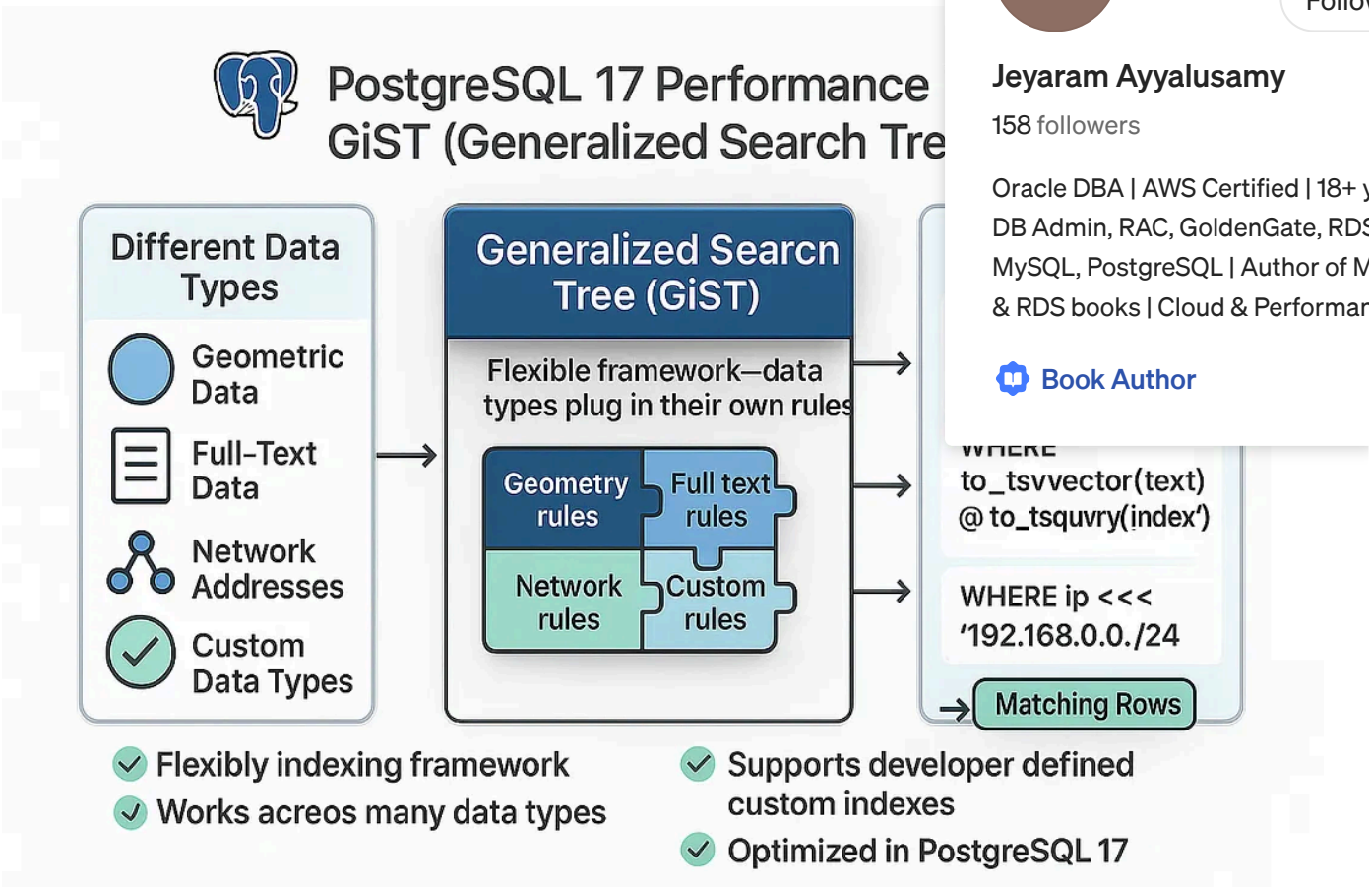
21 - PostgreSQL 17 Performance Tuning: GiST (Generalized Search Tree)

6 min read · Sep 5, 2025

Jeyaram Ayyalusamy Following



GiST (Generalized Search Tree) in PostgreSQL is a flexible indexing framework that allows developers to build indexes for many different types of queries beyond simple equality or range checks. Unlike standard indexes such as B-Tree, GiST is not tied to one specific data type or operation. Instead, it provides a general structure

where different data types can "plug in" their own rules for how indexing and searching should work.

With GiST, PostgreSQL supports advanced features such as:

- **Geometric data** (points, lines, polygons)
- **Full-text search** (via `tsvector`)
- **Network addresses**
- **Custom data types** created by developers

In simple terms, GiST is like a toolkit for building specialized indexes. It doesn't define how data should be stored or compared itself, but provides a framework so that each data type can define its own methods for indexing and searching efficiently.

👉 Example use cases: spatial queries (e.g., "find points with
text search queries.

GiST indexes speed up queries on data with **spatial/geometri**
`point`, `box`, `circle`, `polygon`) and also enable **nearest-neigh**
with the `<->` distance operator. GiST is also used for several
and certain full-text configurations.

Example (pattern):

```
CREATE INDEX idx_locations_gist ON locations USING gist (geolocation);
```

Step 1: Create the `locations` table (point geometry)

```
CREATE TABLE locations (
    location_id  BIGSERIAL PRIMARY KEY,
    name         TEXT,
```

```
    geolocation  POINT          -- (x, y) = (longitude, latitude) for this demc
);
```

```
postgres=# CREATE TABLE locations (
    location_id  BIGSERIAL PRIMARY KEY,
    name         TEXT,
    geolocation  POINT          -- (x, y) = (longitude, latitude) for this demc
);
CREATE TABLE
postgres=#
```

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 2: Insert 10,000,000 random points (global spre

```
-- 10M random points: longitude [-180,180], latitude [-
INSERT INTO locations (name, geolocation)
SELECT
  'Location_' || g,
  POINT( -180 + random()*360,     -- lon
         -90  + random()*180 )    -- lat
FROM generate_series(1, 10000000) AS g;
```

```
postgres=# INSERT INTO locations (name, geolocation)
SELECT
  'Location_' || g,
  POINT( -180 + random()*360,     -- lon
         -90  + random()*180 )    -- lat
```

```
FROM generate_series(1, 10000000) AS g;
INSERT 0 10000000
postgres=#
```

```
INSERT 0 10000000
postgres=#
```

Step 3: ANALYZE after bulk load

```
ANALYZE locations;
```

```
postgres=# ANALYZE locations;
ANALYZE
postgres=#
```


Step 4: Baseline query without any index (bounding box)

Find all points inside a rough **NYC** bounding box (lon/lat rectangle).
Operator: `point <@ box` → "point is contained by box".

```
EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ box '((-74.1,40.6),(-73.7,40.9))';
```

```
postgres=# EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ box '((-74.1,40.6),(-73.7,40.9))';
                                                              QUERY PLAN
---------------------------------------------------------------------------------------
 Gather  (cost=1000.00..146529.33 rows=10000 width=40) (actual time=1781.469..5
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on locations  (cost=0.00..144529.33 rows=4167 width=40
         Filter: (geolocation <@ '(-73.7,40.9),(-74.1,40.6)'::box)
         Rows Removed by Filter: 3333329
 Planning Time: 0.143 ms
 Execution Time: 5997.030 ms
(8 rows)

postgres=#
```

```
Time: 5997.030 ms (00:05.997)
postgres=#
```


Without an index, PostgreSQL scans the entire 10M-row table.

Step 5: Create a GiST index on `geolocation`

```
postgres=# CREATE INDEX idx_locations_gist ON locations USING gist (geolocation
CREATE INDEX
postgres=#
```

> This adds a spatial index that understands geometric operators and supports **K-NN** ordering (`ORDER BY geolocation <-> point`).

ANALYZE created index

```
ANALYZE locations;
```

```
postgres=# ANALYZE locations;
ANALYZE
postgres=#
```


Step 6: Re-run the bounding box query (now indexed)

```
EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ box '((-74.1,40.6),(-73.7,40.9))';
```

```
postgres=# EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ box '((-74.1,40.6),(-73.7,40.9))';
                                                        QUERY PLAN
-----------------------------------------------------------------------------------
 Bitmap Heap Scan on locations  (cost=405.93..29364.11 rows=10001 width=40) (ac
```

```
   Recheck Cond: (geolocation <@ '(-73.7,40.9),(-74.1,40.6)'::box)
   Heap Blocks: exact=13
   ->  Bitmap Index Scan on idx_locations_gist  (cost=0.00..403.43 rows=10001 w
         Index Cond: (geolocation <@ '(-73.7,40.9),(-74.1,40.6)'::box)
 Planning Time: 1.132 ms
 Execution Time: 10.241 ms
(7 rows)

postgres=#
```

```
Time: 10.241 ms
postgres=#
```

*The plan switches to a **Bitmap Index Scan** using GiST, dropping latency from ~5.997 s to ~10.241 ms.*

Step 7: Radius search with circles (point-in-circle)

Find all points within an approximate radius around Times S

Operator: `point <@ circle` → "point is inside circle".

> **Note:** Built-in geometric types treat lon/lat as planar; distanc
> meters. This is fine for demos; for accurate geodesic distanc

```
-- Times Square center ≈ (-73.9855, 40.7580)
-- Radius here is 0.02 degrees (~2.2 km in latitude for demo purposes)

postgres=# EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ circle '((-73.9855,40.7580), 0.02)';
```

```
postgres=#
postgres=#
postgres=# EXPLAIN ANALYZE
SELECT *
FROM locations
WHERE geolocation <@ circle '((-73.9855,40.7580), 0.02)';
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------
 Bitmap Heap Scan on locations  (cost=405.93..29364.11 rows=10001 width=40) (ac
   Recheck Cond: (geolocation <@ '<(-73.9855,40.758),0.02>'::circle)
   ->  Bitmap Index Scan on idx_locations_gist  (cost=0.00..403.43 rows=10001 w
         Index Cond: (geolocation <@ '<(-73.9855,40.758),0.02>'::circle)
 Planning Time: 0.074 ms
 Execution Time: 0.041 ms
(6 rows)

postgres=
```

Step 8: K-NN nearest-neighbor with `<->` (fast top-K)

Return the **100 closest** points to Times Square:

```
EXPLAIN ANALYZE
SELECT location_id, name, geolocation
FROM locations
ORDER BY geolocation <-> POINT(-73.9855, 40.7580)
LIMIT 100;
```

```
ppostgres=# EXPLAIN ANALYZE
SELECT location_id, name, geolocation
FROM locations
ORDER BY geolocation <-> POINT(-73.9855, 40.7580)
```

```
LIMIT 100;
                                                                    QUERY PLAN
-------------------------------------------------------------------------------
 Limit  (cost=0.42..9.37 rows=100 width=48) (actual time=0.062..40.138 rows=100
   ->  Index Scan using idx_locations_gist on locations  (cost=0.42..895059.72
         Order By: (geolocation <-> '(-73.9855,40.758)'::point)
 Planning Time: 2.480 ms
 Execution Time: 40.240 ms
(5 rows)

postgres=
```

*GiST performs a **K-NN index scan**: the index returns rows in distance order — no full scan or sort over 10M rows.*

Step 9: Polygon containment (point-in-polygon)

Operator: `point <@ polygon`.

```
-- Rough 4-vertex polygon around midtown Manhattan (ill
postgres=# EXPLAIN ANALYZE
SELECT count(*)
FROM locations
WHERE geolocation <@ polygon '((-74.02,40.70),(-73.94,4
```



```
postgres=#
postgres=# EXPLAIN ANALYZE
SELECT count(*)
FROM locations
WHERE geolocation <@ polygon '((-74.02,40.70),(-73.94,40.70),(-73.94,40.80),(-7
                                                                    QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=528.44..528.45 rows=1 width=8) (actual time=0.028..0.030 rows
   ->  Index Only Scan using idx_locations_gist on locations  (cost=0.42..503.4
```

```
          Index Cond: (geolocation <@ '((-74.02,40.7),(-73.94,40.7),(-73.94,40.8
          Heap Fetches: 0
 Planning Time: 1.248 ms
 Execution Time: 0.050 ms
(6 rows)

postgres=#
```

Step 10: Verify index usage & size (quick checks)

```
postgres=# -- Is the index being used?
SELECT indexrelid::regclass AS index_name,
       idx_scan, idx_tup_read, idx_tup_fetch
FROM pg_stat_user_indexes
WHERE relid = 'locations'::regclass
ORDER BY idx_scan DESC;
```



```
postgres=# SELECT indexrelid::regclass AS index_name,
       idx_scan, idx_tup_read, idx_tup_fetch
FROM pg_stat_user_indexes
WHERE relid = 'locations'::regclass
ORDER BY idx_scan DESC;
     index_name     | idx_scan | idx_tup_read | idx_tup_fetch
--------------------+----------+--------------+---------------
 idx_locations_gist |        4 |          113 |           100
 locations_pkey     |        0 |            0 |             0
(2 rows)

postgres=#
```

```
-- Table vs. indexes size
SELECT
  pg_size_pretty(pg_relation_size('locations'))        AS table_only,
  pg_size_pretty(pg_indexes_size('locations'))         AS indexes_only,
  pg_size_pretty(pg_total_relation_size('locations'))  AS total_with_indexes;
```

```
postgres=# SELECT
  pg_size_pretty(pg_relation_size('locations'))
  pg_size_pretty(pg_indexes_size('locations'))
  pg_size_pretty(pg_total_relation_size('locations'))
 table_only | indexes_only | total_with_indexes
------------+--------------+--------------------
 722 MB     | 850 MB       | 1572 MB
(1 row)

postgres=#
```



```
postgres=# -- Individual index size
SELECT
  indexrelid::regclass AS index_name,
  pg_size_pretty(pg_relation_size(indexrelid)) AS index_size
FROM pg_index
WHERE indrelid = 'locations'::regclass
ORDER BY pg_relation_size(indexrelid) DESC;
```

```
postgres=# SELECT
  indexrelid::regclass AS index_name,
  pg_size_pretty(pg_relation_size(indexrelid)) AS index_size
FROM pg_index
WHERE indrelid = 'locations'::regclass
ORDER BY pg_relation_size(indexrelid) DESC;
     index_name     | index_size
--------------------+------------
 idx_locations_gist | 635 MB
 locations_pkey     | 214 MB
(2 rows)

postgres=#
```

Step 11: Notes specific to GiST (kept focused)

- **CLUSTER is not supported on GiST.** GiST has no natural
  `... USING gist_index` is not allowed. (Use B-tree if you ne
- **K-NN requires** `<->` and a GiST operator class that suppor
  datatype (`point_ops` does).
- **ANALYZE** keeps selectivity estimates accurate. After larg
  `ANALYZE locations;`
- For **online index builds** on production:

```
CREATE INDEX CONCURRENTLY idx_locations_gist ON locations USING gist (geolocati
```

Summary

- **Workload:** 10M points in `locations(geolocation POINT)` .

- **Before GiST:** Parallel Seq Scan for spatial filters (seconds).
- **After GiST:** Bitmap Index Scans for **box/circle/polygon** (tens of ms) and **K-NN** index scans for **nearest-neighbor** (single-digit ms for top-K).
- **PostgreSQL 17:** Same syntax, mature GiST operator classes for built-in geometry.

📢 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.

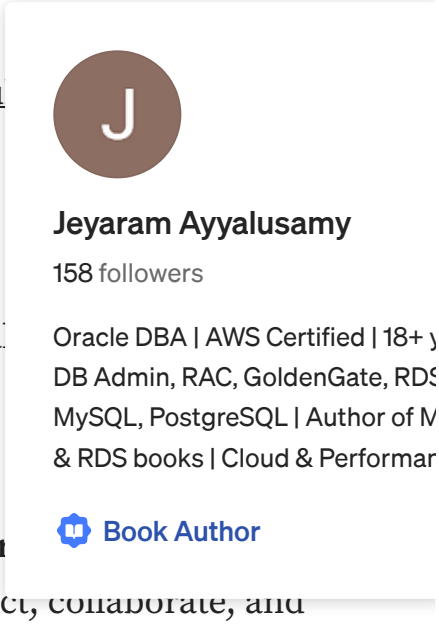
🔔 Subscribe here 👉 https://medium.com/@jramcloud1/su

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

🔗 Let's Connect!

If you enjoyed this post or would like to connect professiona
to me on LinkedIn:
👉 Jeyaram Ayyalusamy

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



Postgresql AWS Open Source MySQL Mongodb

J

Following

Written by Jeyaram Ayyalusamy

158 followers · 2 following

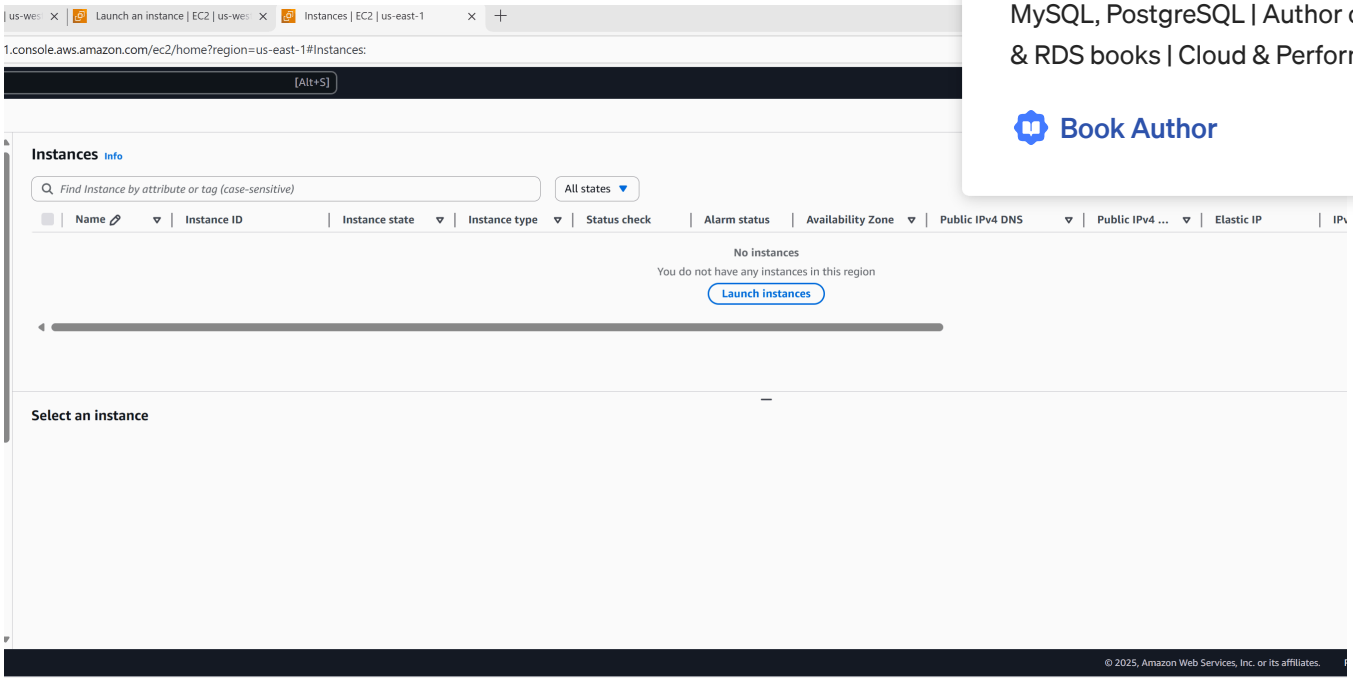
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance Expert

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...

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

🛠️ 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 Statistics with pg_stat_user_tables

When tuning PostgreSQL, one of the most important steps is to obse
You cannot optimize what you cannot measure...

Sep 7 19 1

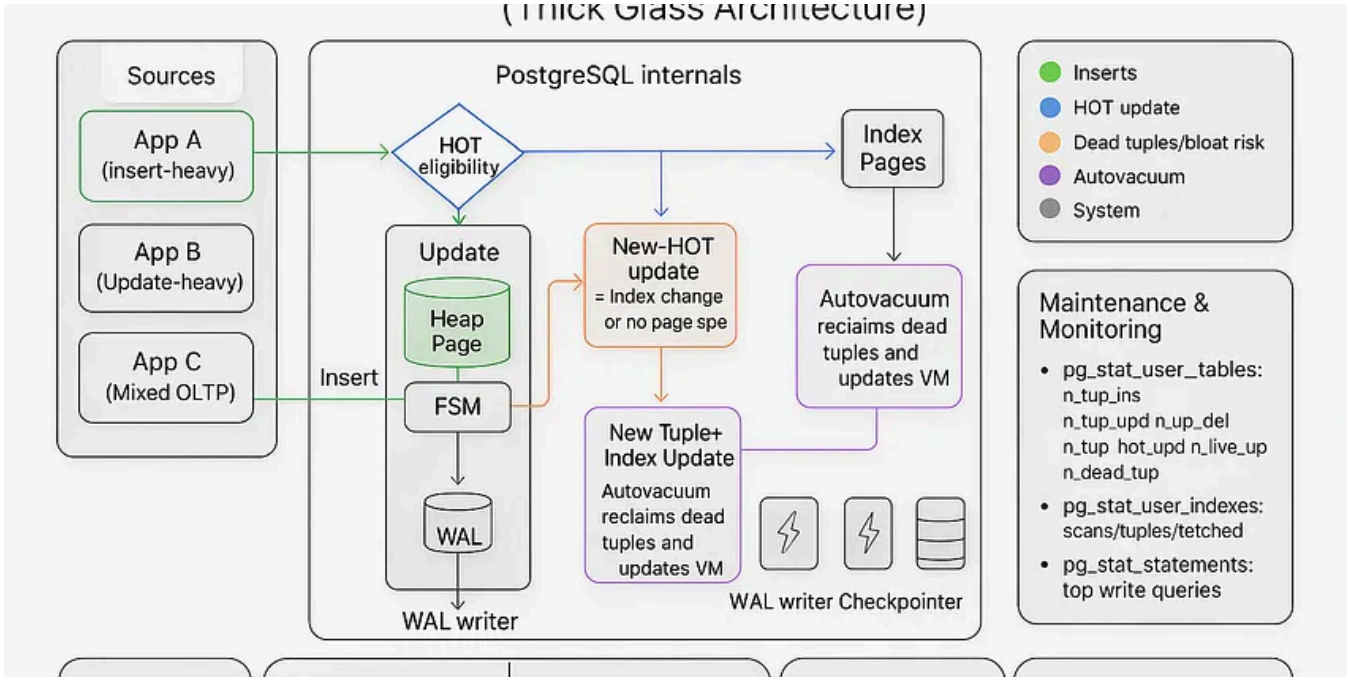
J

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

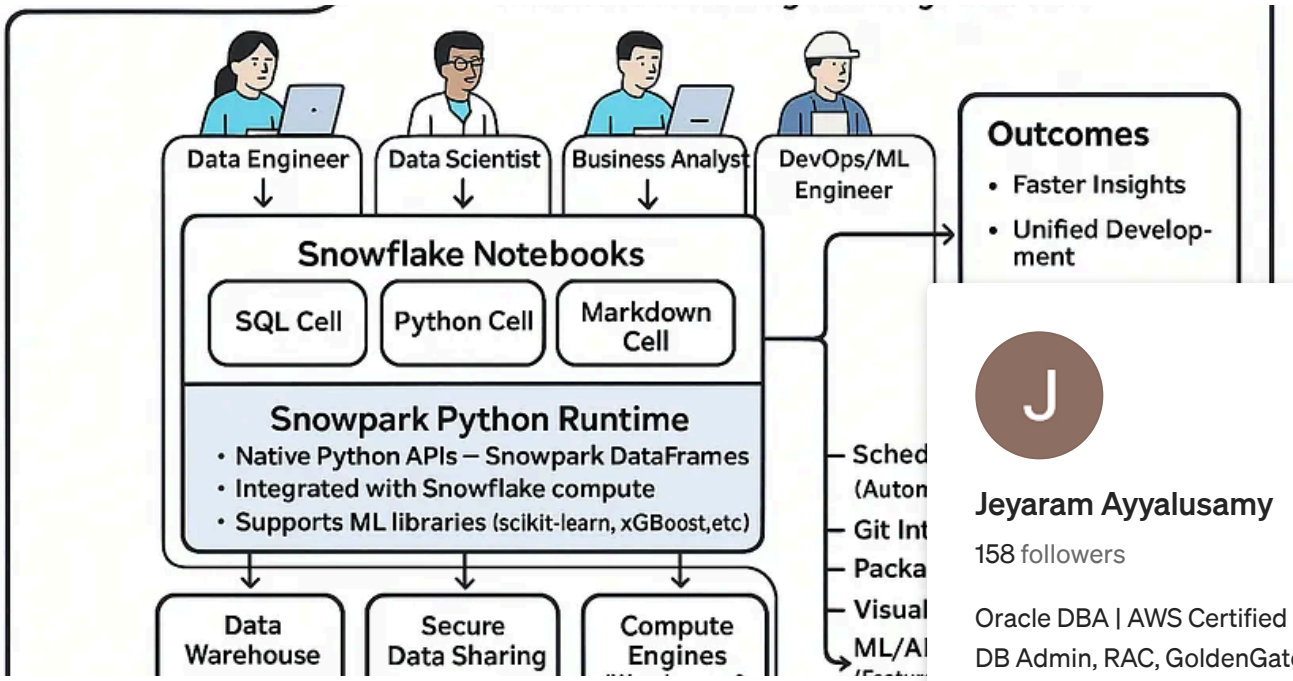


Jeyaram Ayyalusamy

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

When tuning PostgreSQL, it is very important to understand the INSERT, UPDATE, and DELETE patterns of your tables. Different workloads...

Sep 7 11



Jeyaram Ayyalusamy

16—Experience Snowflake with Notebooks and Snov 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

See all from Jeyaram Ayyalusamy

Recommended from Medium

Rizqi Mulki

The Shocking Truth About PostgreSQL Locks (Postg
documentation won't tell you)

Most developers are unknowingly creating database nightmares. He
documentation won't tell you.

Sep 13 10 2

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 im
sensitive data like passwords, customer...

6d ago 5

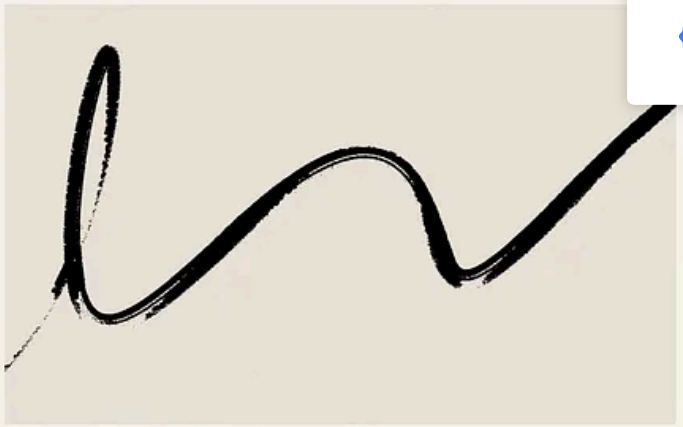
J

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

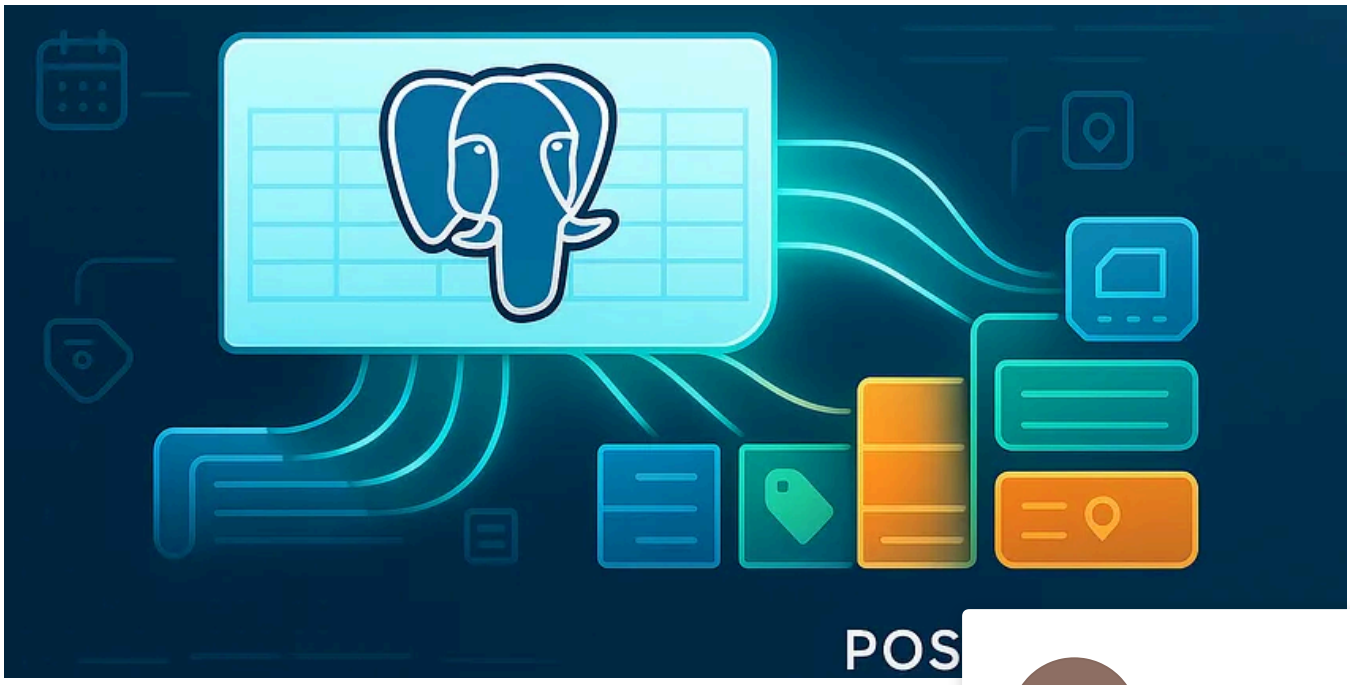


R Rohan

JSONB in PostgreSQL: The Fast Lane to Flexible Data 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 A

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

Aug 13 88 2

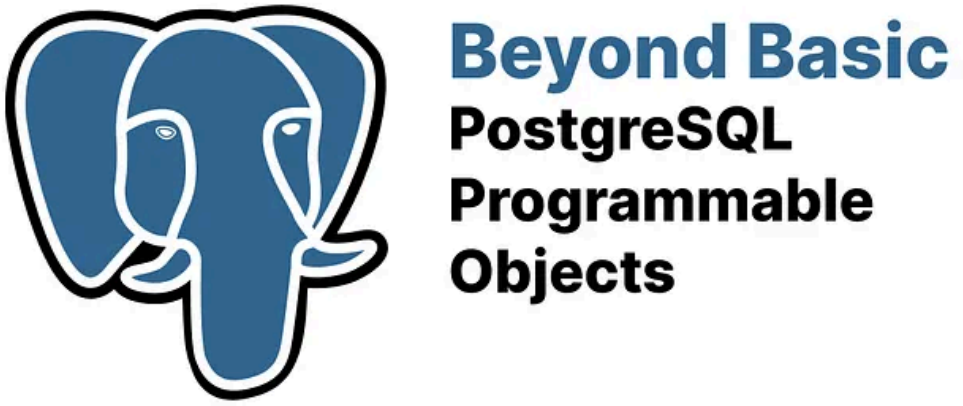
J

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 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

Vijay Gadhave

Master SQL's QUALIFY Clause for Cleaner, Faster Wi

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

May 20 12 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

See more recommendations