

pg_stat_statements: Scaling for the Future

Sami Imseih

Senior Software Development Engineer –
PostgreSQL Contributors Team



A timeline of pg_stat_statements

8.4 · 2009

Introduced, with **4 fields**: calls, total_time, rows, usage

9.2 · 2012

Query normalization (queryid), block I/O timing

9.4 · 2014

Query texts move from shared memory to a file (f0d6f20278b)

13–19 · 2020–26

Planning, WAL, JIT, I/O timing, parallel, plan cache: **42 fields**

The design since 8.4

- One shared hash table
- One global LWLock
- One spinlock per entry, around every counter update

Since 2009, workloads, hardware, and the counter list have **all grown**.

A timeline of pg_stat_statements

commit f0d6f20278b7c5c412ce40a9b86c6b31dc2fbfdd

Author: Tom Lane <tgl@sss.pgh.pa.us>

Date: Mon Jan 27 15:37:54 2014 -0500

8.4 · 200

Keep pg_stat_statements' query texts in a file, not in shared memory.

9.2 · 201

This change allows us to eliminate the previous limit on stored query length, and it makes the shared-memory hash table very much smaller, allowing more statements to be tracked. (The default value of pg_stat_statements.max is therefore increased from 1000 to 5000.)

In typical scenarios, the hash table can be large enough to hold all the statements commonly issued by an application, so that there is little "churn" in the set of tracked statements, and thus little need to do I/O to the file.

9.4 · 201

To further reduce the need for I/O to the query-texts file, add a way to retrieve all the columns of the pg_stat_statements view except for the query text column. This is probably not of much interest for human use but it could be exploited by programs, which will prefer using the queryid anyway.

13–19 · 2020

Ordinarily, we'd need to bump the extension version number for the latter change. But since we already advanced pg_stat_statements' version number from 1.1 to 1.2 in the 9.4 development cycle, it seems all right to just redefine what 1.2 means.

Since 200

Peter Geoghegan, reviewed by Pavel Stehule

The happy path: why it's usually OK

Every executed statement (and planned, with `track_planning`) calls **`pgss_store()`**:

- 1 Take the global LWLock in **shared** mode
- 2 Look up (userid, dbid, queryid, toplevel)
- 3 Take the entry's **spinlock**, update the counters, release it

For a query that's already tracked, no backend needs the global lock in exclusive mode.
This is the common case for most systems.

...until it's not

Not OK #1: many unique queries

When the query is new, or the table is at max:

- A new entry needs the global LWLock that protects the shared hash table, in **exclusive** mode. That blocks every backend, including ones running hot, already-tracked queries
- At max, entry_dealloc() scans every entry, runs a **qsort**, and evicts 5%, still holding that lock

Where uniqueness comes from

- ORMs and generated SQL
- IN lists, temp tables, schemas (better since 18)
- Per user and database: $N \text{ roles} \times M \text{ DBs} = N \times M$ entries

Normalization keeps improving

Each change groups more similar queries under one queryid

Release	Commits	Now grouped together
9.2	7313cc01634	Queries that differ only in constants
16	daa8365a900	Utility statements that differ only in constants
17	31de7e60da3	SAVEPOINT names (similar commits: 2PC, DEALLOCATE, CALL)
18	62d712ecfd9	IN lists of any length
18	787514b30bb	Same-named tables, such as temp tables
18	dc68515968e	SET statements that differ only in values

Still hard to predict how many distinct queries you'll have, and **max is fixed at server start.**

Not OK #2: large queries

Query texts live in an external file: **pgss_query_texts.stat**

- Garbage collection rewrites the **whole file under the exclusive lock**
- Reading the view scans every entry **under the shared lock**, blocking new entries
- Nothing bounds the total size of query texts

Monitoring tools that poll the view add to the contention.

Not OK #3: big machines and the spinlock

One spinlock per entry, taken on **every** execution: **42 fields** updated while it's held

Every backend running the same statement takes **the same spinlock** and writes **the same counters**, so that entry's cache lines bounce between cores.

Proposal and agreement at PGConf.dev

Design goal: the worst case, high churn or a hot spinlock, must not bring the system to a halt

The proposal

- Use the **same cumulative statistics** as other core stats
- Accumulate counters per backend, flush at transaction end
- Query text in shared memory, spilling to disk if needed
- Let max be exceeded slightly; maybe evict in a background worker

What was agreed

- Keep `pg_stat_statements.max`, plus a memory limit for query text
- Handle query text separately from the stats
- Make eviction parallel instead of serialized; LRU may beat LFU
- No background worker: backends shouldn't wait on one

[PGConf.dev 2026 unconference: wiki.postgresql.org/wiki/Scalability_of_pg_stat_statements](https://wiki.postgresql.org/wiki/Scalability_of_pg_stat_statements)

Upstream vs patch

	Upstream	Patch
Counter updates	Spinlock, every execution	Local first, flushed $\leq$ once a second
Visibility	Immediate	After the session flushes
Query text	File; GC under an exclusive lock	Shared memory, capped (4 MB)
max	Fixed at server start	Reloadable
When full	Evict 5% at max; others wait	One backend evicts at 110%; may drop new
Eviction ranking	Usage count, decayed per pass	Same
Not yet settled		LRU vs LFU (No Parallel Eviction)

Benchmark methodology

- **Steady state** (select_only): pgbench -S, **1 unique query**, no churn
- **High churn** (hotcold_100m): 100M possible query texts; **90%** of executions hit 4,000 common ones
- **Measured:** throughput, wait events, and whether the 4,000 hot queries **stay visible**
- c7i.8xlarge (32 vCPU), release builds, max = 5000, clients 1 to 256, 600 s per point

Left to do: a workload with exactly max unique queries, where nothing should be evicted

Scripts and full reports: github.com/samimseih/pg_stat_statements_benchmark

Results: churn workload

hotcold_100m, TPS by client count

clients	upstream TPS	patch TPS	delta
8	217,441	243,481	+12.0%
16	365,666	428,139	+17.1%
32	500,666	1,023,364	+104.4%
64	536,638	923,901	+72.2%
128	526,677	728,773	+38.4%
256	494,291	783,020	+58.4%

- Upstream stops growing at **about 500k TPS**
- Patch reaches **about 1M** at 32 clients (the vCPU count)

Results: churn, retention

hotcold_100m at 32 clients, sampled every 5 s

	upstream	patch
Popular statements kept	4,000 of 4,000	4,000 of 4,000
One-shot entries kept (avg), in progress	900	220
Eviction passes (dealloc)	111,115	70,533
Throughput (TPS)	500,666	1,023,364

Hot statements stay visible; cold ones go sooner. Eviction starts at 110% of max, and **max is now dynamic**, so users can raise it without a restart.

Results: churn, where the waiting went

Wait-event samples from pg_stat_activity, 1 Hz, upstream vs work-in-progress patch (1d8654a01fc)

event (hotcold_100m)	upstream	patch
pg_stat_statements LWLock	148,857	0
new pgstat and query-text LWLocks	0	7,415
SpinDelay	3,793	417 (–89%)
All wait events (excluding ClientRead)	152,650	7,832 (–95%)
On CPU (no wait event)	26,449	23,006 (–13%)

Wait-event samples drop **95%**, and backends seen on CPU drop 13%, while throughput goes up.

Results: select_only

pgbench -S, prepared, scale 10: **1 unique query**, so every client hits the same entry

clients	upstream TPS	patch TPS	delta
16	420,457	421,856	+0.3%
32	1,014,386	1,021,316	+0.7%
64	913,906	912,255	-0.2%
128	792,023	768,084	-3.0%
256	684,201	688,628	+0.6%

SpinDelay samples

upstream: 5,256

patch: 456

-91%

TPS is about the same, but spinning drops 91%. At 32 vCPUs it isn't limiting throughput yet, but it's measurable.

Already in core

Committed pgstat and shared-memory infrastructure this work builds on

Commits	Release	What it provides
7949d959458	18	Pluggable cumulative statistics kinds
4ba012a8ed9	19	Auxiliary data in the stats file
e65c331b8fb	master	Split pending-data setup
44539786b82	master	A dedicated dshash per stats kind (own_hash)

Still needed in core

Small pgstat changes the pg_stat_statements patches depend on, not yet committed

Commit	Change	Status
16eaeff22f0	Allow pg_stat_force_next_flush() to work inside a transaction	Heavily reviewed
056b449de7e	Add a kind-specific drop-matching helper	Need found recently

The pg_stat_statements patches

Branch pg_stat_statements_v14 (work in progress)

Commit	Change
c0f1ee3f8fb	Use custom pgstat entries for pg_stat_statements
1d8654a01fc	Store pg_stat_statements query texts in DSA

With the core pieces now committed, **commitfest 6763 resumes** to continue the work.

[Commitfest: commitfest.postgresql.org/patch/6763](https://commitfest.postgresql.org/patch/6763)

[Branch: github.com/postgres/postgres/compare/master...samimseih:postgres:pg_stat_statements_v14](https://github.com/postgres/postgres/compare/master...samimseih:postgres:pg_stat_statements_v14)

Thank you!

Sami Imseih

AWS

