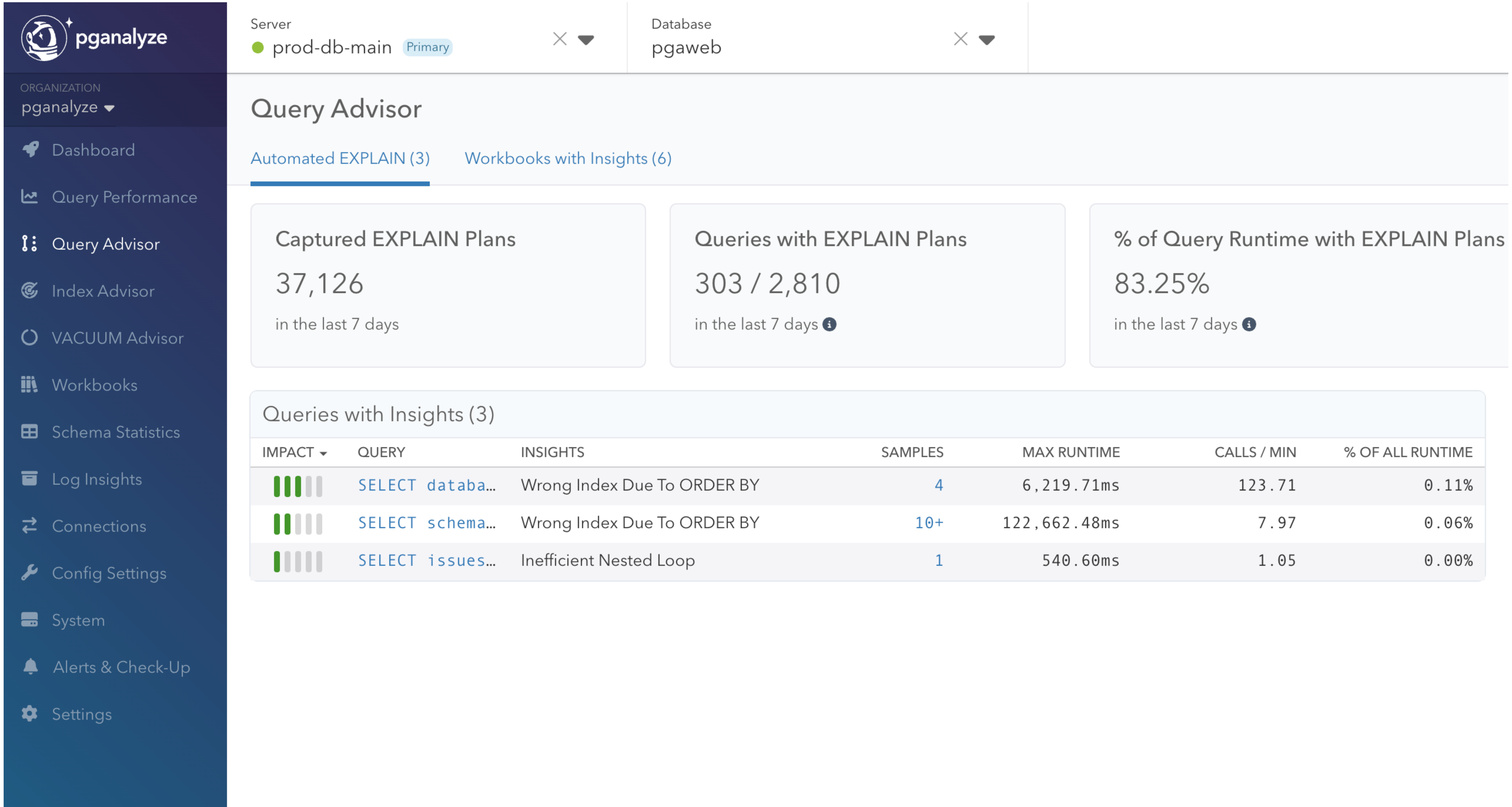
We cross-reference EXPLAIN plan data with schema information, **and create query-specific insights and rewrite recommendations.**


pganalyze Workload Repository
Schema-informed Verification ("Phase 2")
Query Advisor Insight

Let's look at an example insight!

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Server
prod-db-main Primary

Database
pgaweb

Query Advisor

Automated EXPLAIN (3) Workbooks with Insights (6)

Captured EXPLAIN Plans
37,126
in the last 7 days

Queries with EXPLAIN Plans
303 / 2,810
in the last 7 days

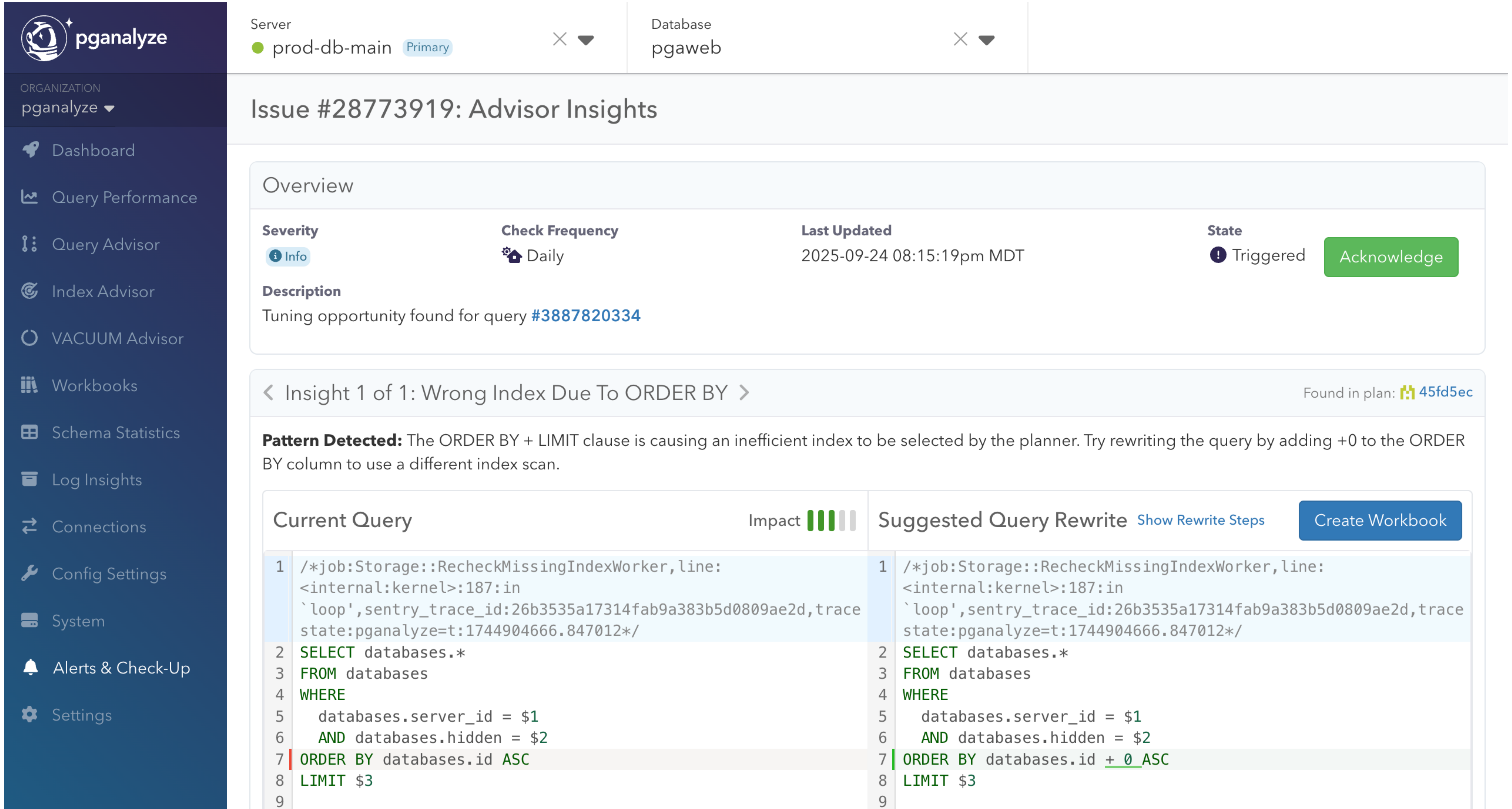
% of Query Runtime with EXPLAIN Plans
83.25%
in the last 7 days

Queries with Insights (3)

IMPACT	QUERY	INSIGHTS	SAMPLES	MAX RUNTIME	CALLS / MIN	% OF ALL RUNTIME
▮▮▮▯▯	SELECT databa…	Wrong Index Due To ORDER BY	4	6,219.71ms	123.71	0.11%
▮▮▯▯▯	SELECT schema…	Wrong Index Due To ORDER BY	10+	122,662.48ms	7.97	0.06%
▮▯▯▯▯	SELECT issues…	Inefficient Nested Loop	1	540.60ms	1.05	0.00%

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Issue #28773919: Advisor Insights

Overview

Severity: Info
Check Frequency: Daily
Last Updated: 2025-09-24 08:15:19pm MDT
State: Triggered

Description

Tuning opportunity found for query #3887820334

Insight 1 of 1: Wrong Index Due To ORDER BY

Found in plan: 45fd5ec

Pattern Detected: The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner. Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

Current Query

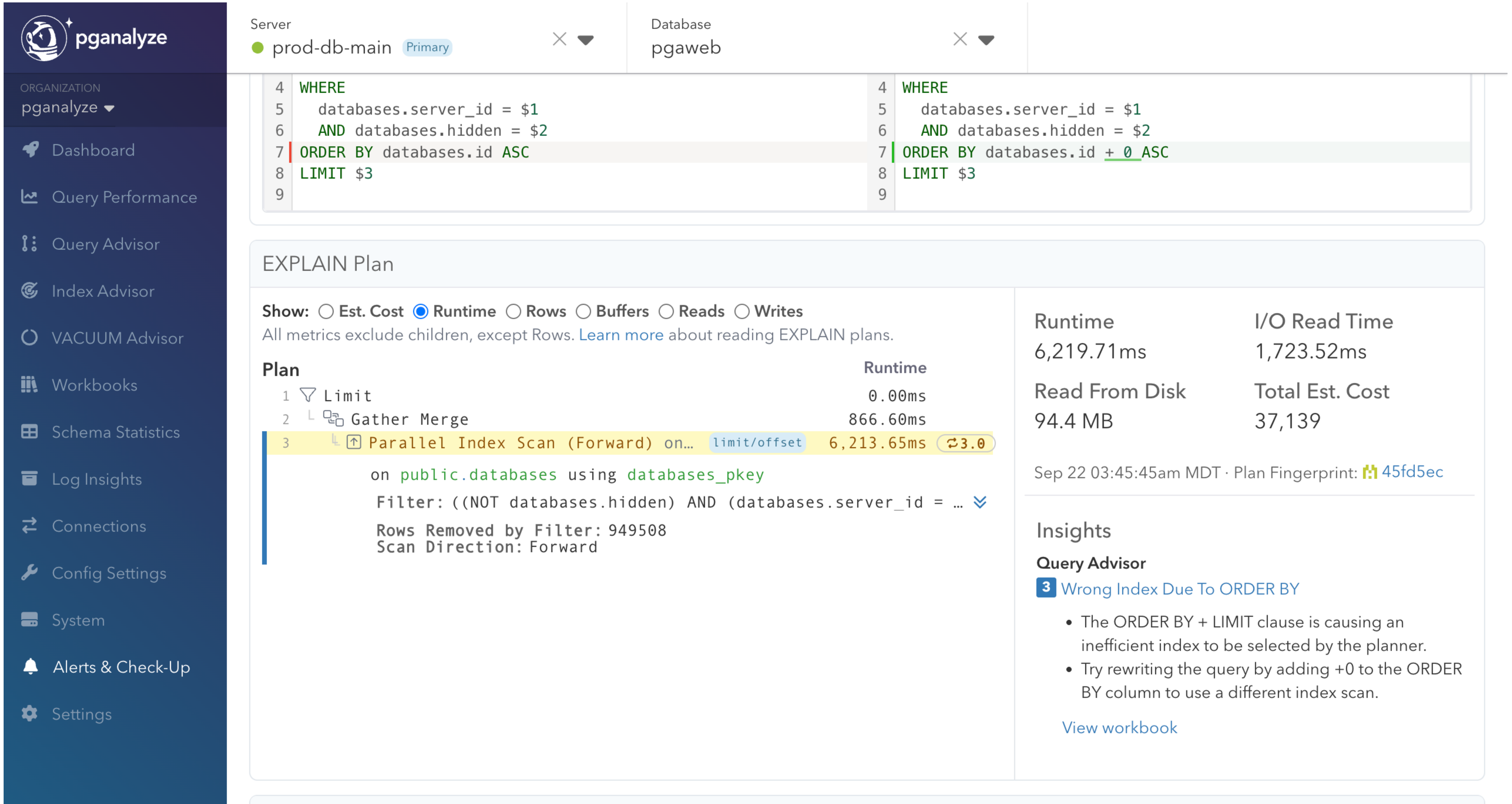
```
/*job:Storage::RecheckMissingIndexWorker,line:
<internal:kernel>:187:in
`loop',sentry_trace_id:26b3535a17314fab9a383b5d0809ae2d,trace
state:pganalyze=t:1744904666.847012*/
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $1
  AND databases.hidden = $2
ORDER BY databases.id ASC
LIMIT $3
```

Suggested Query Rewrite

```
/*job:Storage::RecheckMissingIndexWorker,line:
<internal:kernel>:187:in
`loop',sentry_trace_id:26b3535a17314fab9a383b5d0809ae2d,trace
state:pganalyze=t:1744904666.847012*/
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $1
  AND databases.hidden = $2
ORDER BY databases.id + 0 ASC
LIMIT $3
```

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Server: prod-db-main Primary

Database: pgaweb

```
WHERE
  databases.server_id = $1
  AND databases.hidden = $2
ORDER BY databases.id ASC
LIMIT $3
```

```
WHERE
  databases.server_id = $1
  AND databases.hidden = $2
ORDER BY databases.id + 0 ASC
LIMIT $3
```

EXPLAIN Plan

Show: ( ) Est. Cost [✓ Runtime] ( ) Rows ( ) Buffers ( ) Reads ( ) Writes

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

Plan	Runtime
1 Limit	0.00ms
2 Gather Merge	866.60ms
3 Parallel Index Scan (Forward) on… limit/offset	6,213.65ms 3.0

```
on public.databases using databases_pkey
Filter: ((NOT databases.hidden) AND (databases.server_id = …
Rows Removed by Filter: 949508
Scan Direction: Forward
```

Runtime 6,219.71ms	I/O Read Time 1,723.52ms
Read From Disk 94.4 MB	Total Est. Cost 37,139

Sep 22 03:45:45am MDT · Plan Fingerprint: 45fd5ec

Insights

Query Advisor

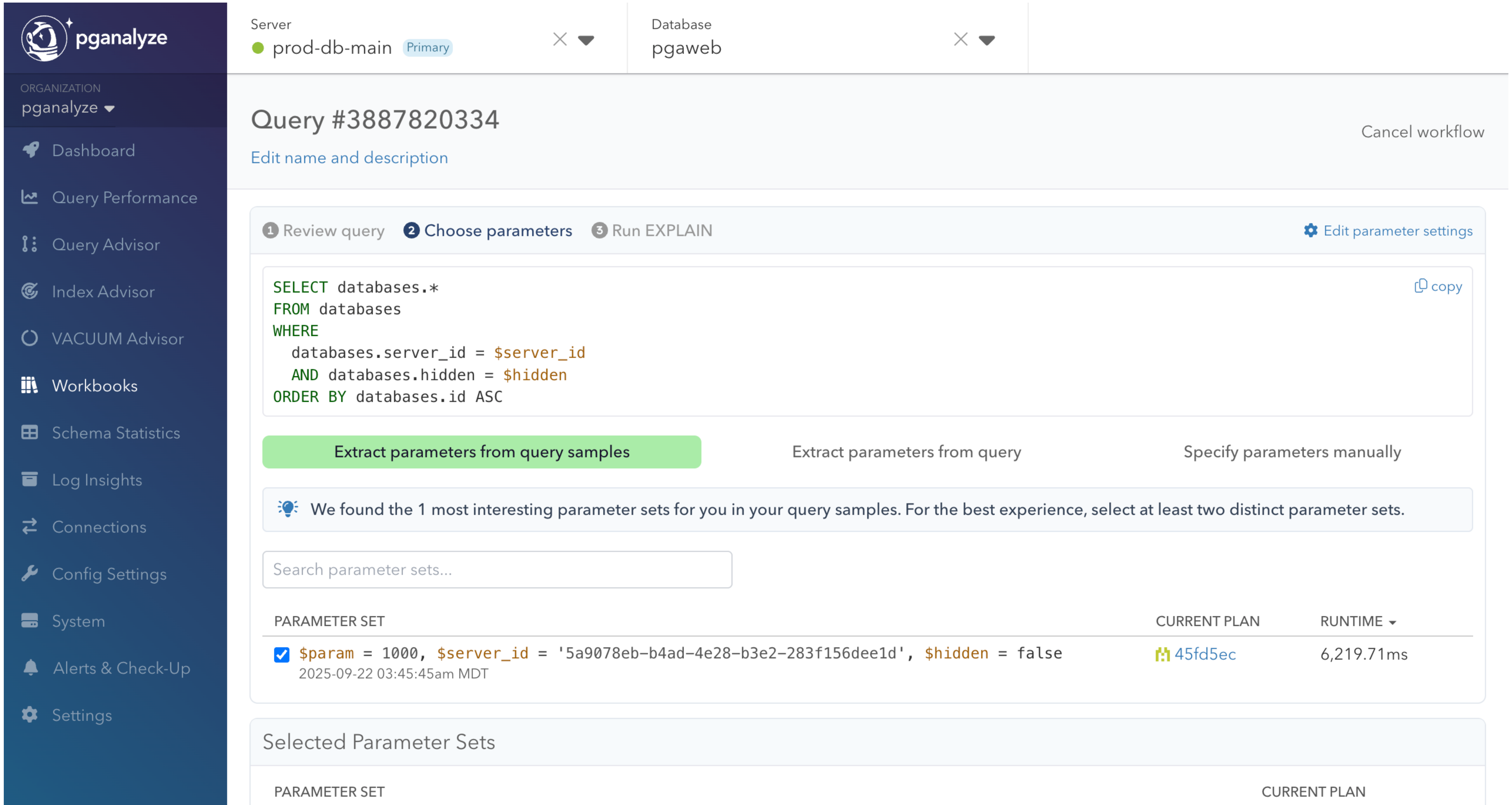
3 Wrong Index Due To ORDER BY

- The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner.
- Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

View workbook

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Server: prod-db-main Primary

Database: pgaweb

Query #3887820334

```
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $server_id
  AND databases.hidden = $hidden
ORDER BY databases.id ASC
```

We found the 1 most interesting parameter sets for you in your query samples. For the best experience, select at least two distinct parameter sets.

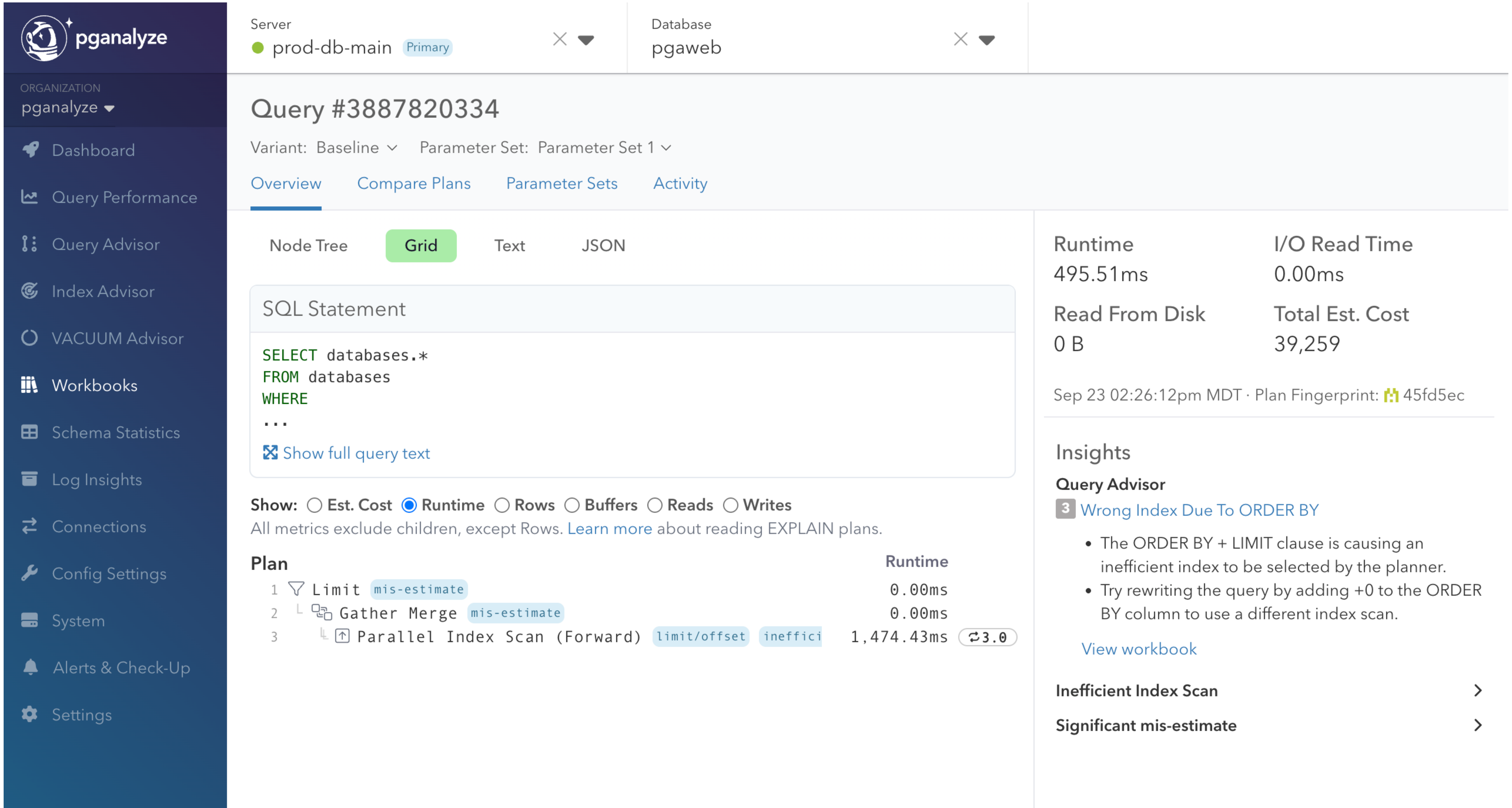
PARAMETER SET	CURRENT PLAN	RUNTIME
✓ $param = 1000, $server_id = '5a9078eb-b4ad-4e28-b3e2-283f156dee1d', $hidden = false 2025-09-22 03:45:45am MDT	45fd5ec	6,219.71ms

Selected Parameter Sets

PARAMETER SET	CURRENT PLAN

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Server: prod-db-main Primary

Database: pgaweb

Query #3887820334

Variant: Baseline   Parameter Set: Parameter Set 1

Overview | Compare Plans | Parameter Sets | Activity

Node Tree | Grid | Text | JSON

SQL Statement

```
SELECT databases.*
FROM databases
WHERE
...
```

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                                                0.00ms
2  └ Gather Merge  mis-estimate                                       0.00ms
3    └ Parallel Index Scan (Forward)  limit/offset  ineffici     1,474.43ms  ⇄3.0
```

Runtime	I/O Read Time
495.51ms	0.00ms
Read From Disk	**Total Est. Cost**
0 B	39,259

Sep 23 02:26:12pm MDT · Plan Fingerprint: 45fd5ec

Insights

Query Advisor

3 Wrong Index Due To ORDER BY

- The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner.
- Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

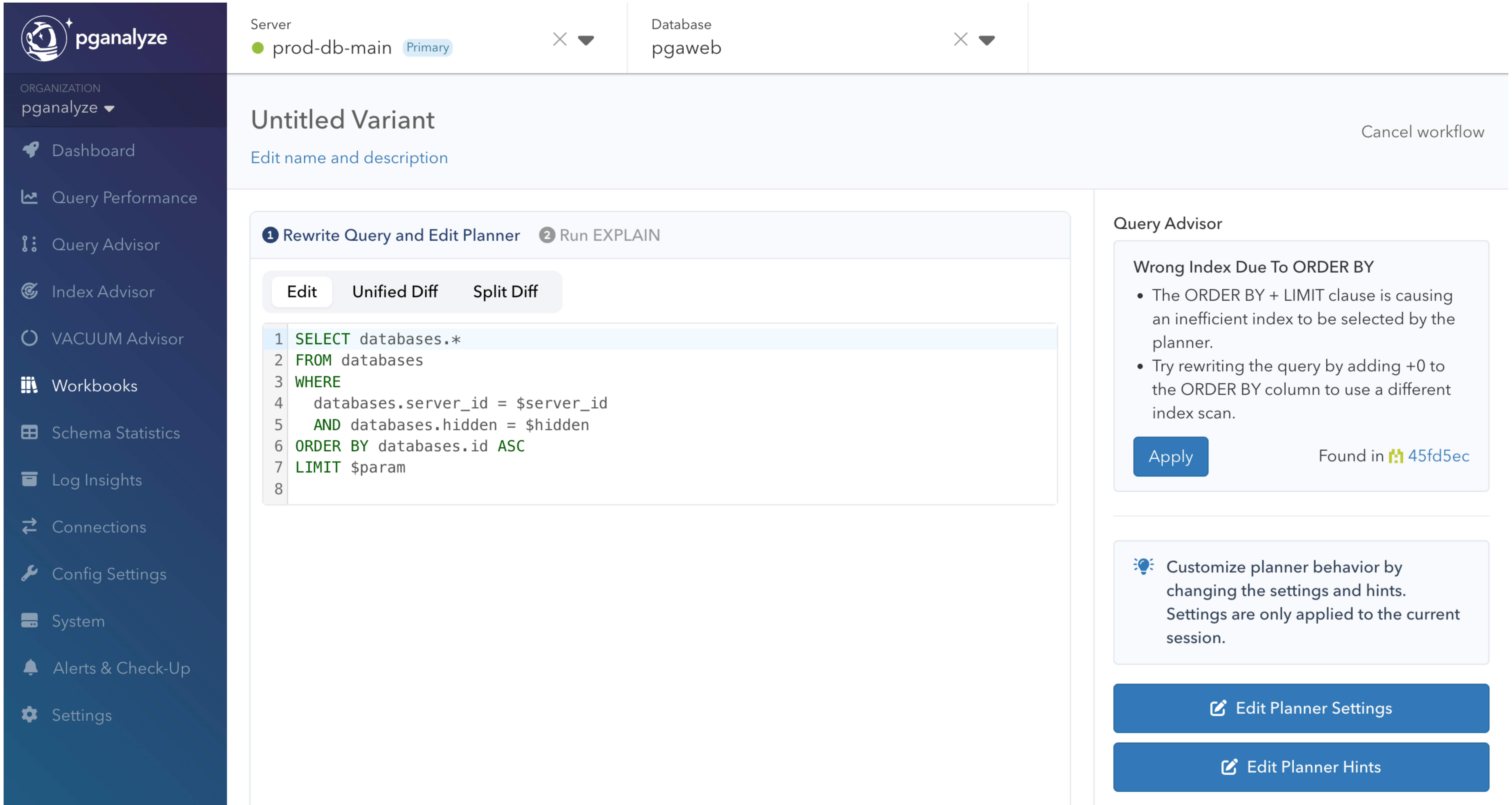
View workbook

Inefficient Index Scan

Significant mis-estimate

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Untitled Variant

```sql
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $server_id
  AND databases.hidden = $hidden
ORDER BY databases.id ASC
LIMIT $param
```

Query Advisor

Wrong Index Due To ORDER BY

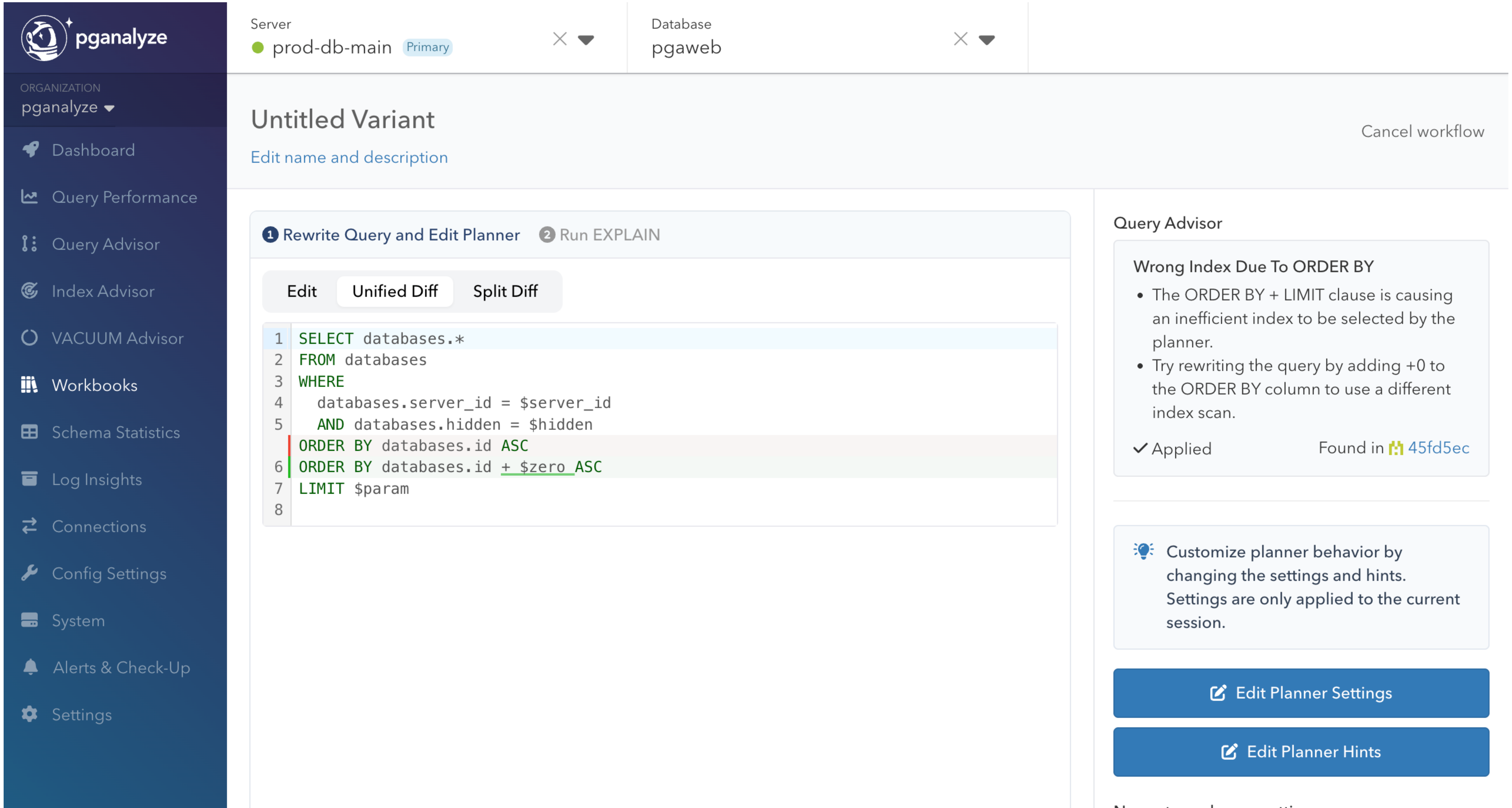
- The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner.
- Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

Found in 45fd5ec

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

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Untitled Variant

```
SELECT databases.*
FROM databases
WHERE
  databases.server_id = $server_id
  AND databases.hidden = $hidden
ORDER BY databases.id ASC
ORDER BY databases.id + $zero ASC
LIMIT $param
```

Query Advisor

Wrong Index Due To ORDER BY

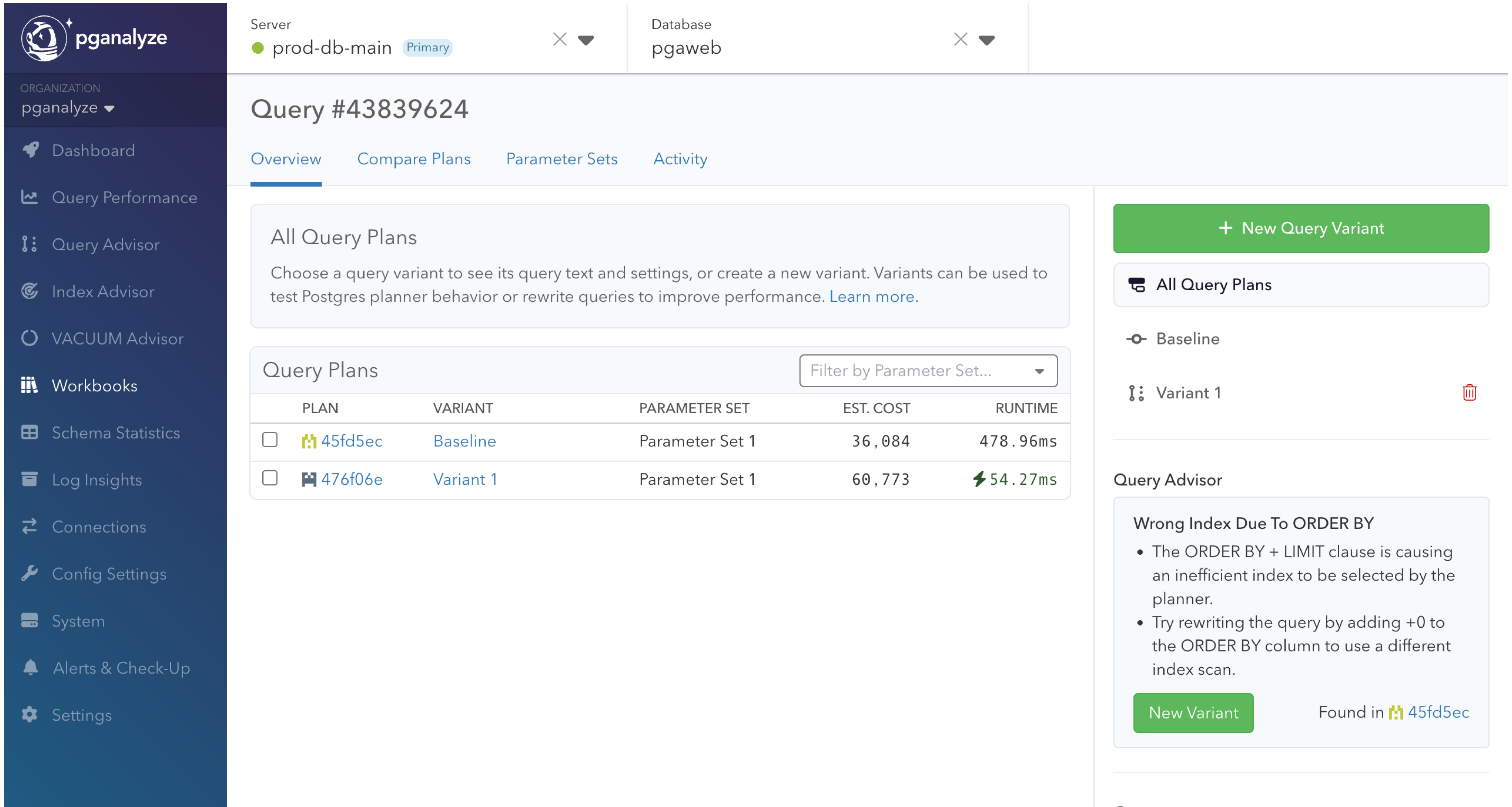
- The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner.
- Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

✓ Applied Found in 45fd5ec

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

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Query #43839624

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

PLAN	VARIANT	PARAMETER SET	EST. COST	RUNTIME
45fd5ec	Baseline	Parameter Set 1	36,084	478.96ms
476f06e	Variant 1	Parameter Set 1	60,773	54.27ms

Query Advisor

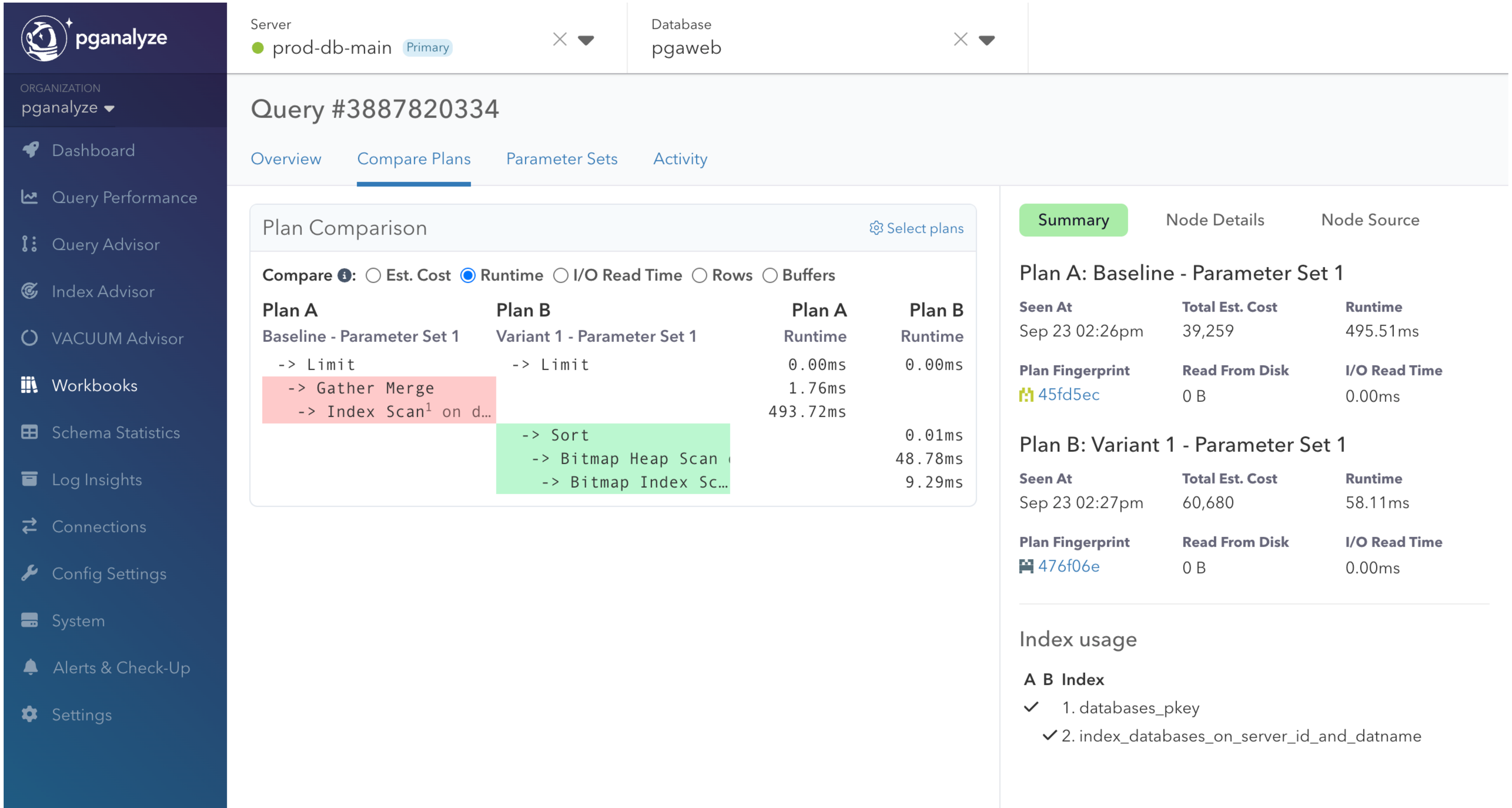
Wrong Index Due To ORDER BY

- The ORDER BY + LIMIT clause is causing an inefficient index to be selected by the planner.
- Try rewriting the query by adding +0 to the ORDER BY column to use a different index scan.

Found in 45fd5ec

pganalyze

Fixing bad query plans with pganalyze Query Advisor



Server: prod-db-main Primary

Database: pgaweb

Query #3887820334

Overview | Compare Plans | Parameter Sets | Activity

Plan Comparison

Select plans

Compare: ( ) Est. Cost [✓ Runtime] ( ) I/O Read Time ( ) Rows ( ) Buffers

Plan A Baseline - Parameter Set 1	Plan B Variant 1 - Parameter Set 1	Plan A Runtime	Plan B Runtime
-> Limit	-> Limit	0.00ms	0.00ms
🟥 -> Gather Merge		1.76ms	
🟥 -> Index Scan[1] on d…		493.72ms	
	🟩 -> Sort		0.01ms
	🟩 -> Bitmap Heap Scan		48.78ms
	🟩 -> Bitmap Index Sc…		9.29ms

Summary | Node Details | Node Source

Plan A: Baseline - Parameter Set 1

Seen At	Total Est. Cost	Runtime
Sep 23 02:26pm	39,259	495.51ms
Plan Fingerprint	**Read From Disk**	**I/O Read Time**
45fd5ec	0 B	0.00ms

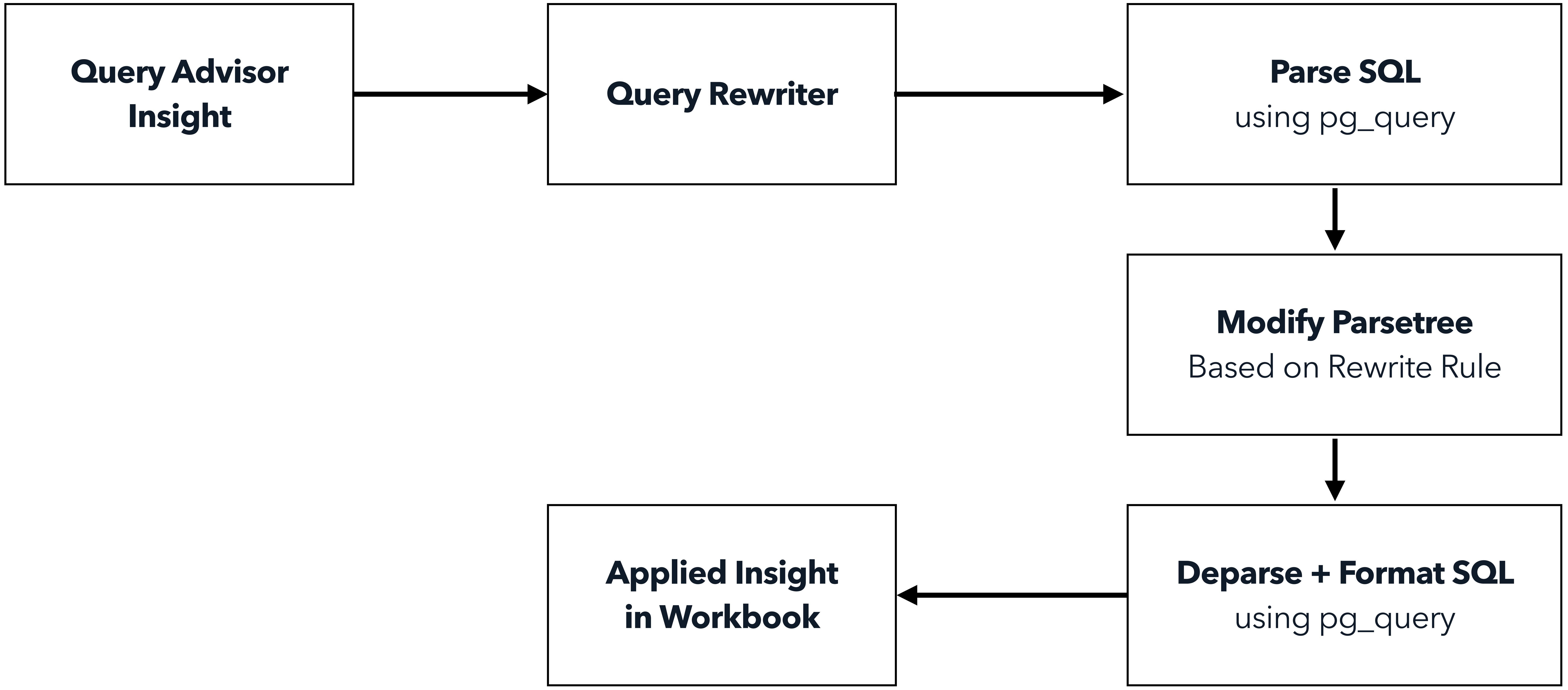
Plan B: Variant 1 - Parameter Set 1

Seen At	Total Est. Cost	Runtime
Sep 23 02:27pm	60,680	58.11ms
Plan Fingerprint	**Read From Disk**	**I/O Read Time**
476f06e	0 B	0.00ms

Index usage

A	B	Index
✓		1. databases_pkey
	✓	2. index_databases_on_server_id_and_datname

Rewrites are "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

Additional insights in the works:

- Other cases of Wrong Index Use
- OR => UNION transformation
- Memoize mis-estimates
- GROUP BY column ordering
- Planner hint suggestions
- Settings changes
  - work_mem
  - random_page_cost
  - etc.

pganalyze Query 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/GenAI doesn't scale for automated analysis)

Available today!
(and does not require Plan Statistics or Postgres 18!)

Stop by the pganalyze booth for a live demo.

pganalyze

Thank you!

Try out pganalyze:

PGANALYZE.COM

Reach out for any questions:

lukas@pganalyze.com
