EDB POSTGRES AI

10x Faster Time to Optimize: Benchmarking the Agentic Database Capabilities of EDB Postgres® AI

INTERNALLY VALIDATED REPORT

enterprisedb.com

Contents

Executive summary

For 20 years, EnterpriseDB (EDB) has made Postgres work for the enterprise, with hardened features and extensions; expert tuning; and mission-critical availability, security, and observability. The agentic database capabilities in EDB Postgres AI (EDB PG AI) extend that expertise into the operational loop itself: The platform observes a live cluster, reasons about what it sees, and recommends or applies fixes under governance the enterprise operator defines.

This report quantifies the operational impact of those capabilities. EDB benchmarked a single, extremely common scenario (a slow-query incident caused by a missing index) and measured the end-to-end effort required to detect, diagnose, and resolve it two ways: with a skilled DBA working manually, and with the agentic workflow in EDB PG AI. The test ran against LedgerDB, a representative financial-transactions workload with a transactions table of roughly 4.2 million rows.

The agentic workflow reduced end-to-end time to resolution from 60–85 minutes to 8–9 minutes, while raising the cluster's index health grade from D to A in under 6 minutes without taking the application offline.

The headline finding: The gap that agentic management closes is not just the *fix*, it is the *time to value*: the hour or longer a DBA spends identifying which query is slow, why, and what to do about it. The benchmark shows that work compresses to minutes, with the reasoning made explicit and every change governed and logged.

Key results

- **~90% reduction in time to resolution** for a missing-index incident (60–85+ minutes down to 8–9 minutes—up to 10x faster)
- **Index health grade restored from D to A** in under 6 minutes
- **Prioritized workload diagnosis in ~30 seconds**, versus an estimated 45–60 minutes of manual senior DBA analysis
- **More than 99% estimated query-cost reduction** from the recommended index in a test workload

What we measured

- **Time to resolution:** Total elapsed effort to detect, diagnose, fix, and verify the incident
- **Database health improvement:** The Recommendation Scorecard index grade, before and after
- **Diagnostic quality and speed:** Time to produce a prioritized, workload-specific recommendation
- **Query performance:** Estimated execution cost reduction from the recommended index

Methodology

The benchmark isolates a recurring, high-frequency operational task: A developer reports that checkout transactions are timing out, and the underlying cause is a missing composite index. EDB measured the full operator workflow (detection, diagnosis, fix authoring, application, and verification) under two conditions.

Test workload and environment

- **Workload:** LedgerDB, a financial-transactions OLTP workload
- **Primary table:** public.transactions, roughly 4.2M rows, actively growing
- **Incident under test:** Slow checkout queries caused by a missing composite index on filtered columns
- **Observation window:** About 10 minutes of live pg_stat_statements activity
- **Platform:** EDB PG AI
- **Capabilities engaged:** Recommendation Scorecard, AI Chatbot, automations, Task Manager, Activity Log

The two conditions compared

A. Manual expert-DBA workflow: A skilled DBA investigates from the console and terminal: reading slow-query logs and pg_stat_statements, analyzing EXPLAIN ANALYZE output, cross-referencing table statistics and recent schema changes, authoring the fix, applying it in a maintenance window, and verifying the result.

B. Agentic workflow (in EDB PG AI): The recommendation scorecard surfaces the issue and the exact fix; the AI Chatbot, connected live to the cluster, verifies the reasoning, queries, plan, and risks; the operator approves the task in Task Manager; the automation engine applies the index with CREATE INDEX CONCURRENTLY and the grade rechecked.

Capabilities under test

- **Recommendation Scorecard:** Continuous health grading (A–F) across indexes, statistics, configuration, security, and workload, with the exact remediating SQL.
- **AI Chatbot:** Conversational analysis grounded in the live cluster's own pg_stat_statements, not generic documentation.
- **Automation Engine:** Five types (disk, CPU, and memory autoscale; index auto-apply; minor-version upgrades), each with a configurable approval mode. Operators choose the spectrum of autonomy: full autonomous mode with no human in the loop, or semiautonomous mode that requires human approval before execution.
- **Task Manager:** Executes every action with the exact operation, approval status, and authorizing reason visible.
- **Activity Log:** An immutable, timestamped, attributable record of every recommendation, task, and approval.

Accuracy and human oversight

Recommendations are grounded in the live workload observed during the run, but a short observation window cannot account for every operational reality, such as a month-end batch job, a planned migration, or a query about to be retired. The system is therefore designed to surface its reasoning rather than act silently, and any action with material consequences waits for human approval. In this report, the operator remained the final decision-maker on the schema change, while the system performed the hours of investigation that normally precede that decision. That human-approval gate is a deliberate choice to reflect how enterprises currently run in production.

About this benchmark: Results reflect EDB's internal testing on a representative workload. Manual workflow timings represent typical expert DBA investigation effort for an incident of this class. Query cost and grade figures are produced by EDB PG AI against the observed LedgerDB workload. Absolute timings will vary with workload, data volume, and operator familiarity.

Deep dive on agentic database features and our testing

The Recommendation Scorecard: Your database's health report card

The first place to land when you open a cluster in the EDB PG AI console is the **Recommendation Scorecard**. Think of it as a continuous health check that never sleeps.

Your cluster's Postgres health is graded across five dimensions:

- **Indexes:** Is your workload actually using the right indexes? Are there obvious gaps?
- **Statistics:** Is the query planner working with fresh, accurate statistics?
- **Configuration:** Are your Postgres settings tuned for your environment and workload profile?
- **Security:** Are there hardening gaps—open roles, weak authentication, unencrypted connections?
- **Workload:** This is a forthcoming dimension that will add even deeper runtime analysis.

Each dimension gets a letter grade (A through F), not just a color, so there's no ambiguity about severity. But unlike a monitoring dashboard that just shows you a red light and leaves you to figure out what's wrong, **the scorecard shows you exactly what it found and exactly how to fix it.**

From D to A in under six minutes

Here's what this looks like in practice. A cluster running a financial transactions workload, LedgerDB, comes up with an index grade of D. In that time it accessed the database metrics (including `pg_stat_statements`), identified the slowest query patterns, analyzed which columns those queries filter on, and concluded that a specific index is missing.

The recommendation isn't vague. It looks like this:

```
CREATE INDEX CONCURRENTLY idx_transactions_status_account
ON public.transactions (status, account_id);
```

And it estimates that this single index will fully reduce query costs on the observed workload.

No log scraping. No cross-referencing query plans. No guessing. The system observed, reasoned, and produced the exact SQL. That's exactly the kind of work that shouldn't require an expert to do from scratch every time.

You can approve it right there (more on that in the Automations section). But first, how does the system know what it knows? That leads us to the chatbot.

The AI Chatbot: A senior DBA always on duty

The Recommendation Scorecard is the automated layer. The AI Chatbot is the conversational layer, and it's one of the most practically powerful tools in the console, once you understand what it's actually connected to.

Here's the critical distinction: **This chatbot is not talking to generic Postgres documentation.** It is connected live to your cluster. It reads `pg_stat_statements` directly. Every answer it gives you is grounded in what that specific database is actually doing right now—the actual queries, actual execution times, actual table structures, actual cardinality.

This changes the nature of the questions you can ask.

Starting with the real question: What's actually slow?

Instead of opening five tabs and writing queries against `pg_stat_statements` yourself, you ask:

> *"Looking at the queries running against LedgerDB, what schema changes would improve performance the most?"*

The chatbot's response isn't a list of Postgres best practices. It's an analysis of your workload:

- It pulls the **top queries by total execution time** from `pg_stat_statements` and shows you real SQL, real numbers.
- For each query, it explains the access pattern: This query filters on `status` and `account_id`, performs an aggregation, requires an ordered scan—which means it's doing a full sequential scan on a table that's growing fast.
- It produces **numbered, prioritized schema recommendations**, with the reasoning for each.

An example of what that looks like in practice:

1. **Composite index on `transactions(status, account_id)` – High priority**
   Your top query by total execution cost filters on both `status` (low cardinality, ~5 distinct values) and `account_id` (high cardinality, ~2M distinct values). A composite B-tree index with `status` first reduces the search space dramatically for equality filters on status before ranging over account IDs. Estimated cost reduction: 95%–100% for this query pattern.
2. **Partial index on `transactions(created_at) WHERE status = 'pending'`**
   A second high-frequency query is scanning for pending transactions in a time window. Since pending transactions are a small fraction of the total table (estimated 3%–5%), a partial index covering only this subset will be far smaller and faster than a full index on `created_at`.
3. **Consider `account_id` as a `BIGINT`, not `INTEGER`**
   At current growth rates, account_id will exceed INTEGER bounds within 18 months. Schema migration is significantly cheaper to do now than post-overflow.

This is not something the system invented. Every recommendation cites the specific query that motivated it, the column statistics it checked, and the reasoning chain. Without significant downtime, a senior DBA may struggle to anticipate growth and calculate accordingly. Conducting the analysis above would take 45–60 minutes, not 30 seconds.

Going deeper: Ask *why* a specific recommendation was made

One of the most useful interactions you can have with the chatbot is to follow up on something the Recommendation Scorecard surfaced. The scorecard told you *what* to do. The chatbot can explain *why*, in as much depth as you need.

Let's say the scorecard recommends the B-tree index on `transactions(status, account_id)`. You want to understand the reasoning before approving it, especially in a production environment. You ask the chatbot:

> *"The scorecard recommended a B-tree index on transactions(status, account_id). Walk me through why. What specific queries is it based on, what is the current execution plan, and are there any risks to applying it?"*

The chatbot's response walks you through the full chain:

> The recommendation is driven by three queries that together account for **67% of total execution time** on LedgerDB over the last 10 minutes:
>
> **Query 1** (42% of total execution time):

```
SELECT account_id, SUM(amount)
FROM transactions
WHERE status = 'completed' AND account_id = $1
GROUP BY account_id;
```

> Current plan: **Seq Scan** on transactions (estimated 4.2M rows scanned, cost: 87,000). With the composite index, this becomes an **Index Scan**, estimated cost: 8.
>
> **Query 2** (15% of total execution time):

```
SELECT * FROM transactions
WHERE status = 'pending' AND account_id = ANY($1)
ORDER BY created_at DESC LIMIT 50;
```

> Current plan: **Seq Scan + Sort**. With the index, **Index Scan** on the composite key, sort eliminated via index ordering.
>
> **Why a composite, not two separate indexes?** Because both columns appear together in the `WHERE` clause of the dominant query patterns. PostgreSQL can use a composite index for both equality + range lookups on this pattern. Two separate single-column indexes would require a bitmap heap scan merge, which is less efficient for this specific workload.
>
> **Risks:** The index will be built with `CONCURRENTLY`, so it will not lock reads or writes. Index build time on the current table size (~4.2M rows) is estimated at 2–4 minutes. During build, there is a slight overhead on writes. No risk of downtime. The only risk is if the workload pattern changes significantly, in which case the scorecard will flag the index as unused and recommend dropping it.

That's the kind of answer that would previously require a DBA to sit down, reproduce the slow query, pull `EXPLAIN ANALYZE`, cross-reference with table statistics, think through the index strategy options, and write it up for whoever asked. Now it takes one follow-up question.

Automations: The system that doesn't sleep

The chatbot and scorecard tell you what to do. **Automations** are how the system does it for you—on a schedule, within guardrails you define, with exactly as much or as little human oversight as the situation warrants.

The current release ships five automation types:

1. Disk autoscale

The problem it solves: Disk fills up, Postgres stops writing, everything goes down. It's entirely preventable, and yet it still happens.

How it works: When disk utilization hits a threshold—configured at 80%, for example—the system adds storage (10 GB increments) automatically, up to a hard cap (500 GB). We can configure whether approval is required or not. The reasoning is pragmatic: The worst outcome is a slightly larger disk. Nothing breaks. If a human is needed in the loop, we can configure it to require approval. If not, we just set it to "approval not required."

Without this: Someone is woken up at 2 a.m. when a disk hits 95%.

2. Index Recommendations — auto apply

The problem it solves: Performance issues accumulate between the time someone notices them and the time a DBA has capacity to investigate and act.

How it works: At every configured interval—for example, two minutes—the system evaluates the workload for missing or suboptimal indexes. When it identifies one, it queues an Apply Index Recommendation task in Task Manager. By default, this requires human approval, because an index build takes resources, and you often want a human in the loop for schema changes. When approved, the index is applied with `CREATE INDEX CONCURRENTLY`, so the application keeps serving traffic throughout the build.

The key design decision: You choose the approval mode. You can run it fully automated (the system applies indexes on its own) or with mandatory human sign-off. Instead of choosing between speed and control, you're configuring the balance that fits your environment.

3. Minor version upgrades

The problem it solves: Minor Postgres version upgrades contain bug fixes and security patches. They pile up because scheduling and executing them requires coordination of maintenance windows, change management sign-off, risk assessment.

How it works: The automation watches for new Postgres minor version images. When one is available, it queues an upgrade task pinned to your configured maintenance window (for example, Saturdays at 3 a.m. UTC). Just as with other tasks, you can configure it to require approval or not, and it flows through the same Task Manager and audit log as every other automated action. Change control is built in, not bolted on.

4–5. CPU and memory autoscale

The problem they solve: Workloads aren't flat. Batch jobs, end-of-month reporting runs, and traffic spikes are all part of fluctuating CPU and memory demands. Right-sizing in response to demand used to mean manual intervention or over-provisioning.

How they work: Like disk autoscale, these are threshold-triggered with configurable ceilings and approval modes. Both are available in the current version and configurable per cluster. Whether you want autonomous scaling or human-gated scaling is your call.

Task Manager: Where the work happens, visibly

Every automated action—index applies, version upgrades, scaling events—flows through the **Task Manager**. This is the execution engine, and it's designed to make every action legible.

When a task is queued, it shows up in Task Manager with:

- **What it is:** The specific recommendation it's acting on
- **What it will do:** The exact SQL or operation to be executed
- **Its approval status:** Pending, approved, running, succeeded, failed
- **Why it requires approval** (if applicable): The policy you configured

When approval is required, you review the task, type a reason (a one-liner that flows into the audit log), and click Approve. The system executes. You watch the status move from Pending to Running to Success.

The reason field isn't bureaucracy for its own sake; instead, it's the thread that connects a live production change back to the human decision that authorized it.

The Audit Log: Immutable, attributable, compliance ready

Every action the system takes—every recommendation evaluated, every task executed, every approval recorded—is written to the **Activity Log**. It is time-stamped and immutable.

This matters in two distinct contexts:

- **For your compliance team:** When an auditor asks what changed on LedgerDB last Tuesday and who authorized it, the answer is in the log. Every automated action is attributed to the automation that triggered it. Every approval is attributed to the human who gave it. There's no ambiguity.
- **For your AI agents:** When an external AI agent (your own copilot, Claude via MCP, or another tool) needs to understand what the system has been doing before it takes action, it reads the activity log. The audit trail is also the context window for agentic workflows.

The MCP integration: Same expertise, any AI agent

One final capability worth understanding in depth is this: **The same advisory surface that powers the chatbot and the scorecard is also exposed as an MCP (Model Context Protocol) server.**

This means that an external AI agent, be it Claude, your own internal copilot, or any MCP-compatible tool, can connect to the EDB PG AI console and drive the same Postgres advisor programmatically. An agent can query your cluster's health, retrieve recommendations, read the activity log, and trigger approved actions—all through the same interface, the same expertise, the same guardrails.

This is important because it means the EDB advisory layer becomes composable with the rest of your AI toolchain. You're not choosing between the EDB console and your preferred AI platform. You're using both.

Results

1. Time-to-resolution: Manual vs. agentic

End-to-end effort to detect, diagnose, fix, and verify, by step:

Step	Time
Identify affected queries in `pg_stat_statements`	10–15 min.
Analyze `EXPLAIN ANALYZE` output for each	15–20 min.
Cross-reference with table stats and schema	10–15 min.
Identify the missing index or config change	10–15 min.
Write and validate the fix	5–10 min.
Apply during maintenance window	5 min.
Verify improvement	5 min.
Total	**60–85 min.**

After, with EDB PG AI Console:

Step	Time
Scorecard surfaces the issue and proposes an exact fix	30 sec.
Ask chatbot to explain the recommendation, verify reasoning	2 min.
Approve the index task in Task Manager	1 min.
Apply during maintenance window	5 min.
Total	**8.5 min.**

2. Database-health improvement

In under six minutes, EDB PG AI read the workload, isolated the slowest patterns, and restored the index grade from **D to A** on applying its recommendation.

3. Query performance

The dominant query (42% of total execution time) moved from a sequential scan to an index scan. The planner's estimated cost for that query fell from roughly **87,000 to about 8**, a reduction exceeding 99% for that pattern.

4. Diagnostic speed

Asked in plain language which schema changes would most improve LedgerDB, the chatbot returned a numbered, prioritized set of recommendations, each citing the specific query and statistics behind it, in about 30 seconds. The equivalent manual analysis is estimated at 45–60 minutes of senior DBA time.

What the results mean

The improvement is structural, not marginal. Applying an index takes seconds in either workflow; the cost has always been the investigation around it. By compressing detection and diagnosis from the better part of an hour to a few minutes, and by making the reasoning explicit rather than locked in one expert's head, the agentic workflow changes the economics of day-to-day database operations.

Crucially, it does so across a spectrum of autonomy: The database operates itself if the enterprise deems it appropriate, or it asks first if the enterprise has determined that the stakes are higher.

Compounding effects

- **Expert time is reclaimed.** Hours not spent on routine index investigations move to work that truly needs judgment, including capacity planning, schema design, and migrations.
- **The math multiplies across a fleet.** One DBA can maintain visibility and responsiveness across an estate that previously required a team.
- **Expertise is democratized.** A developer who has never tuned a database can ask why a query is slow and receive a plain-language answer with the exact SQL to fix it.
- **Every change stays governed.** Configurable approval modes let low-risk actions run autonomously while higher-stakes changes wait for sign-off. A visible Task Manager and an immutable Activity Log mean speed never comes at the expense of control or auditability.

Featured use cases

- **Incident response:** Cut slow-query triage from an hour-plus to minutes, with the fix and its rationale produced for you.
- **Proactive tuning:** Continuous A–F grading surfaces index, statistics, and configuration gaps before they page someone.
- **Fleet operations:** Governed autoscale and auto-apply keep many clusters healthy within guardrails you define.
- **Agentic toolchains:** An MCP server exposes the same advisor to external agents, with the same expertise and the same guardrails.

Conclusion

Enterprises standardizing on PostgreSQL gain flexibility and a rich feature set, but getting the best operational performance from a database has always demanded scarce expert time. EDB PG AI's agentic capabilities close the gap between finding a problem and fixing it. In this benchmark, time to resolution for a missing index incident is reduced by roughly 90%, while the cluster's health grade is restored from D to A and every action is governed and logged. The system handles the operational steady state so your experts can focus on the work that genuinely needs them.

Point it at a real cluster. Let EDB PG AI watch your workload for 10 minutes. It will likely find something an hour of manual work would have missed. Contact us to get started.

EDB POSTGRES AI

EDB Postgres AI is the first open, enterprise-grade sovereign data and AI platform, with a secure, compliant, and fully scalable environment, on premises and across clouds. Supported by a global partner network, EDB Postgres AI unifies transactional, analytical, and AI workloads, enabling organizations to operationalize their data and LLMs where, when, and how they need them. For more information, visit enterprisedb.com