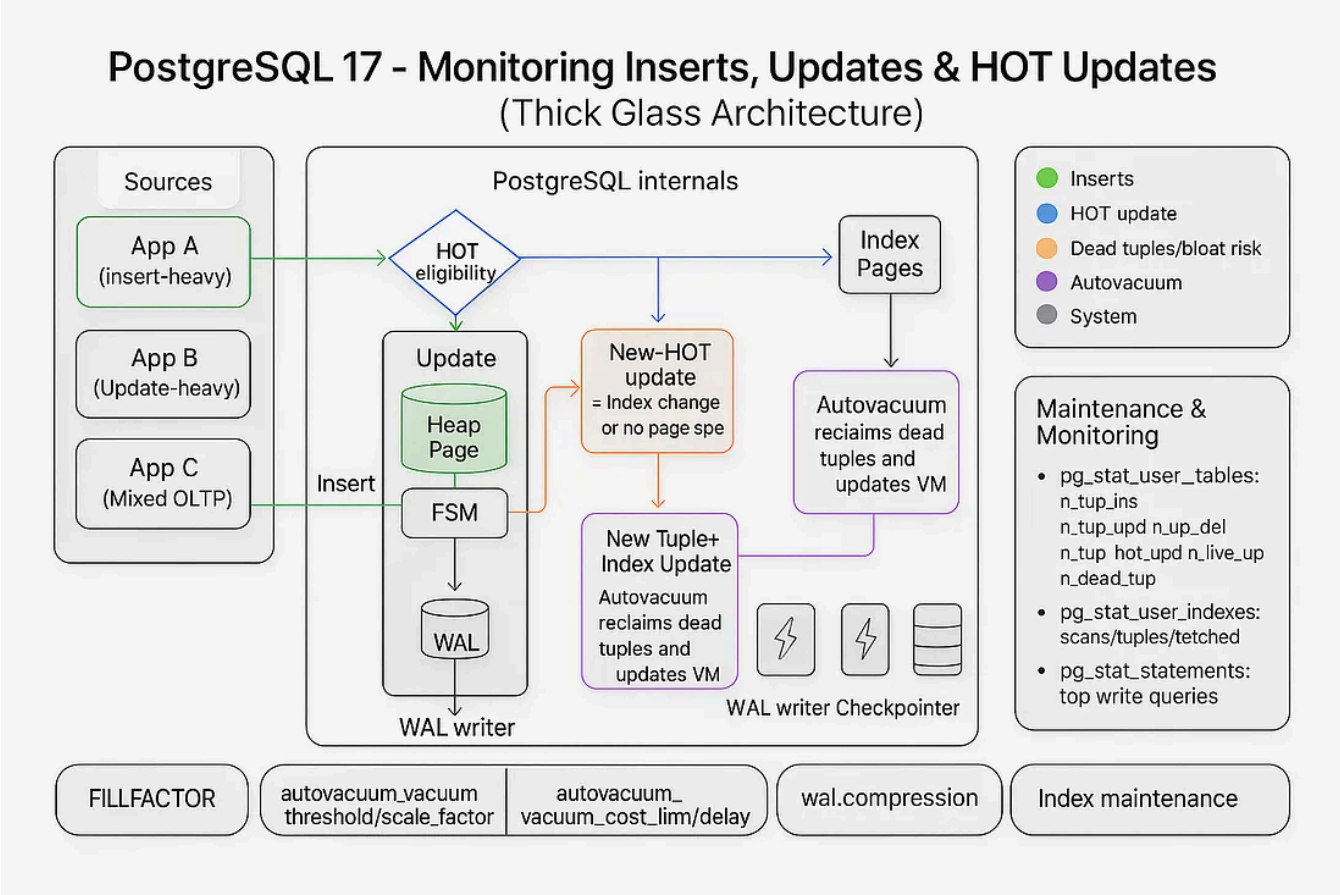
23 - PostgreSQL 17 Performance Tuning: Monitoring Inserts, Updates, and HOT Updates

4 min read · Sep 7, 2025

Jeyaram Ayyalusamy Following



When tuning PostgreSQL, it is very important to understand the **INSERT, UPDATE, and DELETE patterns** of your tables. Different workloads create different storage challenges — for example, some tables may only ever receive inserts, while others may be heavily updated.

By analyzing these patterns, you can decide whether you need to run **reindexing**, **CLUSTER**, or whether you are benefiting from **HOT (Heap-Only Tuple) updates**, which significantly improve update performance.

Step 1: Create a Large Products Table

For this demo, let's work with a `products` table containing 10 million rows:

```
CREATE TABLE products (
    product_id   BIGSERIAL PRIMARY KEY,
    product_name TEXT,
    category     TEXT,
    price        NUMERIC(10,2),
    stock_qty    INT
);
```

```
postgres=# CREATE TABLE products (
    product_id   BIGSERIAL PRIMARY KEY,
    product_name TEXT,
    category     TEXT,
    price        NUMERIC(10,2),
    stock_qty    INT
);
CREATE TABLE
postgres=#
```

```
-- Insert 10 million rows
INSERT INTO products (product_name, category, price, stock_qty)
SELECT
```

```
    'Product_' || g,
    'Category_' || (g % 50),
    (random()*500)::NUMERIC(10,2),
    (random()*100)::INT
FROM generate_series(1, 10000000) g;
ANALYZE products;
```

```
postgres=# -- Insert 10 million rows
INSERT INTO products (product_name, category, price, stock_qty)
SELECT
    'Product_' || g,
    'Category_' || (g % 50),
    (random()*500)::NUMERIC(10,2),
    (random()*100)::INT
FROM generate_series(1, 10000000) g;
ANALYZE products;
INSERT 0 10000000
ANALYZE
postgres=#
```

Step 2: Run Inserts and Updates

We'll simulate different workloads:

```
-- Insert new products (append-only workload)
INSERT INTO products (product_name, category, price, stock_qty)
SELECT
    'New_Product_' || g,
    'Category_' || (g % 50),
    (random()*500)::NUMERIC(10,2),
    (random()*100)::INT
FROM generate_series(1, 100000) g;
```

```
postgres=# -- Insert new products (append-only workload)
INSERT INTO products (product_name, category, price, stock_qty)
SELECT
    'New_Product_' || g,
    'Category_' || (g % 50),
    (random()*500)::NUMERIC(10,2),
    (random()*100)::INT
FROM generate_series(1, 100000) g;
INSERT 0 100000
postgres=#
```

```
-- Update stock quantity (update-heavy workload)
UPDATE products
SET stock_qty = stock_qty + 5
WHERE category = 'Category_10';
```

```
postgres=# -- Update stock quantity (update-heavy workload)
UPDATE products
SET stock_qty = stock_qty + 5
WHERE category = 'Category_10';
UPDATE 202000
postgres=#
```

```
-- Delete some rows (sparse blocks may remain)
DELETE FROM products WHERE category = 'Category_20';
```

```
postgres=# DELETE FROM products WHERE category = 'Category_20';
DELETE 202000
postgres=#
```

Step 3: Inspect Table Modification Stats

Now, let's check how many inserts, updates, and deletes have been performed on `products`.

```
SELECT relname,
       n_tup_ins AS inserts,
       n_tup_upd AS updates,
       n_tup_del AS deletes,
       n_tup_hot_upd AS hot_updates
FROM pg_stat_user_tables
WHERE relname = 'products';
```

📊 Sample Output:

```
postgres=# SELECT relname,
       n_tup_ins AS inserts,
       n_tup_upd AS updates,
       n_tup_del AS deletes,
       n_tup_hot_upd AS hot_updates
FROM pg_stat_user_tables
WHERE relname = 'products';
 relname  | inserts  | updates | deletes | hot_updates
----------+----------+---------+---------+-------------
 products | 10100000 |  404000 |  202000 |           2
```

```
(1 row)

postgres=#
```

Step 4: Interpreting the Results

- **Inserts** → show how many rows were added. Tables that only grow (append-only) are generally simpler to maintain.
- **Updates** → indicate how often rows are modified. Heavy updates increase the chance of **dead tuples**, requiring VACUUM and possibly reindexing.
- **Deletes** → reveal potential for **sparse blocks**, where space is wasted. In such cases, you may consider running `CLUSTER` to fully reorganize the table.
- **HOT Updates** → these are the most interesting metric. HOT (Heap-Only Tuple) updates mean that PostgreSQL was able to update a row **without moving it to a new page**, keeping it in the same block.

👉 A **high percentage of HOT updates** is a very good sign because:

- Updates stay local to the page.
- Fewer index changes are needed.
- Performance is better, especially for update-heavy workloads.

Why HOT Updates Matter

Normally, PostgreSQL creates a new row version for each update, even if only one column changes. This means more dead tuples, more index churn, and more VACUUM overhead.

With HOT updates:

- The row version stays **inside the same page**.

- Index entries don't need to be updated.
- VACUUM has less work to do.

The general rule is:

- **If your workload is update-heavy, aim for a high HOT update ratio.**
- This shows PostgreSQL is efficiently handling updates without bloating indexes.

✅ By monitoring **inserts, updates, deletes, and HOT updates** in `pg_stat_user_tables`, you can clearly see how your workload behaves and adjust your **maintenance strategy** (VACUUM, REINDEX, CLUSTER) accordingly.

📢 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 Oracle Open Source AWS MySQL

Following

Written by Jeyaram Ayyalusamy

158 followers · 2 following

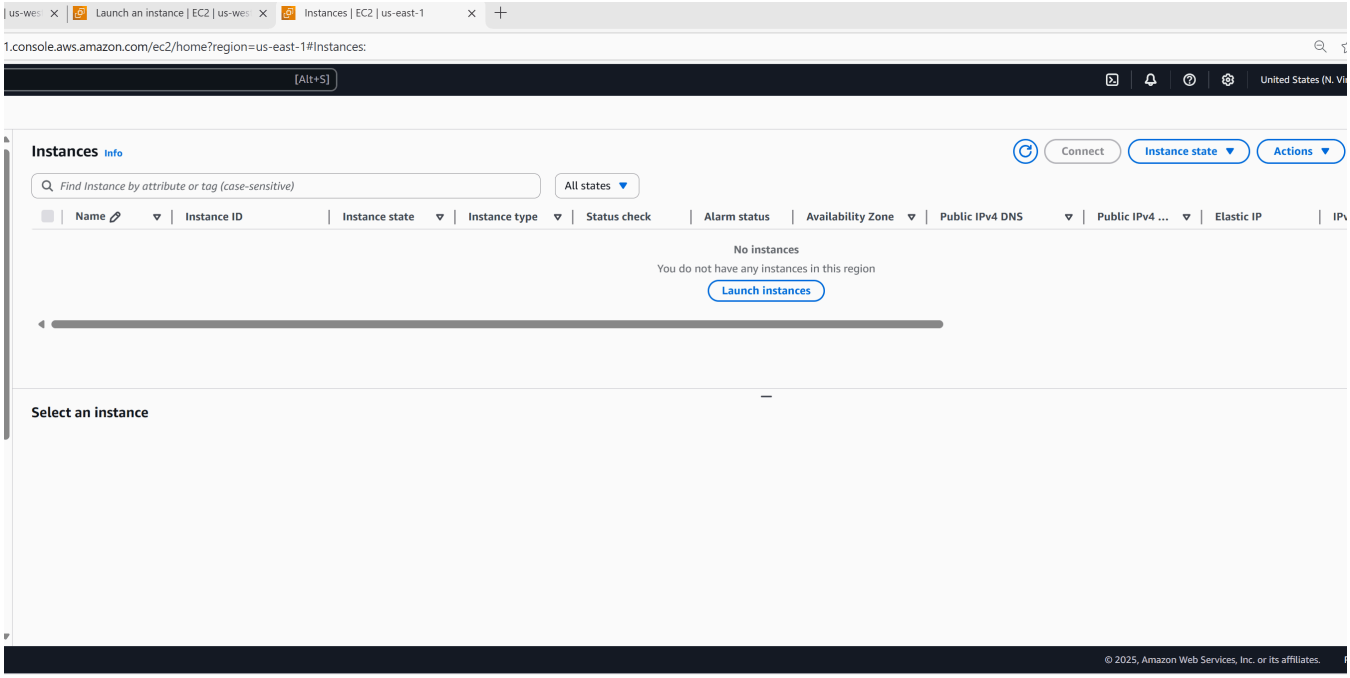
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance Expert

No responses yet

Gvadakte

What are your thoughts?

More from Jeyaram Ayyalusamy

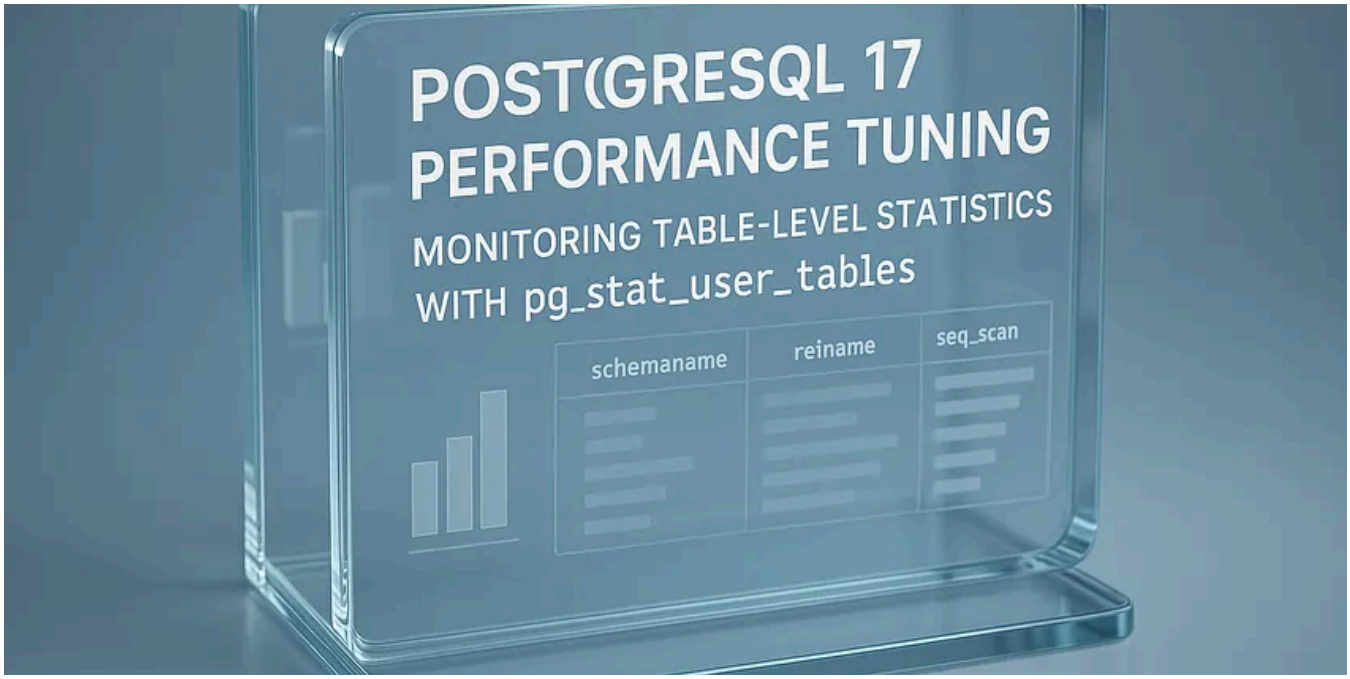


Jeyaram Ayyalusamy

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

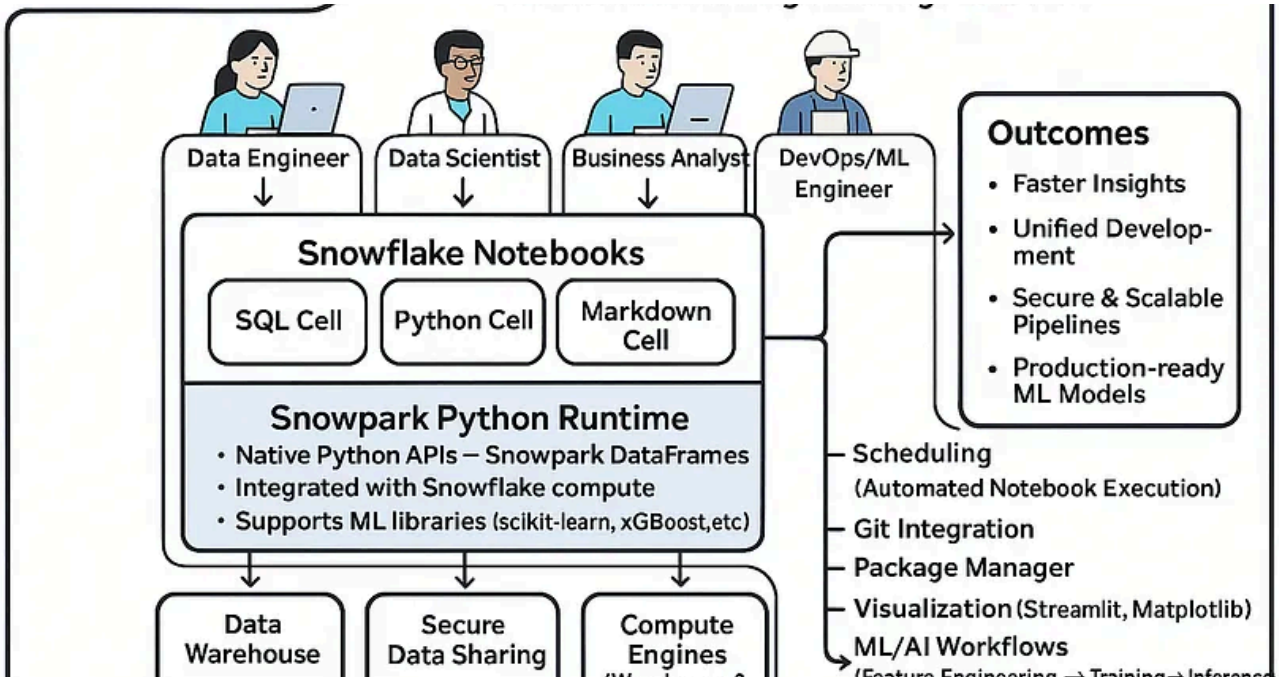


Jeyaram Ayyalusamy

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

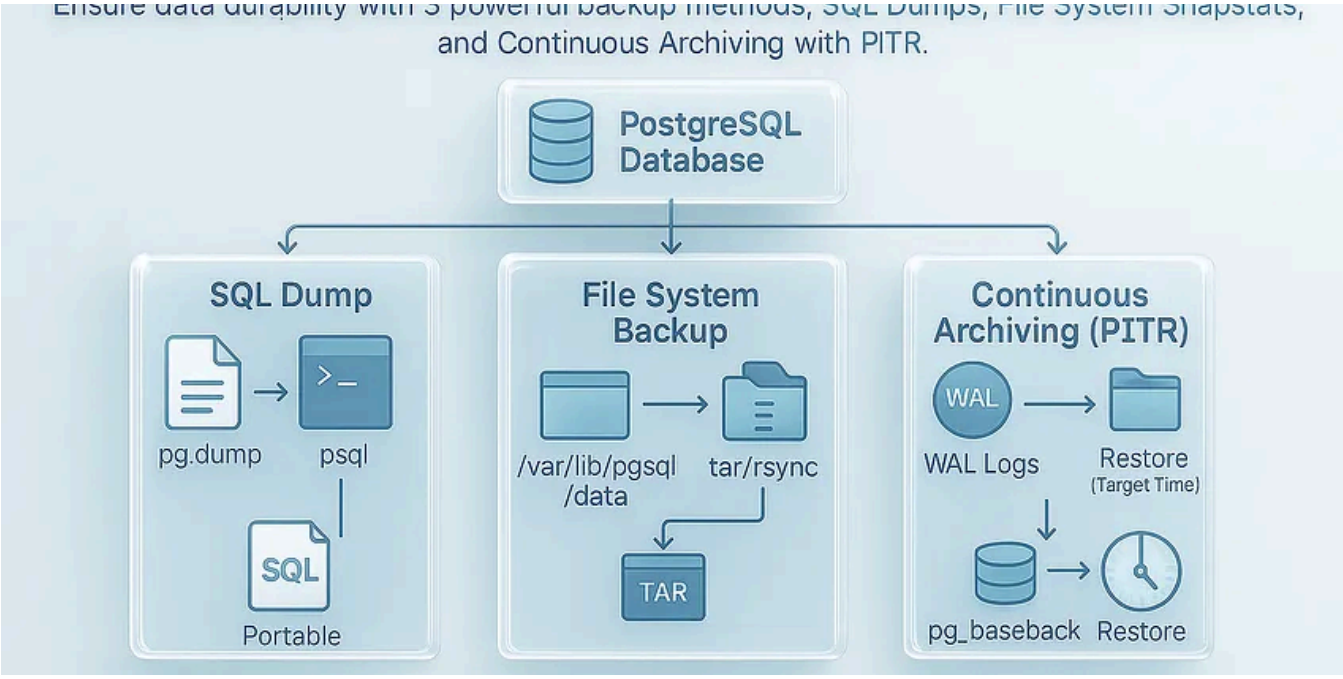


Jeyaram Ayyalusamy

16—Experience Snowflake with Notebooks and Snowpark Python: A Unified Data Engineering Platform

In the fast-moving world of data, organizations are no longer just collecting information—they're leveraging it to drive business…

Jul 13



Jeyaram Ayyalusamy

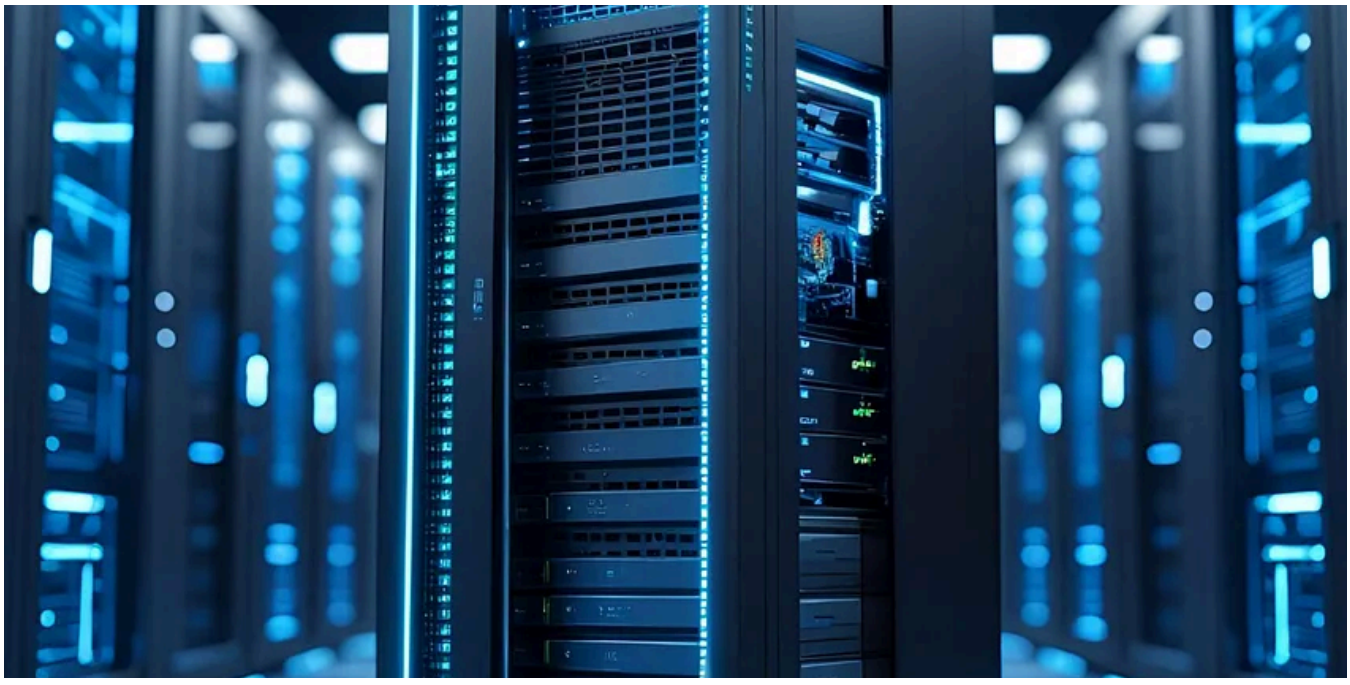
🛡️ Essential Techniques Every DBA Should Know: PostgreSQL Backup Strategies

In the world of databases, data is everything—and losing it can cost your business time, money, and trust. Whether you're managing a...

Aug 20 26

See all from Jeyaram Ayyalusamy

Recommended from Medium



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

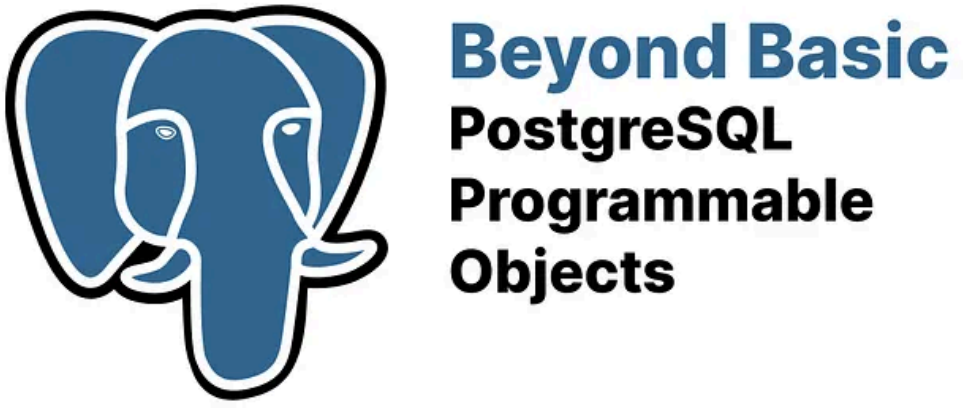
Sep 15 11 1

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

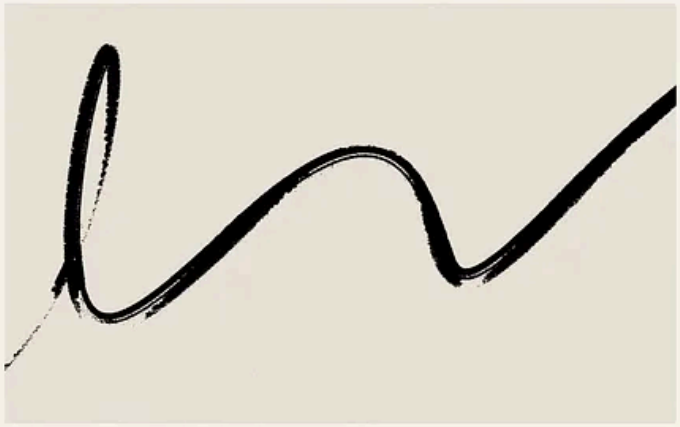


In Stackademic by bektiaw

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

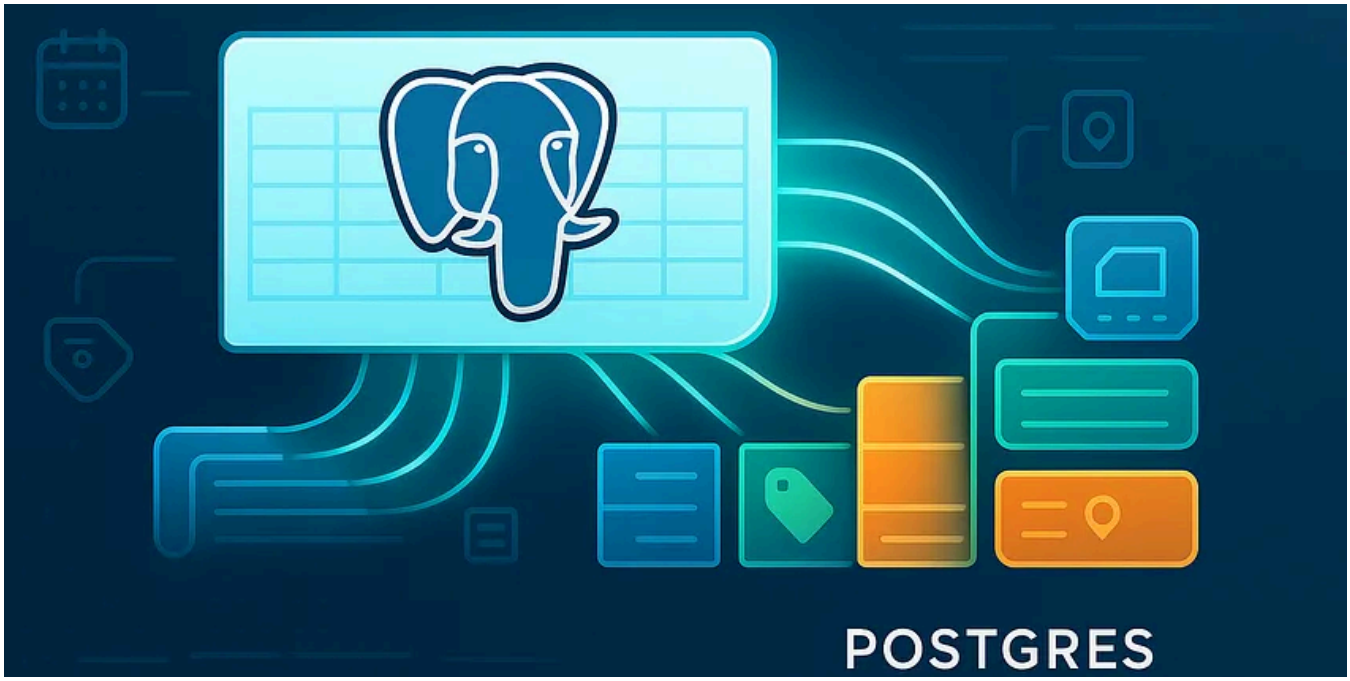


Rohan

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



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

Ajaymaurya

PostgreSQL 18 Features That Change Everything

When a new PostgreSQL release drops, the developer world always pauses. PostgreSQL 18 is no exception—it's bold, innovative, and packed...

Sep 14 19 2

See more recommendations