

The AI Tuning Loop to Fix Slow Postgres Queries

Claude Code and other AI tools iterate on real EXPLAIN ANALYZE results from your production workload, safely collected through pganalyze.

Postgres AI-Enabled Tuning Loop



Why slow queries stay slow

Every query optimization is a loop: benchmark the slow query, form a thesis, try a change, benchmark again. Run by hand, that loop is expensive. In a recent demo, it took an expert who knew the data model 45 minutes; for anyone less familiar, the same investigation takes hours, sometimes days. That cost is why production systems stay full of optimization opportunities nobody takes, and why the loop is worth automating end to end.

The loop, step by step

1

Slow query alert

The loop starts from real evidence: an alert, an APM trace, or a support ticket that points at one query. In the demo, a customer reported a slow page; the trace showed an 11-second request with 8.6 seconds spent in a single query, and the EXPLAIN plan pganalyze had recorded from production showed the runtime was almost entirely I/O read time. That plan is the baseline every rewrite is benchmarked against.

2

Form a thesis

Read the plan for what stands out, and name the change worth trying. In the demo the standout was a nested loop repeating the same index lookup 2,000 times across daily partitions the planner could not skip. The thesis: give Postgres a filter it can use to prune those partitions. This step stays human; the tools' job is telling you whether an opportunity is worth your time at all.

3

Draft query variants

Write several candidate rewrites, not one, and record what each is trying to do. The demo produced variants that bounded the join with two-day and seven-day look-back filters so the planner could skip most partitions, alongside a hand-written rewrite that materialized the looked-up data first. Keeping the thesis attached to each variant means the attempts stay reviewable instead of scattering across a chat log.

4

Validate performance

Run every variant with EXPLAIN ANALYZE against production data, collected through pganalyze inside a read-only transaction with a timeout. The benchmark decides, not intuition: the demo's warm-cache baseline of 150 milliseconds dropped to 13 with the best first-round variant, the hand-tuned rewrite hit 5, and a hybrid of the two ideas ran in 4.1 milliseconds. Testing locally would have hidden all of this: small tables and warm caches make everything look fast.

5

Ship the changes

The winning variant becomes an application-side code change, ORM code included, with the EXPLAIN ANALYZE results attached as evidence. Then the loop closes: the same alerting that surfaced this query surfaces the next one, and the fix that took days of expert attention now compresses into a supervised session.

Benchmarked, not guessed

A coding agent writes better code when it can run the tests itself. Query tuning works the same way: a model can propose a rewrite that looks right, but it cannot know what the rewrite actually does to runtime. The actual performance data has to flow back into the loop.

EXPLAIN ANALYZE is the test for the query rewrite. It's the way that we know that the query is actually improved and not just theoretically improved.

— Lukas Fittl, founder of pganalyze

AGENT BACKPRESSURE

An agent that must ask a human for every EXPLAIN ANALYZE result runs at human speed, and one that never benchmarks keeps iterating in the wrong direction. "Claude can't know what the performance was, and oftentimes it will get things wrong, but the backpressure of the actual performance data will get it zoned in on the right direction."

What safe validation requires

- **No direct database access.** The agent never connects to Postgres; the pganalyze collector, running in the team's own environment with the read-only `pg_monitor` role, does all the reading.
- **One narrow capability, not a shell.** The agent can run EXPLAIN ANALYZE and nothing else: a forced read-only transaction, protected by a timeout, no arbitrary SQL.
- **An audit trail humans can review.** Every run is recorded as a workbook in pganalyze, with attribution separating what the agent did from what a person did.



The demo in this brief pairs Claude Code with the pganalyze MCP Server and Workbooks. Watch it end to end, from the first trace to the 4.1 millisecond hybrid variant, or run the loop on your own workload.

[Watch the demo](#)

[Start a free trial](#)

