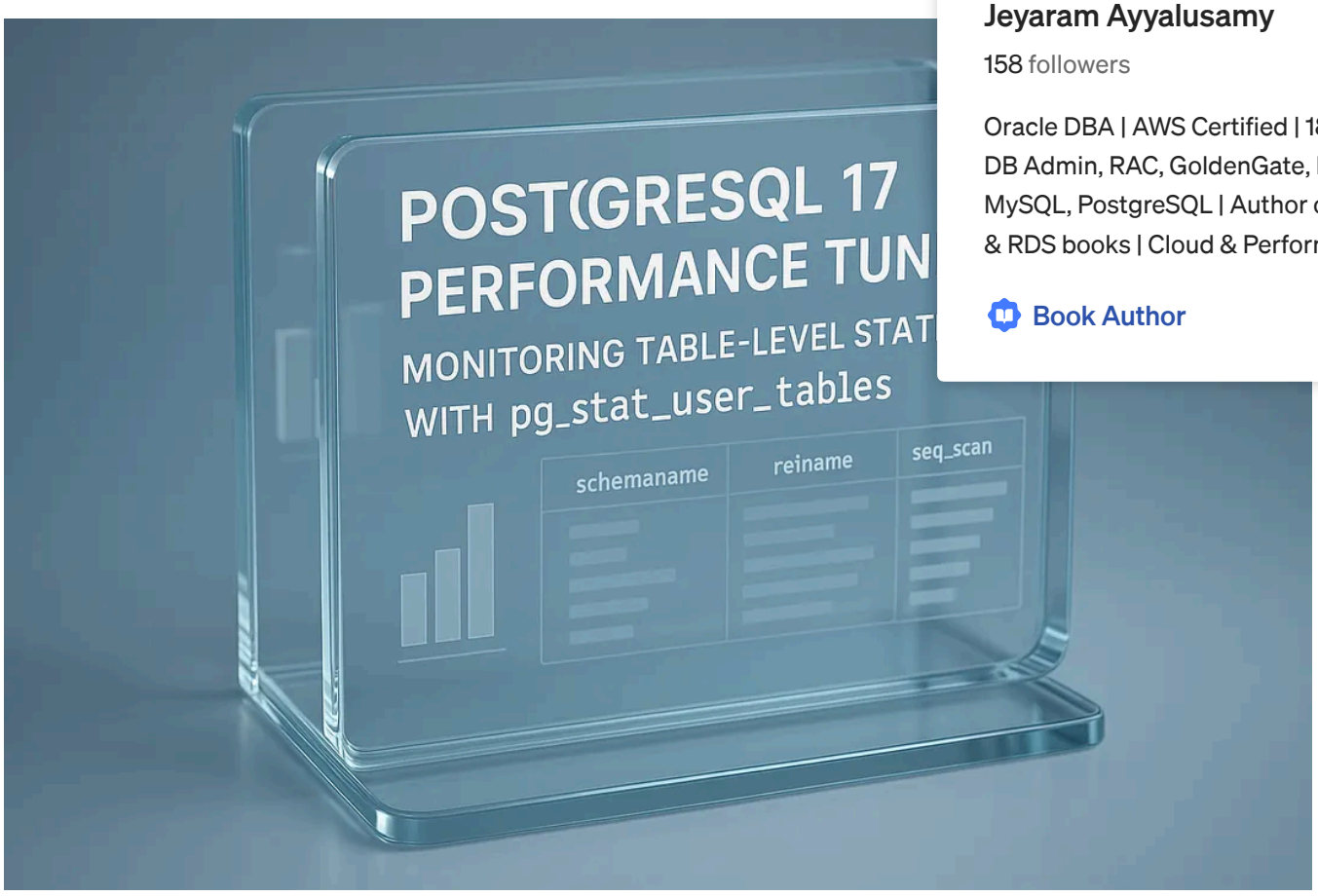
24 - PostgreSQL 17 Performance Tuning: Monitoring Table-Level Statistics with `pg_stat_user_tables`

4 min read · Sep 7, 2025

Jeyaram Ayyalusamy Following

Listen Share More

Jeyaram Ayyalusamy
158 followers
Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar
Book Author



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

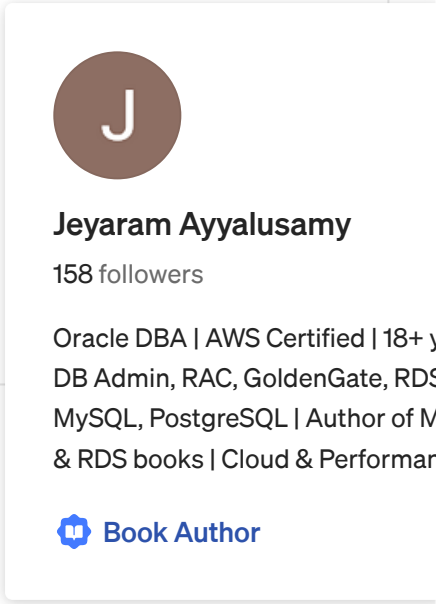
system view **`pg_stat_user_tables`**, which tracks how your tables are accessed, how many tuples (rows) are alive or dead, and when maintenance operations like VACUUM last occurred.

This system view is invaluable for performance tuning — when used properly, it can reveal how your queries and workloads truly interact with your data.

Step 1: Setup Demo with a Products Table

Let's first create a large table (`products`) with multiple columns and insert 10 million rows to simulate a real workload.

```
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),      -- 50 categories
    (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, st
SELECT
    'Product_' || g,
    'Category_' || (g % 50),      -- 50 categories
    (random()*500)::NUMERIC(10,2),
    (random()*100)::INT
FROM generate_series(1, 10000000) g;
ANALYZE products;
INSERT 0 10000000
ANALYZE
postgres=#
```


Now we have a realistic table large enough to observe **sequential scans, index scans, and vacuum behavior.**

Step 2: Generate Some Load

To simulate activity, run queries that update and read from this table:

```
-- Update stock for a specific category
UPDATE products SET stock_qty = stock_qty + 1
```

```
WHERE category = 'Category_10';
```

```
postgres=# -- Update stock for a specific category
UPDATE products SET stock_qty = stock_qty + 1
WHERE category = 'Category_10';
UPDATE 200000
postgres=#
```

```
-- Select by primary key (index scan)
SELECT * FROM products WHERE product_id = 5000000;


-- Select by category (likely sequential scan if no ind
SELECT * FROM products WHERE category = 'Category_20';
```



```
postgres=# SELECT * FROM products WHERE product_id = 5000000;
 product_id |  product_name   |  category  | price  | stock_qty
------------+-----------------+------------+--------+-----------
    5000000 | Product_5000000 | Category_0 | 194.44 |        85
(1 row)

postgres=#
```

```
postgres=# -- Select by category (likely sequential scan if no index exists)
SELECT * FROM products WHERE category = 'Category_2';
```

```
 product_id |   product_name    |  category  | price  | stock_qty
------------+-------------------+------------+--------+-----------
        252 | Product_252       | Category_2 | 236.69 |        73
        302 | Product_302       | Category_2 |   9.44 |        50
        352 | Product_352       | Category_2 | 310.24 |        24
        402 | Product_402       | Category_2 | 175.06 |        52
        452 | Product_452       | Category_2 |  74.66 |        91
        502 | Product_502       | Category_2 |  73.18 |        98
       2402 | Product_2402      | Category_2 |  57.82 |        87
       2452 | Product_2452      | Category_2 | 136.81 |        25
       2502 | Product_2502      | Category_2 | 490.25 |        12
       2552 | Product_2552      | Category_2 | 456.85 |         6
       2602 | Product_2602      | Category_2 | 215.48 |        55
       2652 | Product_2652      | Category_2 | 278.45 |        53
       2702 | Product_2702      | Category_2 | 358.29 |        64
       2752 | Product_2752      | Category_2 | 178.38 |        24
       2802 | Product_2802      | Category_2 |  77.39 |        23
```

Running these repeatedly (or using a tool like `pgbench` for loa
useful statistics in `pg_stat_user_tables`.

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author

Step 3: Inspect `pg_stat_user_tables`

Now, query the system view to see how PostgreSQL is handli

```
SELECT relname,
       n_live_tup,
       n_dead_tup,
       seq_scan,
       seq_tup_read,
       idx_scan,
       idx_tup_fetch,
       last_vacuum,
       last_autovacuum
FROM pg_stat_user_tables
WHERE relname = 'products';
```

Step 4: Interpreting the Results

- `n_live_tup` → current estimate of live rows in the table.
- `n_dead_tup` → number of dead tuples waiting for cleanup (important for monitoring VACUUM effectiveness).
- `seq_scan` / `seq_tup_read` → how many sequential scans occurred and how many rows they processed.
- `idx_scan` / `idx_tup_fetch` → how many index scans occurred and how many rows they fetched.
- `last_vacuum` / `last_autovacuum` → when the table was last vacuumed.

For example, you may observe:

```
postgres=# SELECT relname,
       n_live_tup,
       n_dead_tup,
       seq_scan,
       seq_tup_read,
       idx_scan,
       idx_tup_fetch,
       last_vacuum,
       last_autovacuum
FROM pg_stat_user_tables
WHERE relname = 'products';
 relname  | n_live_tup | n_dead_tup | seq_scan | seq_tup_read | idx_scan | idx_
----------+------------+------------+----------+--------------+----------+-----
 products |    9798007 |     606000 |       10 |     60398000 |        5 |
(1 row)

postgres=#
```


👉 This means:

- Around **50,000 dead tuples** exist, which VACUUM will eventually reclaim.
- Sequential scans processed millions of rows — showing that some queries are likely **not using indexes**.

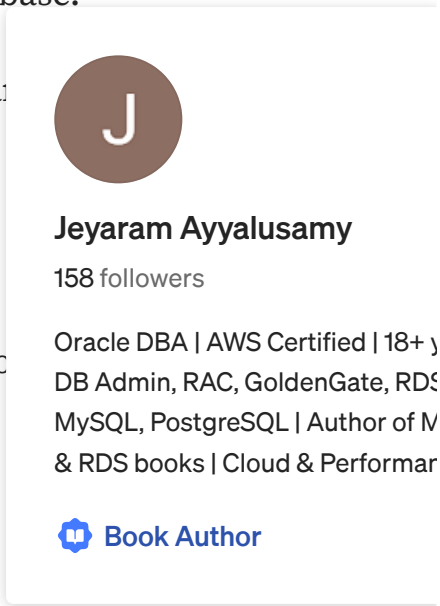
- Index scans fetched 600,000 rows across 6,000 scans — showing frequent lookups by indexed columns.

Why This Matters

- If **dead tuples keep growing** and VACUUM isn't cleaning them efficiently, you risk **table bloat** and performance degradation.
- If **sequential scans dominate**, queries may be missing useful indexes or the planner finds them cheaper due to data distribution.
- Both **number of scans** and **rows fetched** are critical metrics — they reveal not just how queries run, but their total impact on your database.

✅ By continuously monitoring `pg_stat_user_tables`, you ca

- When to VACUUM or ANALYZE more aggressively.
- Which queries need indexing help.
- Whether workloads are efficiently using PostgreSQL's sto decisions.



📢 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 Open Source Oracle MySQL AWS

Following

Written by Jeyaram Ayyalusamy

158 followers · 2 following

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

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author

Responses (1)

Gvadakte

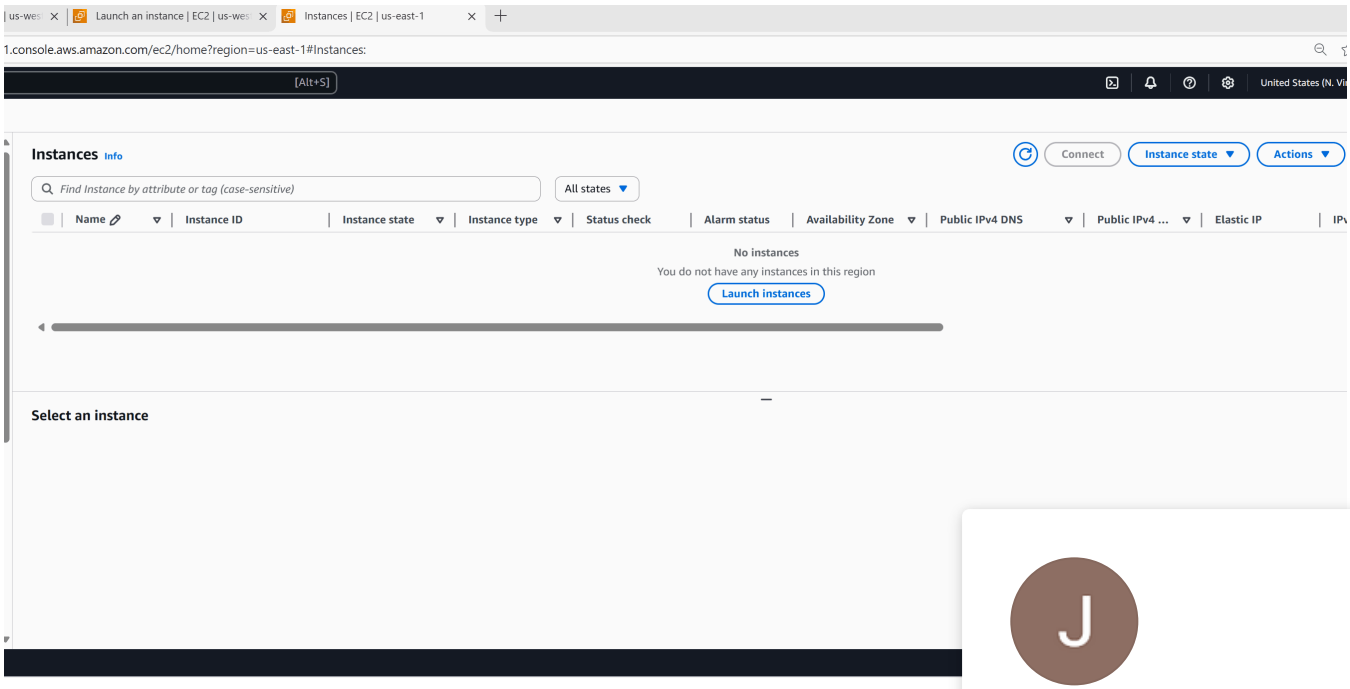
What are your thoughts?

Awsmzn
Sep 14

Very insightful

1 Reply

More from Jeyaram Ayyalusamy



Jeyaram Ayyalusamy

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

🛠️ A Complete Step-by-Step Guide to Installing PostgreSQL 16 on a Explanations)

Aug 4 40

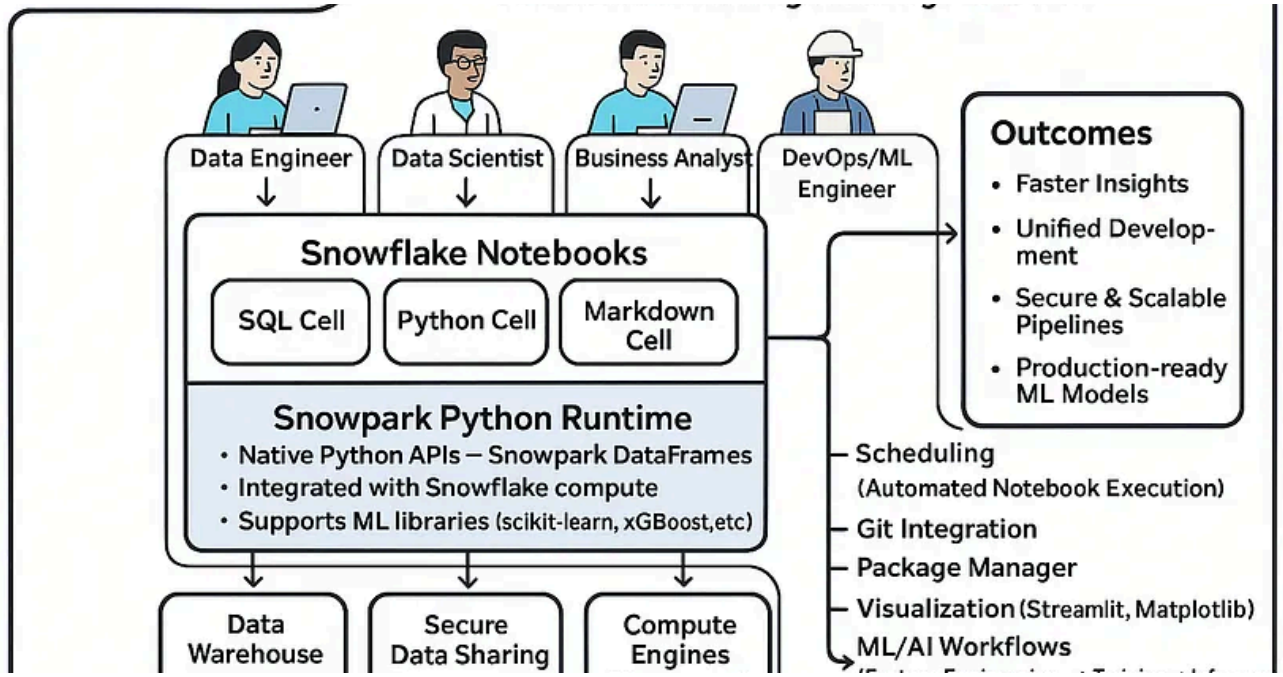
J

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author

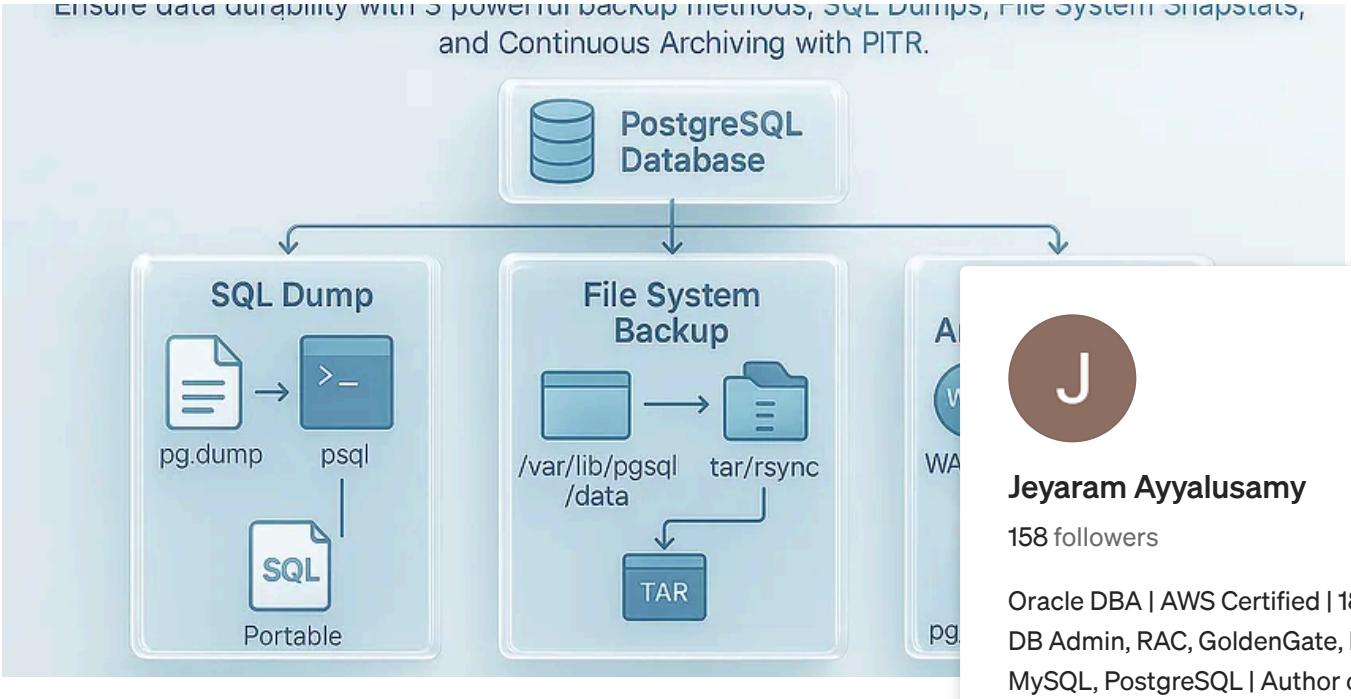


Jeyaram Ayyalusamy

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

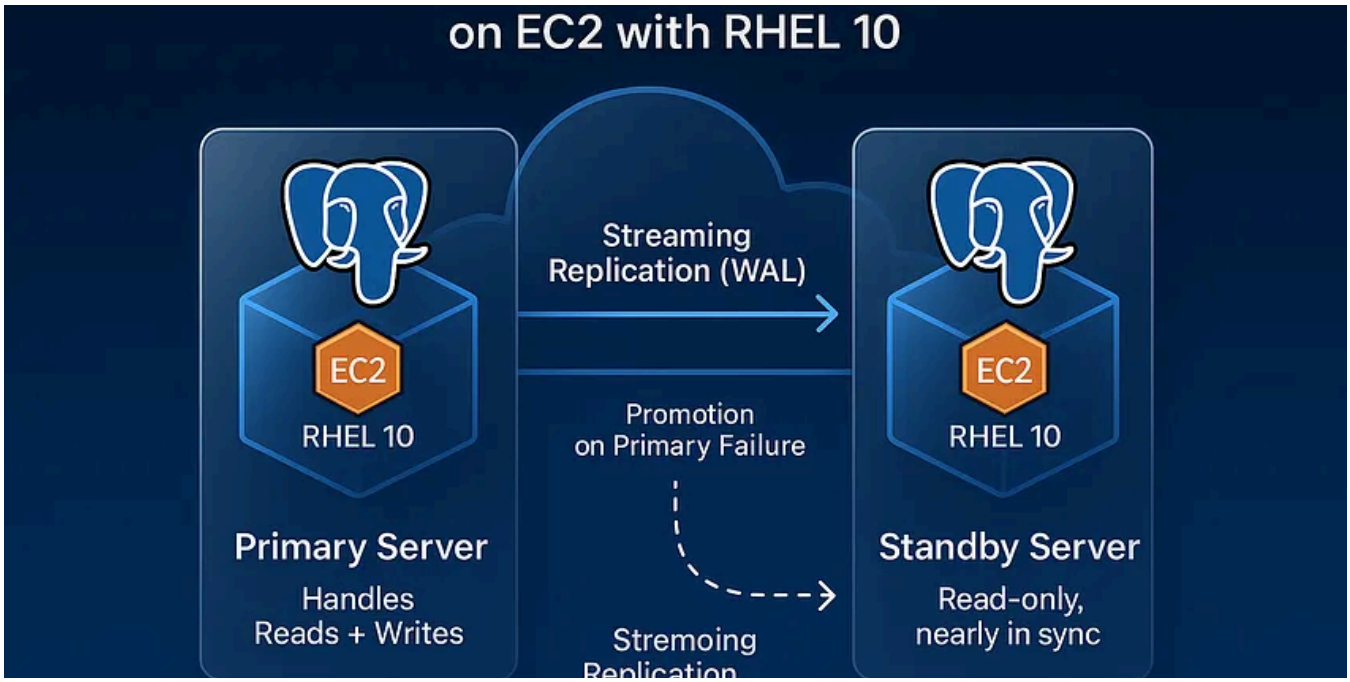


Jeyaram Ayyalusamy

🛡️ Essential Techniques Every DBA Should Know: Po… 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



Jeyaram Ayyalusamy

🚀 Streaming Replication in PostgreSQL 17 on EC2 w
Deep Dive

🐘 Introduction: Why Streaming Replication Matters

Aug 20 👏 50

J

Jeyaram Ayyalusamy

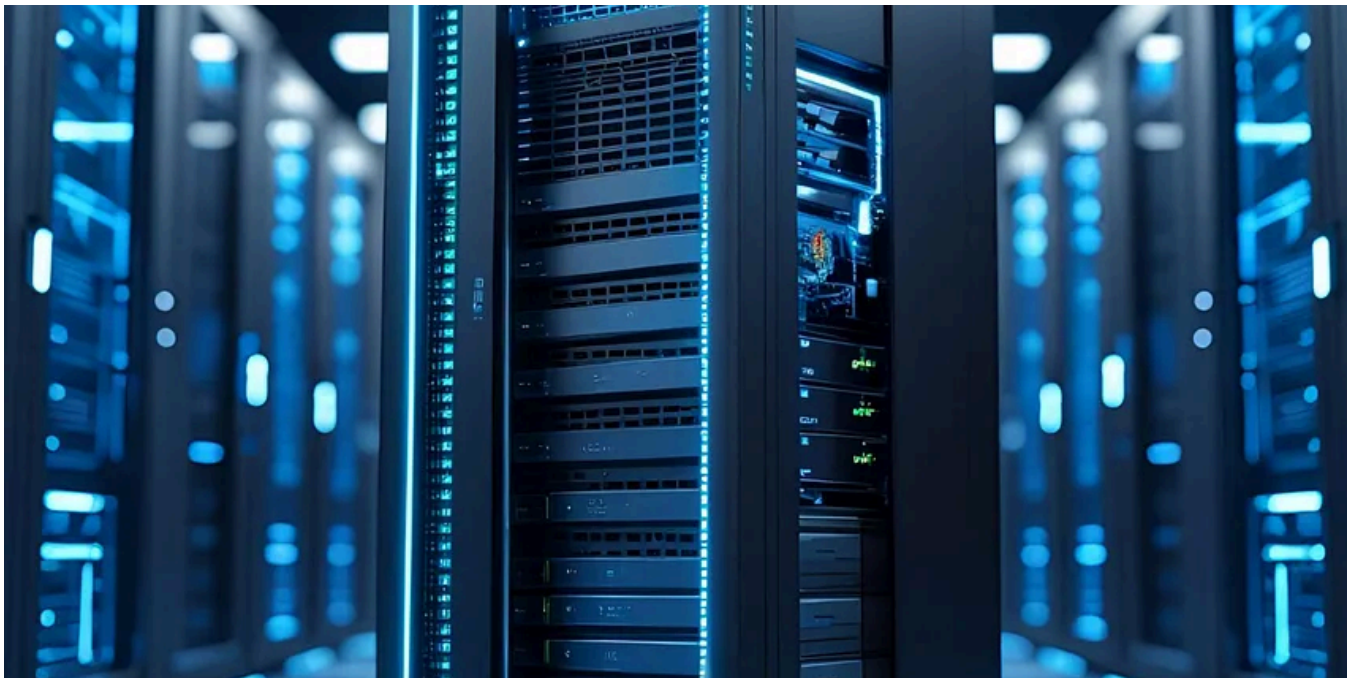
158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author

See all from Jeyaram Ayyalusamy

Recommended from Medium



Rizqi Mulki

PostgreSQL Index Bloat: The Silent Killer of Performa

How a 10TB database became 3x faster by fixing the invisible proble
production PostgreSQL deployments

Sep 15 11 1

J

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

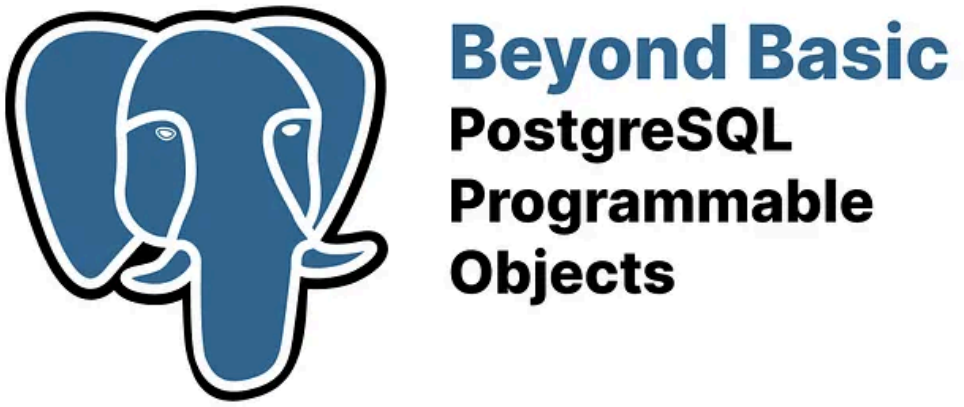
Book Author

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 (
Optimize, Control)

Tired of repeating the same SQL logic? Learn how PostgreSQL can c

Sep 1 68 1

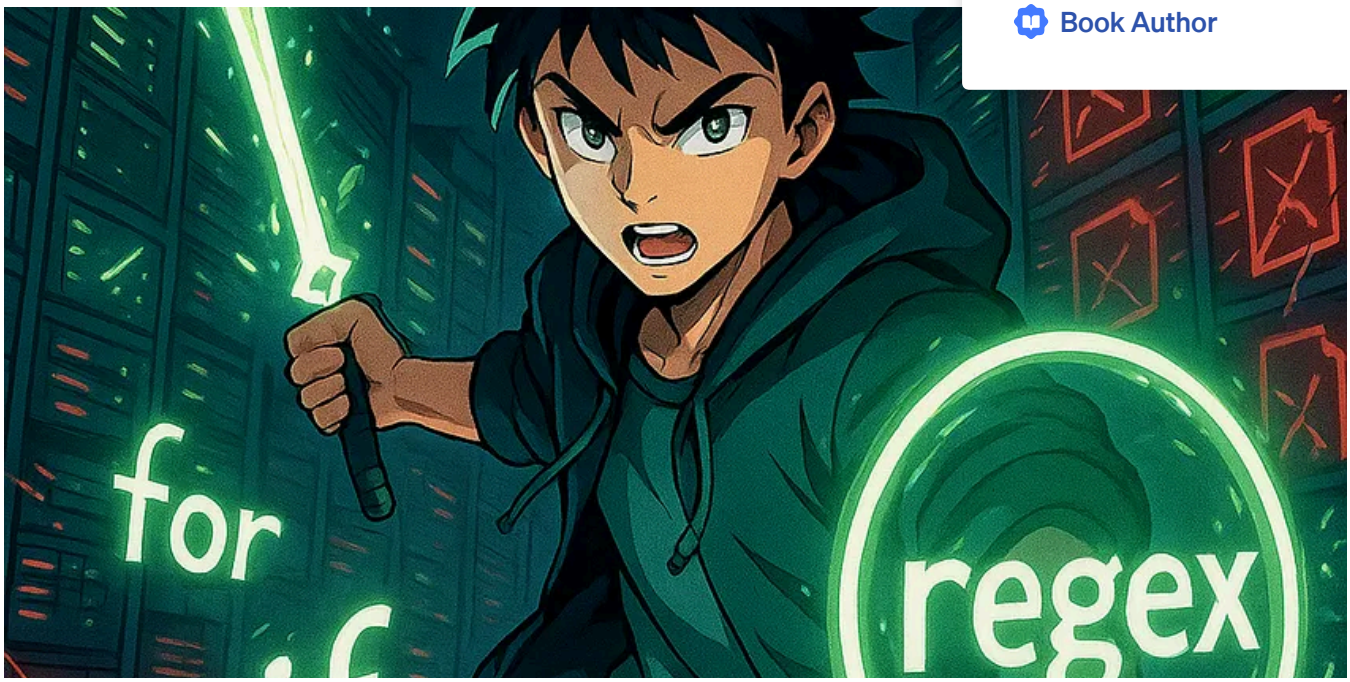
J

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author

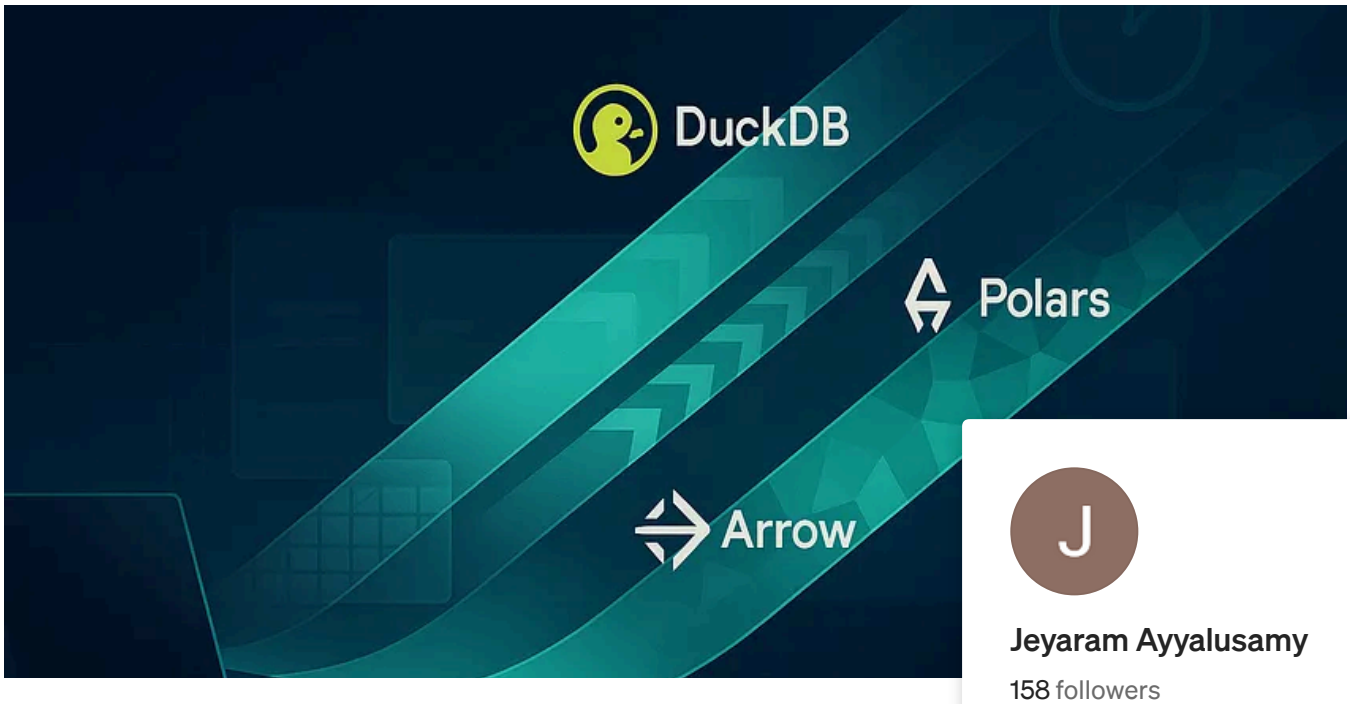


In CodeX by MayhemCode

The Secret Weapons That Separate Bash Beginners from Command Line Legends

Ever wondered how senior developers seem to have magical powers when it comes to bash? They write scripts that validate data, find patterns…

4d ago 7



Thinking Loop

5 DuckDB–Arrow–Polars Workflows in Minutes

Turn day-long pipelines into small, local, reproducible runs without c

5d ago 14

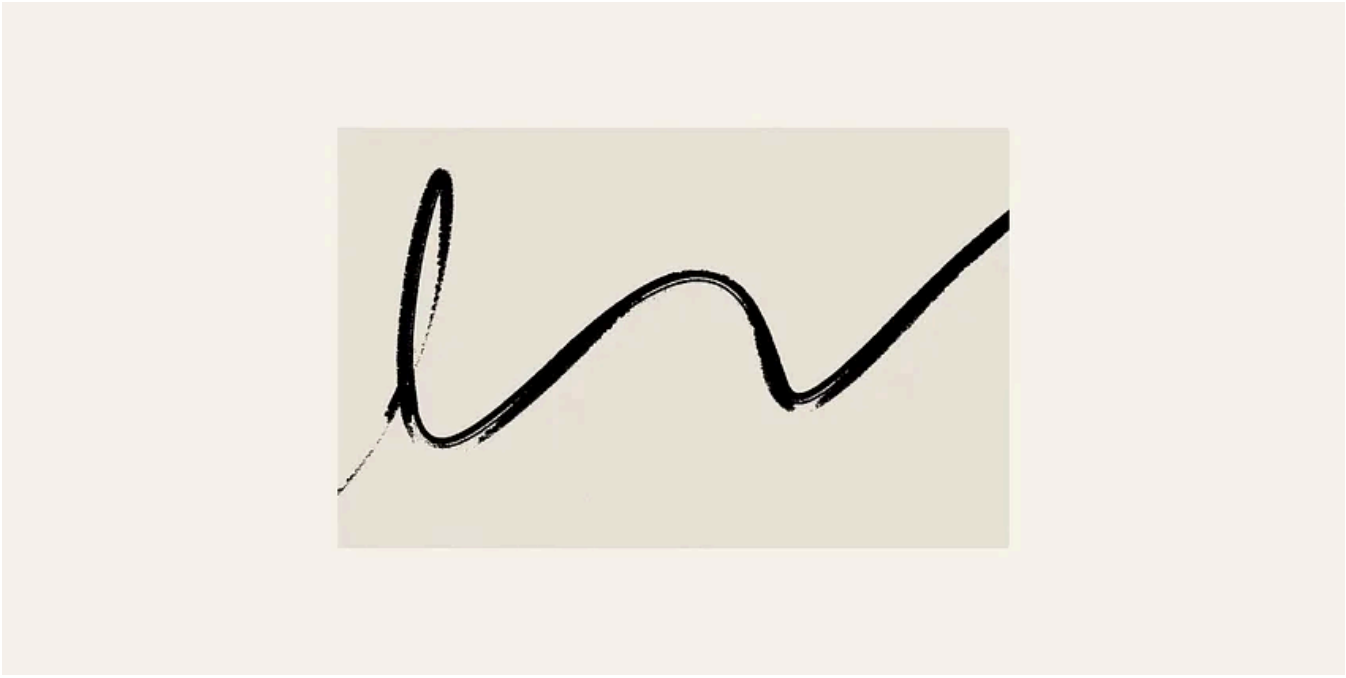
J

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author



R Rohan

JSONB in PostgreSQL: The Fast Lane to Flexible Data

"Why spin up yet another database just to store semi-structured dat
discovered JSONB

Jul 18 12 1

See more recommendations

J

Jeyaram Ayyalusamy

158 followers

Oracle DBA | AWS Certified | 18+ y
DB Admin, RAC, GoldenGate, RDS
MySQL, PostgreSQL | Author of M
& RDS books | Cloud & Performar

Book Author