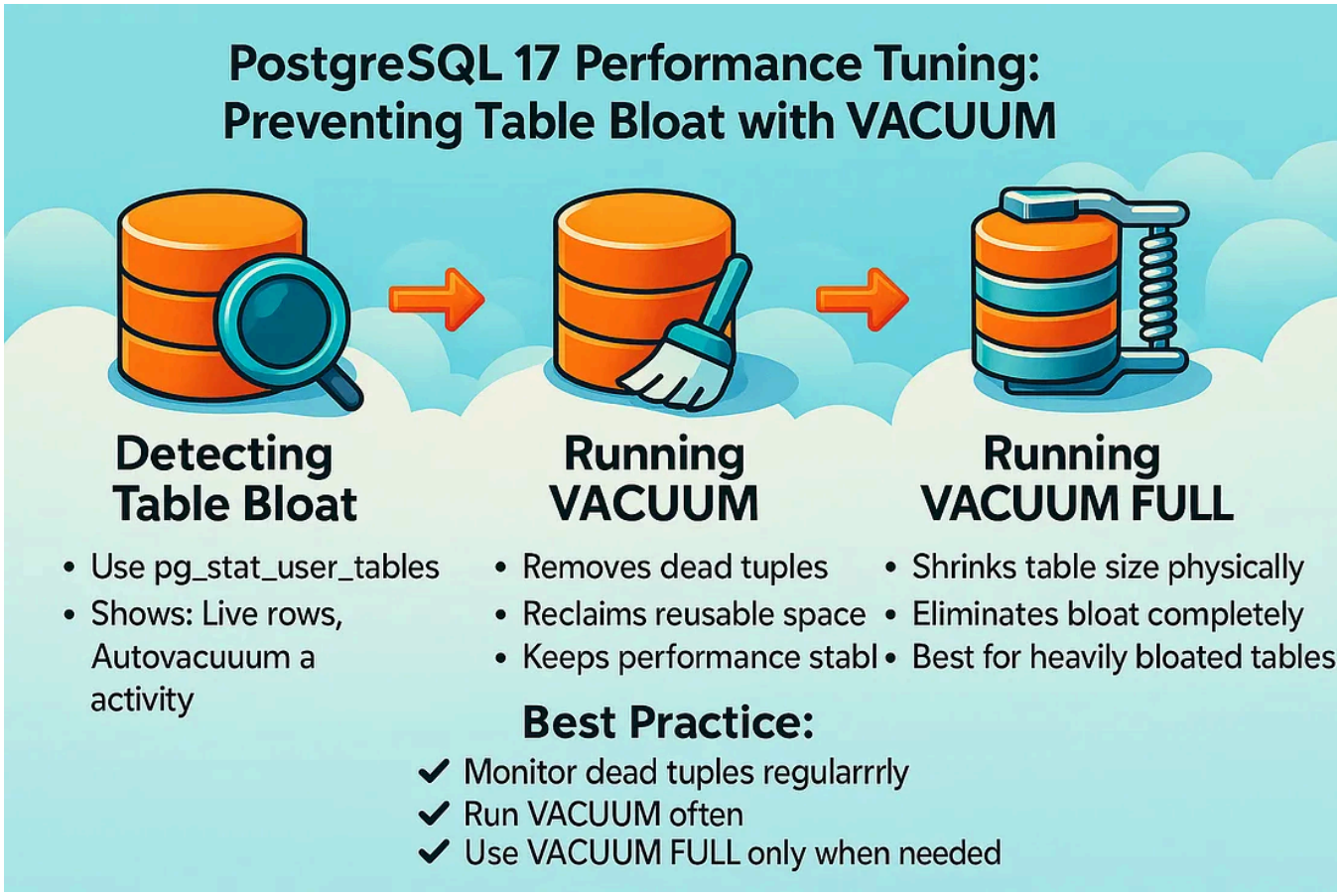
10 - PostgreSQL 17 Performance Tuning: Why Vacuum More Often Prevents Table Bloat

4 min read · Sep 2, 2025

Jeyaram Ayyalusamy Following



One of the most common problems in PostgreSQL is **table bloat** — when dead tuples (old row versions) pile up inside a table and indexes grow unnecessarily large. Many administrators assume VACUUM should be avoided because it is resource-intensive, but the truth is the opposite:

👉 The real solution is to **vacuum more often.**

Running VACUUM regularly means each cycle does less work and keeps tables from growing too large in the first place. Let's see how to detect bloat and why frequent vacuuming matters.

Example: Detecting Table Bloat

To investigate table health, PostgreSQL provides system views such as `pg_stat_user_tables`. This view shows live rows, dead rows, and autovacuum activity.

Here's a query that helps identify tables with high numbers of dead tuples:

```
SELECT
    relname AS table_name,
    n_live_tup AS live_rows,
    n_dead_tup AS dead_rows,
    last_autovacuum,
    last_vacuum
FROM pg_stat_user_tables
ORDER BY n_dead_tup DESC
LIMIT 10;
```

```
postgres=# SELECT
    relname AS table_name,
    n_live_tup AS live_rows,
    n_dead_tup AS dead_rows,
    last_autovacuum,
    last_vacuum
FROM pg_stat_user_tables
ORDER BY n_dead_tup DESC
LIMIT 10;
 table_name | live_rows | dead_rows |        last_autovacuum        | last_vacu
------------+-----------+-----------+-------------------------------+----------
```

```
 orders      |    1948839 |    1051161 | 2025-08-31 19:56:27.311642+00 |
(1 row)

postgres=#
```

👉 In this example:

- The **`orders`** table has **1,051, 161 dead rows**, which is a sign of bloat.
- Autovacuum last ran on `orders` recently, but not frequently enough to prevent dead tuple buildup.

Example: Running VACUUM

If we notice a table with excessive dead tuples, we can run a manual VACUUM:

```
VACUUM orders;
```

```
postgres=# VACUUM orders;
VACUUM
postgres=#
```

```
postgres=# SELECT pg_size_pretty(pg_relation_size('orders'));
 pg_size_pretty
----------------
 219 MB
```

```
(1 row)

postgres=#
```

This will clean up space for reuse but will **not shrink the file size on disk.** Instead, it records free space in the Free Space Map (FSM), so new inserts or updates reuse existing pages.

Example: Running VACUUM FULL

If the table is badly bloated and we need to physically shrink it, we can use:

```
VACUUM FULL orders;
```

```
postgres=# VACUUM FULL orders;
VACUUM
postgres=#
```

```
postgres=# SELECT pg_size_pretty(pg_relation_size('orders'));
 pg_size_pretty
----------------
 142 MB
(1 row)

postgres=#
```

- This rewrites the entire table, removing all dead space.
- The size on disk will shrink.
- But it requires an **exclusive lock**, which means no other queries can write to the table during the process.

Why Autovacuum Matters

To avoid disruptive operations like `VACUUM FULL`, it's best to let **autovacuum** run frequently:

- It keeps dead tuples under control.
- It reduces the risk of severe table bloat.
- It prevents performance drops caused by overly large tables and indexes.

For example, if we notice that `orders` keeps accumulating dead tuples, we can adjust autovacuum for that table:

```
postgres=# WITH src AS (
  SELECT
    (1 + (random()*999999)::BIGINT)             AS customer_id,
    random()                                    AS r,   -- used for status dis
    ROUND((5 + random()*495)::NUMERIC, 2)       AS amount_usd,
    GREATEST(1, (random()*5)::INT)              AS item_count,
    now() - ((random()*730)::INT || ' days')::INTERVAL  AS created_at_rand,
    random()                                    AS ro   -- used for updated_at
  FROM generate_series(1, 3000000) AS gs
)
INSERT INTO orders (customer_id, status, amount_usd, item_count, created_at, up
SELECT
  customer_id,
  CASE
    WHEN r < 0.12 THEN 'PENDING'::order_status
    WHEN r < 0.30 THEN 'PROCESSING'::order_status
    WHEN r < 0.60 THEN 'SHIPPED'::order_status
    WHEN r < 0.95 THEN 'DELIVERED'::order_status
```

```
    ELSE 'CANCELLED'::order_status
  END                                                AS status,
  amount_usd,
  item_count,
  created_at_rand,
  created_at_rand + ((ro*14)::INT || ' days')::INTERVAL
FROM src;
INSERT 0 3000000
postgres=#
postgres=#
```

```
postgres=# SELECT pg_size_pretty(pg_relation_size('orders'));
 pg_size_pretty
----------------
 361 MB
(1 row)

postgres=#
```

```
ALTER TABLE orders
SET (autovacuum_vacuum_scale_factor = 0.05,  -- trigger cleanup at 5% dead rows
     autovacuum_analyze_scale_factor = 0.02); -- update stats more often
```

```
postgres=# ALTER TABLE orders
SET (autovacuum_vacuum_scale_factor = 0.05,  -- trigger cleanup at 5% dead rows
     autovacuum_analyze_scale_factor = 0.02); -- update stats more often
```

```
ALTER TABLE
postgres=#
```

This makes autovacuum run more aggressively for that table, keeping it lean.

```
DELETE FROM orders WHERE status = 'DELIVERED';
```

```
postgres=# DELETE FROM orders WHERE status = 'DELIVERED';
DELETE 1051039
postgres=#
postgres=#
```

```
SELECT pg_size_pretty(pg_relation_size('orders'));
```

```
postgres=# SELECT pg_size_pretty(pg_relation_size('orders'));
 pg_size_pretty
----------------
 361 MB
(1 row)

postgres=#
```

✅ **Key takeaway:**
Avoiding VACUUM creates larger problems. The right approach is to **vacuum more often** — either manually or by tuning **auto vacuum** — so tables stay small, indexes stay efficient, and PostgreSQL performance remains stable.

📢 Stay Updated with Daily PostgreSQL & Cloud Tips!

If you've been finding my blog posts helpful and want to stay ahead with daily insights on PostgreSQL, Cloud Infrastructure, Performance Tuning, and DBA Best Practices — I invite you to subscribe to my Medium account.

🔔 Subscribe here 👉 https://medium.com/@jramcloud1/subscribe

Your support means a lot — and you'll never miss a practical guide again!

🔗 Let's Connect!

If you enjoyed this post or would like to connect professionally, feel free to reach out to me on LinkedIn:
👉 Jeyaram Ayyalusamy

I regularly share content on **PostgreSQL, database administration, cloud technologies, and data engineering.** Always happy to connect, collaborate, and discuss ideas!

Postgresql MySQL Oracle AWS Open Source

Following

Written by Jeyaram Ayyalusamy

158 followers · 2 following

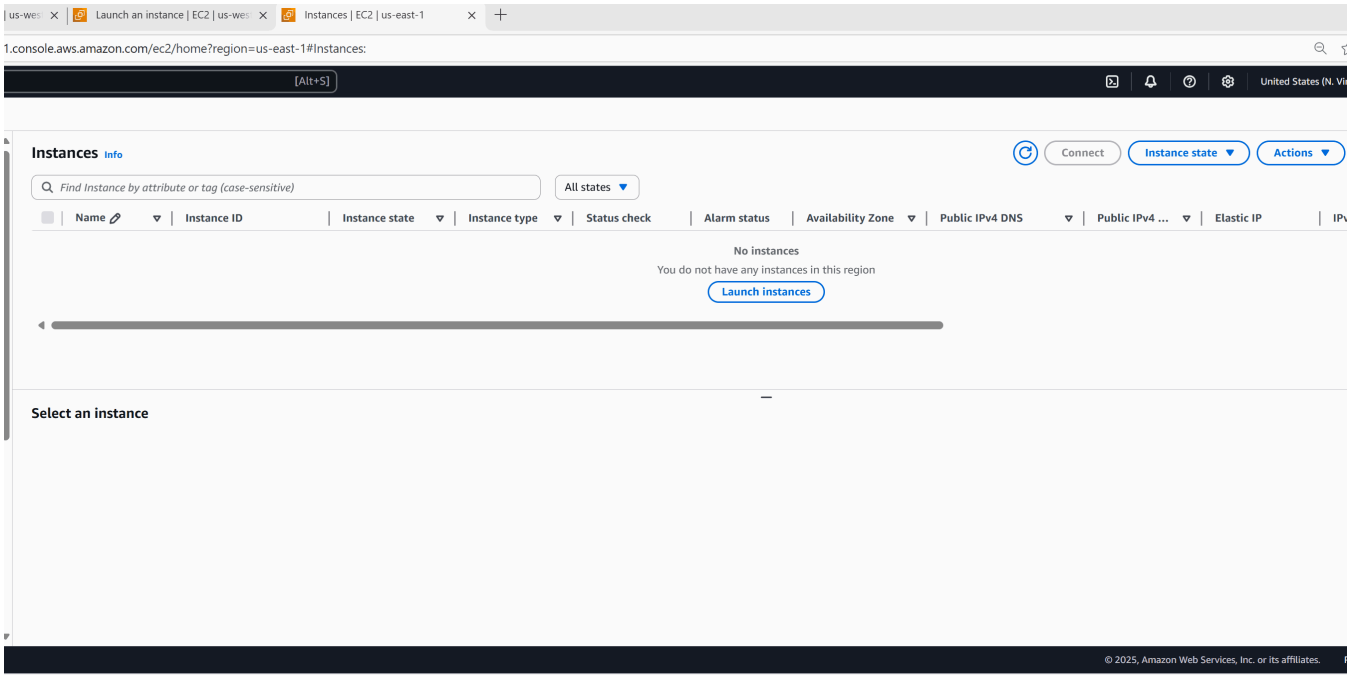
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance Expert

No responses yet

Gvadakte

What are your thoughts?

More from Jeyaram Ayyalusamy

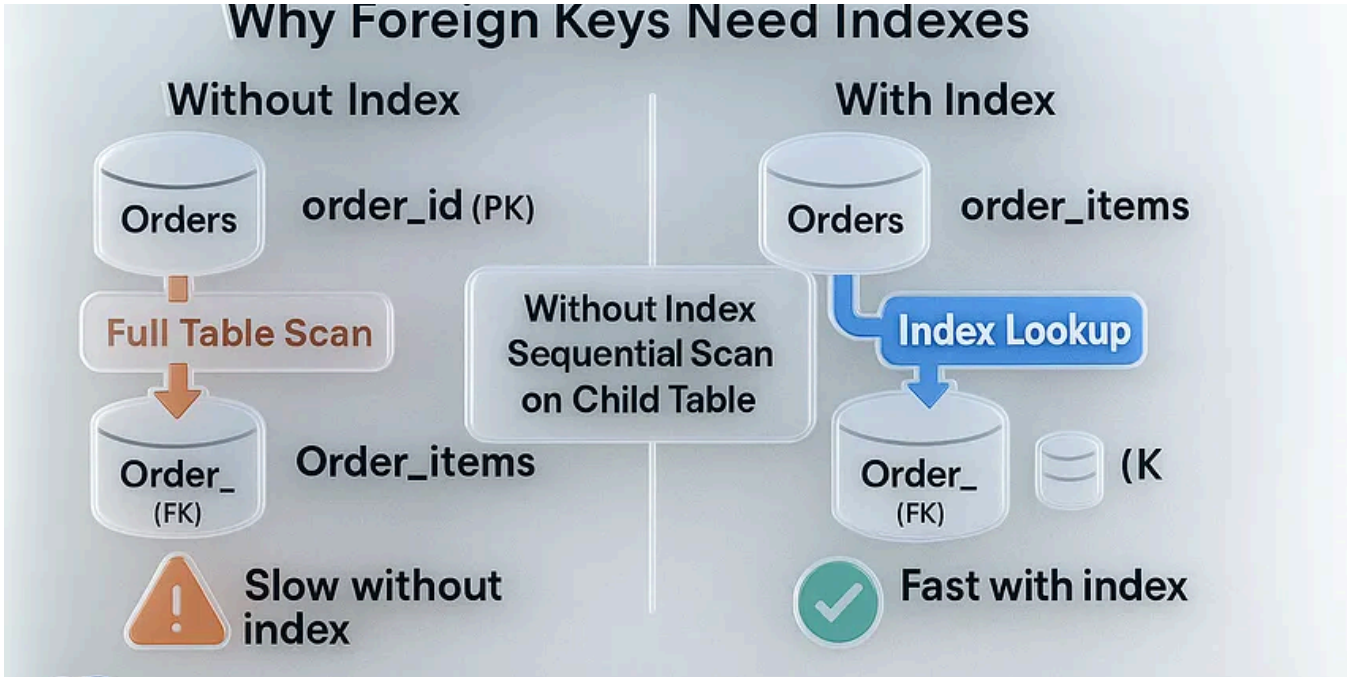


Jeyaram Ayyalusamy

Upgrading PostgreSQL from Version 16 to Version 17 Using pg_upgrade on a Linux Server AWS EC2...

🛠️ A Complete Step-by-Step Guide to Installing PostgreSQL 16 on a Linux Server (With Explanations)

Aug 4 40

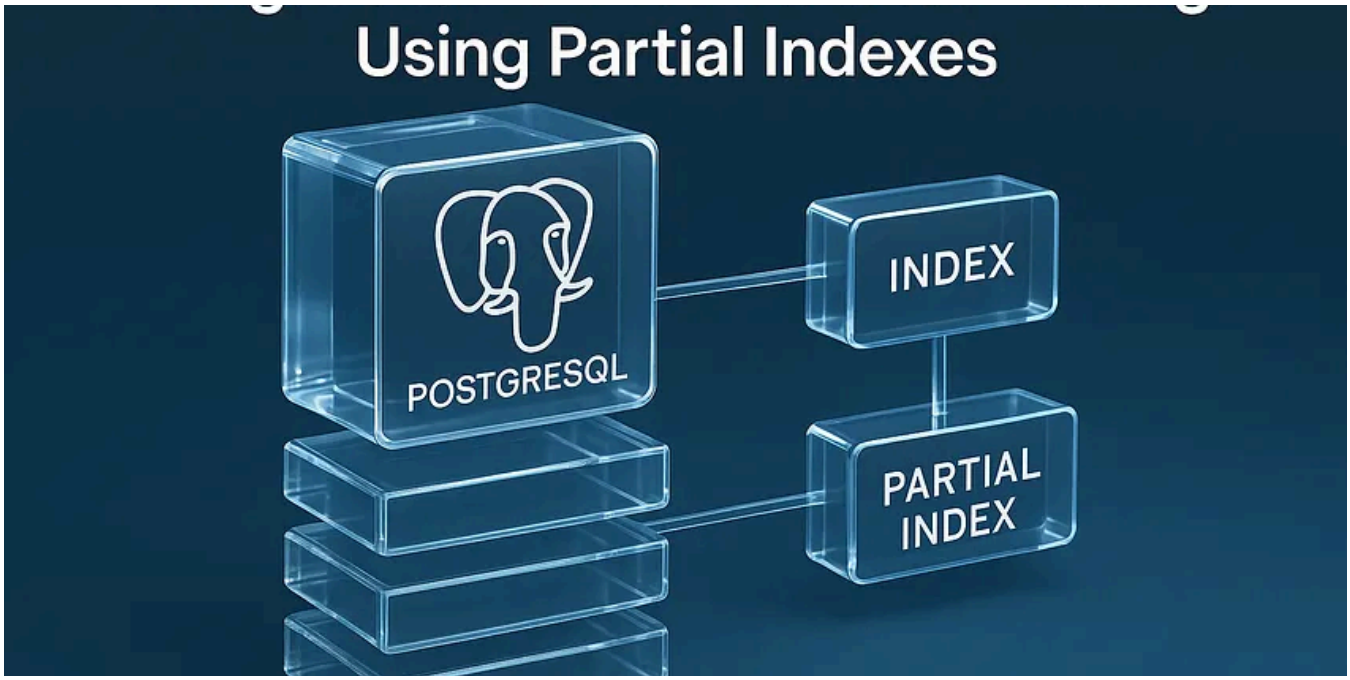


Jeyaram Ayyalusamy

16 -PostgreSQL 17 Performance Tuning: Why Foreign Keys Need Indexes

When designing relational databases, parent-child relationships are common. A classic case is the Orders table (parent) and the Items table…

Sep 3 3 2

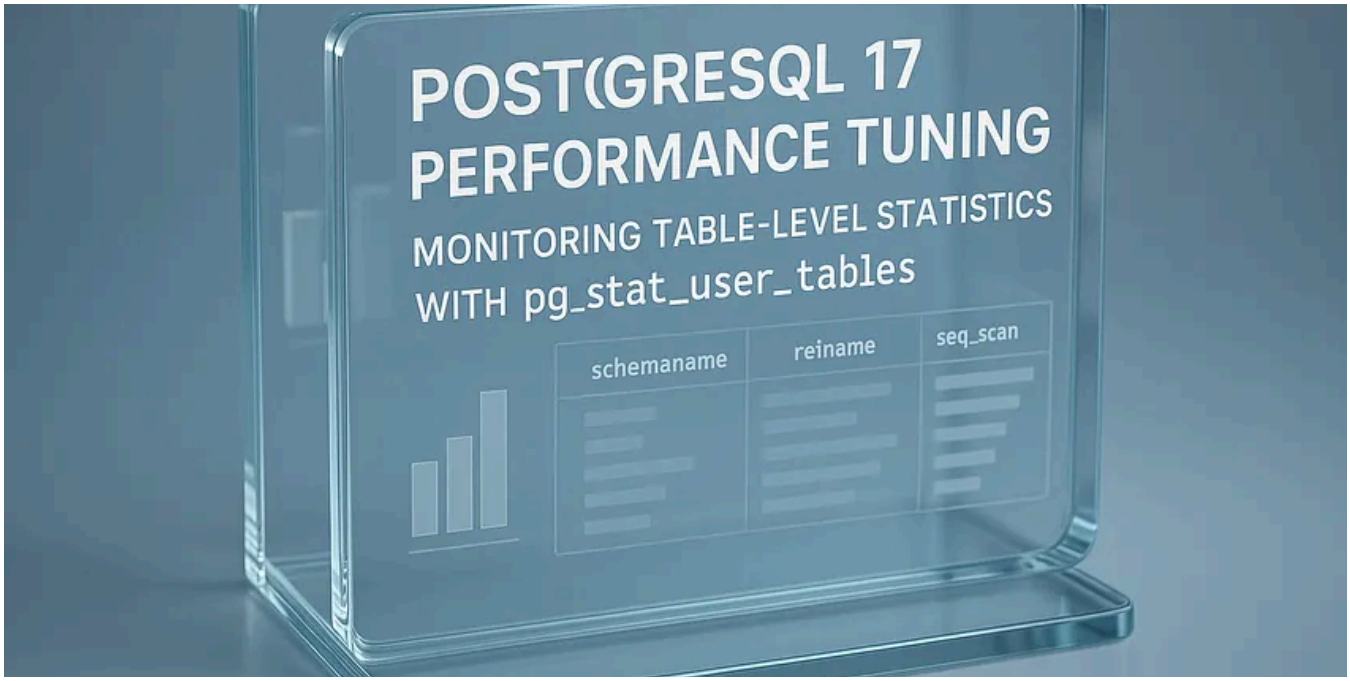


Jeyaram Ayyalusamy

17 - PostgreSQL 17 Performance Tuning: Using Partial Indexes

Even though we usually create indexes on all rows of a table, PostgreSQL also allows you to create an index on just a subset of records...

Sep 4 3



Jeyaram Ayyalusamy

24 - PostgreSQL 17 Performance Tuning: Monitoring Table-Level Statistics with pg_stat_user_tables

When tuning PostgreSQL, one of the most important steps is to observe table-level statistics. You cannot optimize what you cannot measure...

Sep 7 19 1

See all from Jeyaram Ayyalusamy

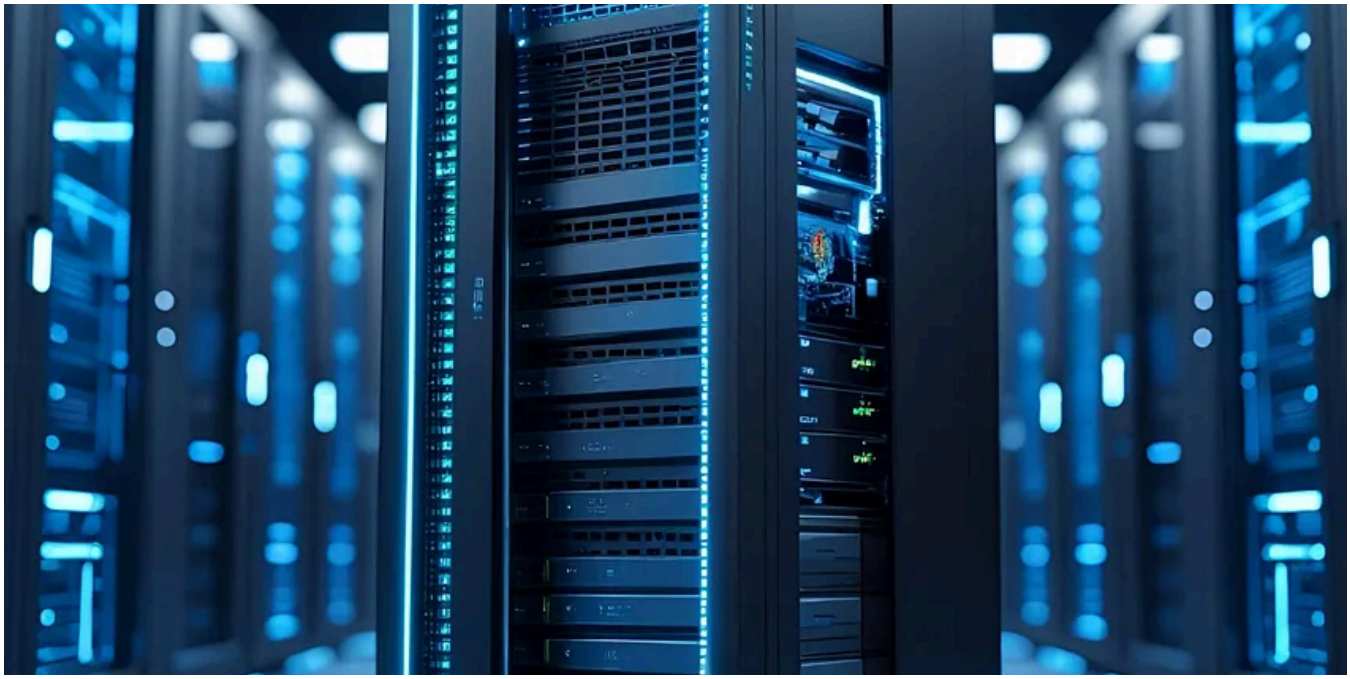
Recommended from Medium

Tomasz Gintowt

Security in PostgreSQL

PostgreSQL is a powerful open-source database. Security is very important when working with sensitive data like passwords, customer...

6d ago 5

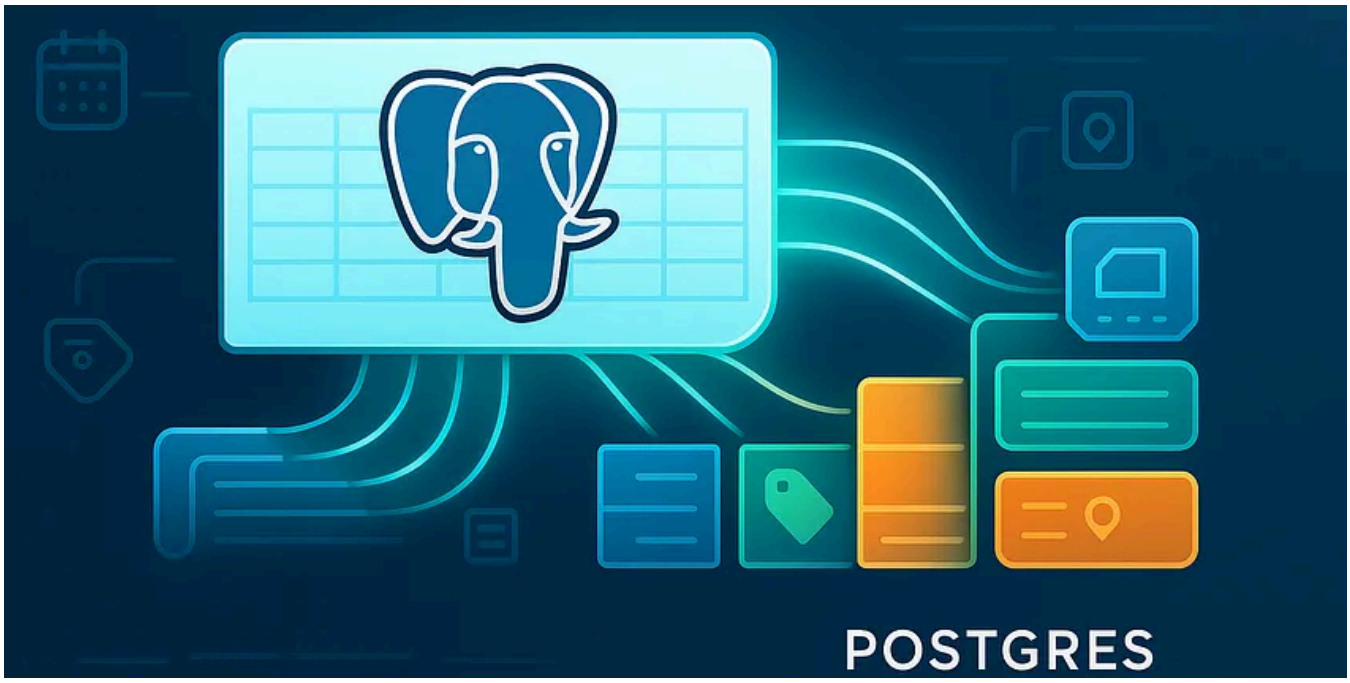


Rizqi Mulki

PostgreSQL Index Bloat: The Silent Killer of Performance

How a 10TB database became 3x faster by fixing the invisible problem that plagues 90% of production PostgreSQL deployments

Sep 15 11 1



Thinking Loop

10 Postgres Partitioning Patterns for Internet-Scale Apps

Keep Postgres fast past a billion rows with real patterns and ready-to-run SQL.

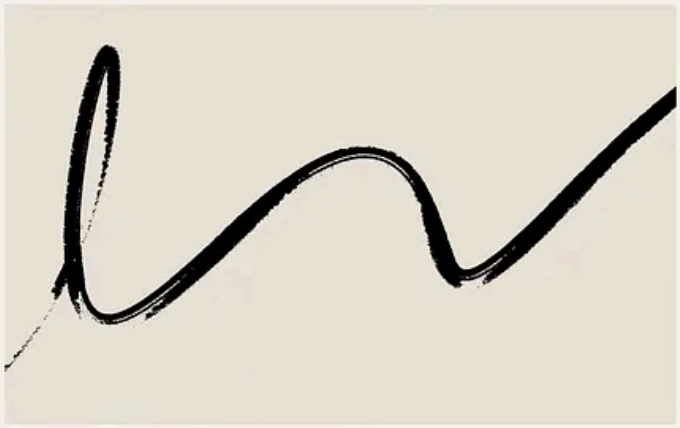
Aug 13 88 2

Vijay Gadhave

Master SQL's QUALIFY Clause for Cleaner, Faster Window Queries

Note: If you're not a medium member, CLICK HERE

May 20 12 1

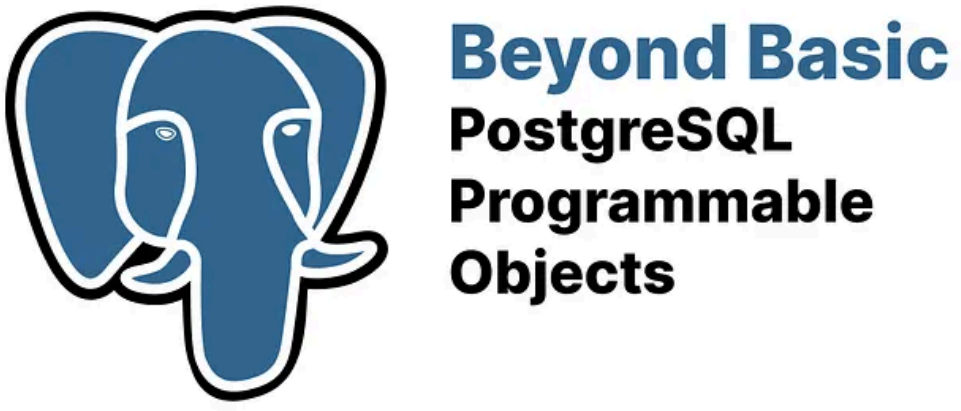


Rohan

JSONB in PostgreSQL: The Fast Lane to Flexible Data Modeling 🚀

"Why spin up yet another database just to store semi-structured data?"—Every engineer who discovered JSONB

Jul 18 12 1



In Stackademic by bektiaw

Beyond Basics: PostgreSQL Programmable Objects (Automate, Optimize, Control)

Tired of repeating the same SQL logic? Learn how PostgreSQL can do the work for us.

Sep 1 68 1