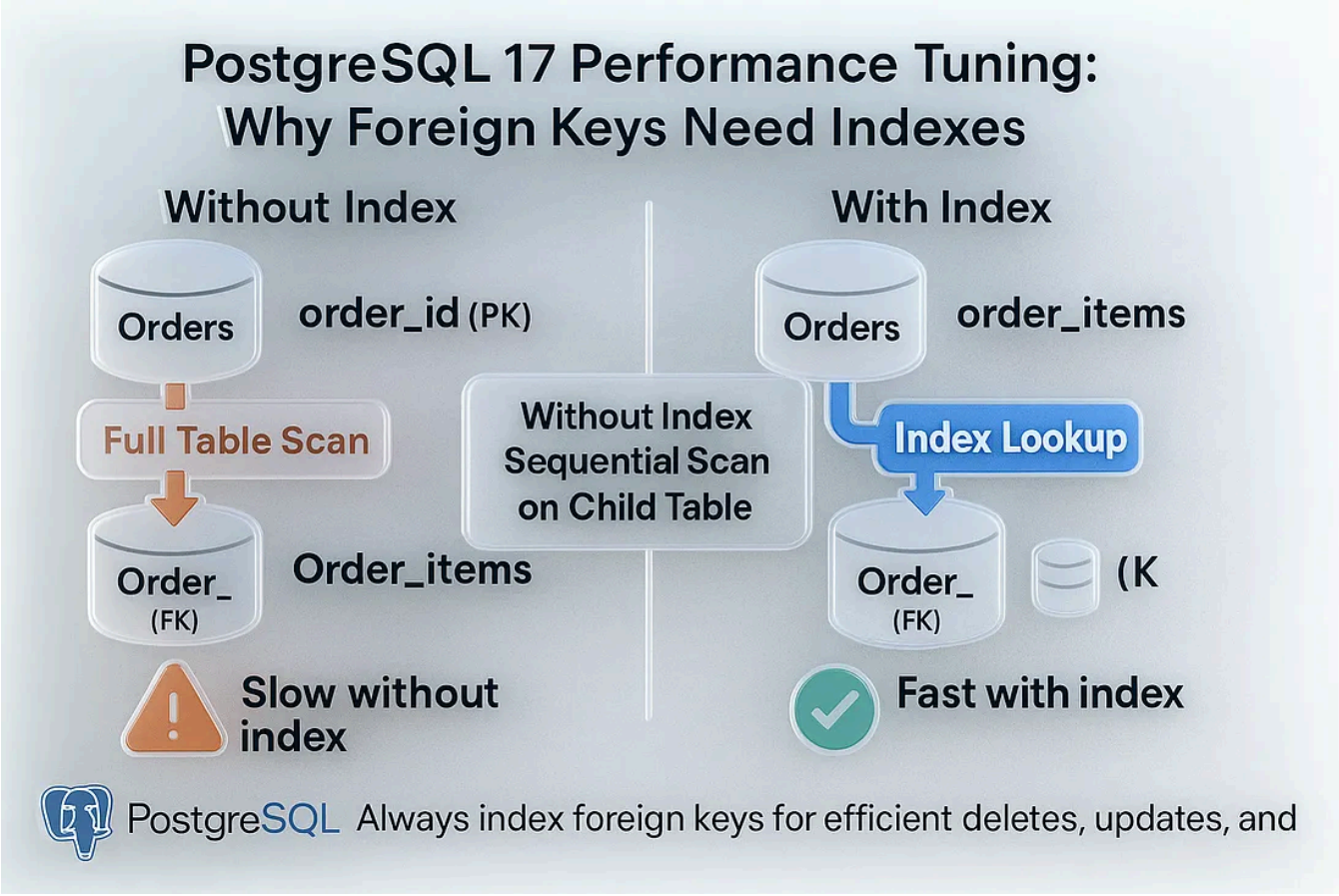
16 -PostgreSQL 17 Performance Tuning: Why Foreign Keys Need Indexes

5 min read · Sep 3, 2025

Jeyaram Ayyalusamy Following



When designing relational databases, parent-child relationships are common. A classic case is the **Orders** table (parent) and the **Items** table (child). Each order can have many items, so the child table typically has far more rows than the parent.

👉 In such cases, it's always a good practice to create an index on the **foreign key column** in the child table.

There are two main reasons:

1. **Faster joins** → Queries joining parent and child tables almost always join on the foreign key. An index dramatically speeds this up.
2. **Faster parent changes** → When deleting or updating parent rows, PostgreSQL must check child rows for constraint enforcement. Without an index, these checks can be painfully slow.

Let's build a demo to see the impact.

Step 1: Create Parent and Child Tables

```
-- Parent table: Orders
CREATE TABLE orders (
    order_no   SERIAL PRIMARY KEY,
    order_date DATE DEFAULT CURRENT_DATE
);
```

```
postgres=# -- Parent table: Orders
CREATE TABLE orders (
    order_no   SERIAL PRIMARY KEY,
    order_date DATE DEFAULT CURRENT_DATE
);
CREATE TABLE
postgres=#
```

```
-- Child table: Items
CREATE TABLE items (
```

```
    item_no       SERIAL PRIMARY KEY,
    order_no      INT,
    product_name  TEXT,
    description   TEXT,
    created_at    TIMESTAMP DEFAULT now(),
    CONSTRAINT fk_order FOREIGN KEY (order_no)
        REFERENCES orders(order_no)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

```
postgres=# -- Child table: Items
CREATE TABLE items (
    item_no       SERIAL PRIMARY KEY,
    order_no      INT,
    product_name  TEXT,
    description   TEXT,
    created_at    TIMESTAMP DEFAULT now(),
    CONSTRAINT fk_order FOREIGN KEY (order_no)
        REFERENCES orders(order_no)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
CREATE TABLE
postgres=#
```

Step 2: Populate Tables with Data

We'll insert **1 million rows** into `orders` and **3 million rows** into `items` (4 items per order).

```
-- Insert 1M orders
INSERT INTO orders (order_no)
SELECT g FROM generate_series(1, 1000000) g;
```

```
postgres=# -- Insert 1M orders
INSERT INTO orders (order_no)
SELECT g FROM generate_series(1, 1000000) g;
INSERT 0 1000000
postgres=#
```

```
-- Insert 3M items (4 per order)
INSERT INTO items (order_no, product_name, description)
SELECT (g/4)+1, 'Product_' || g, 'Desc_' || g
FROM generate_series(1, 3000000) g;
```

```
postgres=# -- Insert 3M items (4 per order)
INSERT INTO items (order_no, product_name, description)
SELECT (g/4)+1, 'Product_' || g, 'Desc_' || g
FROM generate_series(1, 3000000) g;
INSERT 0 3000000
postgres=#
```

👉 Now:

- `orders` has 1M rows.
- `items` has 3M rows linked by the foreign key.

Step 3: Query Without Index on Foreign Key

Let's try a join query:

```
\timing
SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
```

```
postgres=# \timing
Timing is on.
postgres=#
postgres=# SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
 order_no | order_date | item_no  | order_no | product_name | description |
----------+------------+----------+----------+--------------+-------------+----
      120 | 2025-09-02 | 12000476 |      120 | Product_476  | Desc_476    | 202
      120 | 2025-09-02 | 12000477 |      120 | Product_477  | Desc_477    | 202
      120 | 2025-09-02 | 12000478 |      120 | Product_478  | Desc_478    | 202
      120 | 2025-09-02 | 12000479 |      120 | Product_479  | Desc_479    | 202
(4 rows)

Time: 194.021 ms
postgres=#
```

Result:

- PostgreSQL performs a **sequential scan** on the `items` table (4 million rows) to find just 4 matches.
- Execution is slow.

Check the plan:

```
EXPLAIN ANALYZE
SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
```

```
postgres=# EXPLAIN ANALYZE
SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
                                                              QUERY PLAN
---------------------------------------------------------------------------------------
 Nested Loop  (cost=1000.42..44661.88 rows=4 width=51) (actual time=0.249..198.
   ->  Index Scan using orders_pkey on orders o  (cost=0.42..8.44 rows=1 width=
         Index Cond: (order_no = 120)
   ->  Gather  (cost=1000.00..44653.40 rows=4 width=43) (actual time=0.230..198
         Workers Planned: 2
         Workers Launched: 2
         ->  Parallel Seq Scan on items i  (cost=0.00..43653.00 rows=2 width=43
               Filter: (order_no = 120)
               Rows Removed by Filter: 999999
 Planning Time: 0.105 ms
 Execution Time: 198.853 ms
(11 rows)

Time: 200.196 ms
postgres=#
```

👉 You'll see a **Seq Scan on items**, which is inefficient.

Step 4: Add Index on the Foreign Key Column

Now add an index:

```
CREATE INDEX idx_items_order_no ON items(order_no);
```

```
postgres=# CREATE INDEX idx_items_order_no ON items(order_no);
CREATE INDEX
postgres=#
```

Step 5: Query With Index

Run the same query again:

```
EXPLAIN ANALYZE
SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
```

```
postgres=# EXPLAIN ANALYZE
SELECT *
FROM orders o
JOIN items i ON o.order_no = i.order_no
WHERE o.order_no = 120;
                                                          QUERY PLAN
------------------------------------------------------------------------------
 Nested Loop  (cost=0.85..16.98 rows=4 width=51) (actual time=0.030..0.041 rows
   ->  Index Scan using orders_pkey on orders o  (cost=0.42..8.44 rows=1 width=
         Index Cond: (order_no = 120)
   ->  Index Scan using idx_items_order_no on items i  (cost=0.43..8.50 rows=4
```

```
          Index Cond: (order_no = 120)
 Planning Time: 0.091 ms
 Execution Time: 0.060 ms
(7 rows)

Time: 0.442 ms
postgres=#
```

Result:

- PostgreSQL switches to an **Index Scan on items.**
- The query time drops dramatically.

👉 Instead of scanning 3M rows, it jumps directly to the 4 matching rows.

Step 6: Parent Delete Performance

Indexes also help when deleting from the parent:

```
DELETE FROM orders WHERE order_no = 500;
```

```
postgres=# DELETE FROM orders WHERE order_no = 500;
DELETE 1
Time: 4.355 ms
postgres=#
```

- Without an index on `items.order_no`, PostgreSQL must scan the entire `items` table to check for child rows.
- With the index, it can instantly find and remove matching rows.

Key Takeaways

- Always **index foreign key columns** in child tables.
- Benefits:
- **Joins are faster** (avoid sequential scans).
- **Parent updates/deletes are efficient** (quick referential checks).
- Without indexes, queries and deletes on large parent-child datasets can degrade to **full-table scans**, crippling performance.

✅ In PostgreSQL 17, just as in earlier versions, indexing foreign keys is one of the simplest yet most impactful tuning steps for real-world workloads.

📢 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 AWS Oracle 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

Responses (2)

Gvadakte

What are your thoughts?

Mishra Ram
Sep 7

Insightful

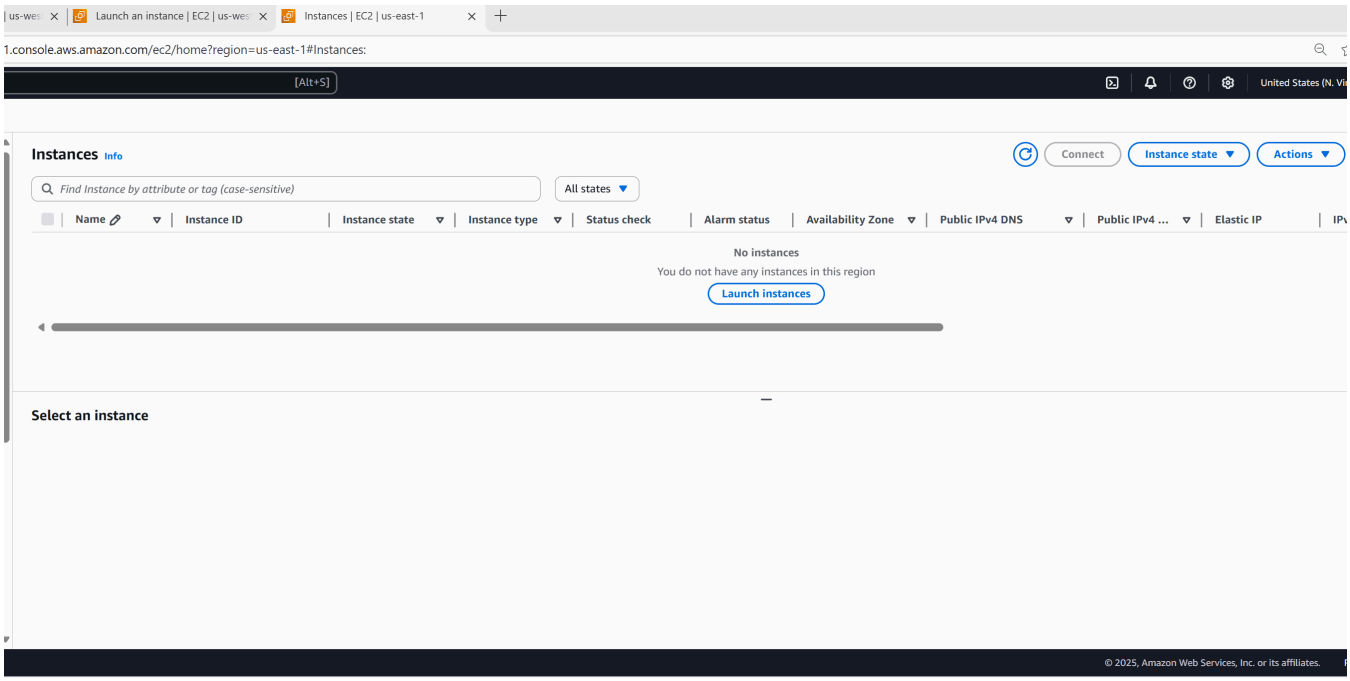
1 Reply

Awsmzn
Sep 3 (edited)

very insightful

1 Reply

More from Jeyaram Ayyalusamy

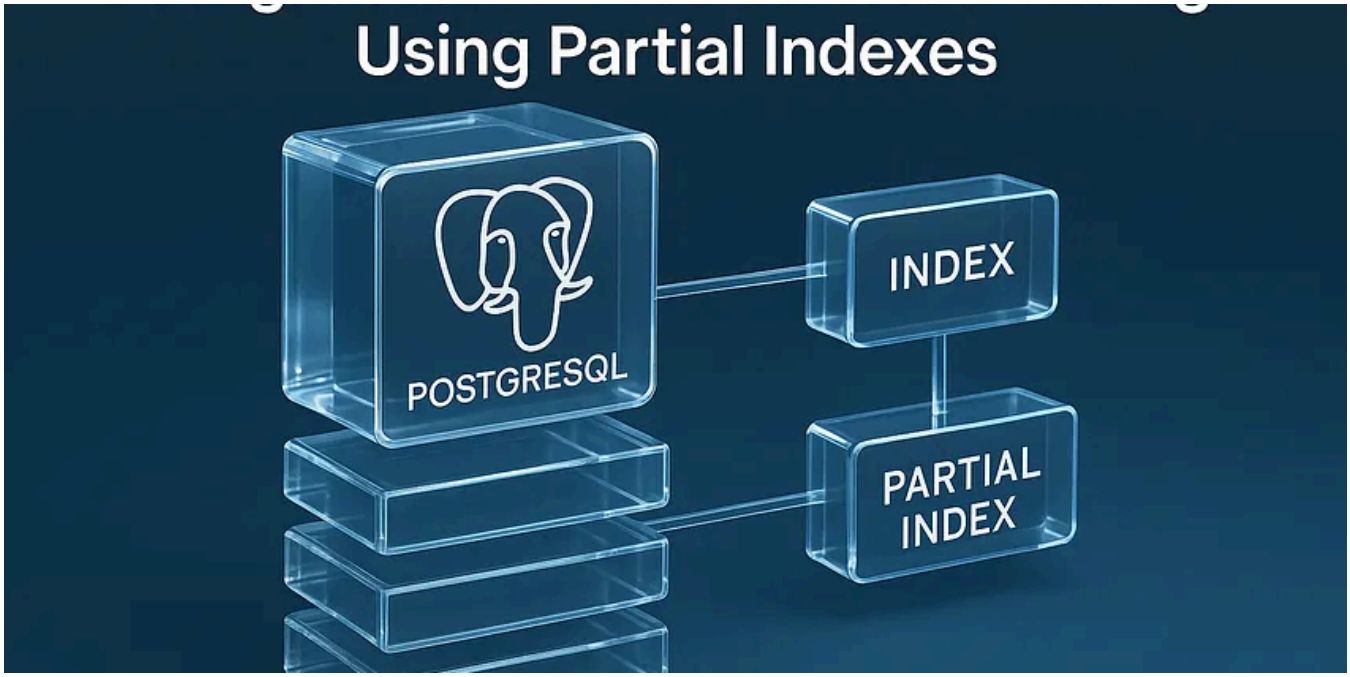


Jeyaram Ayyalusamy

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

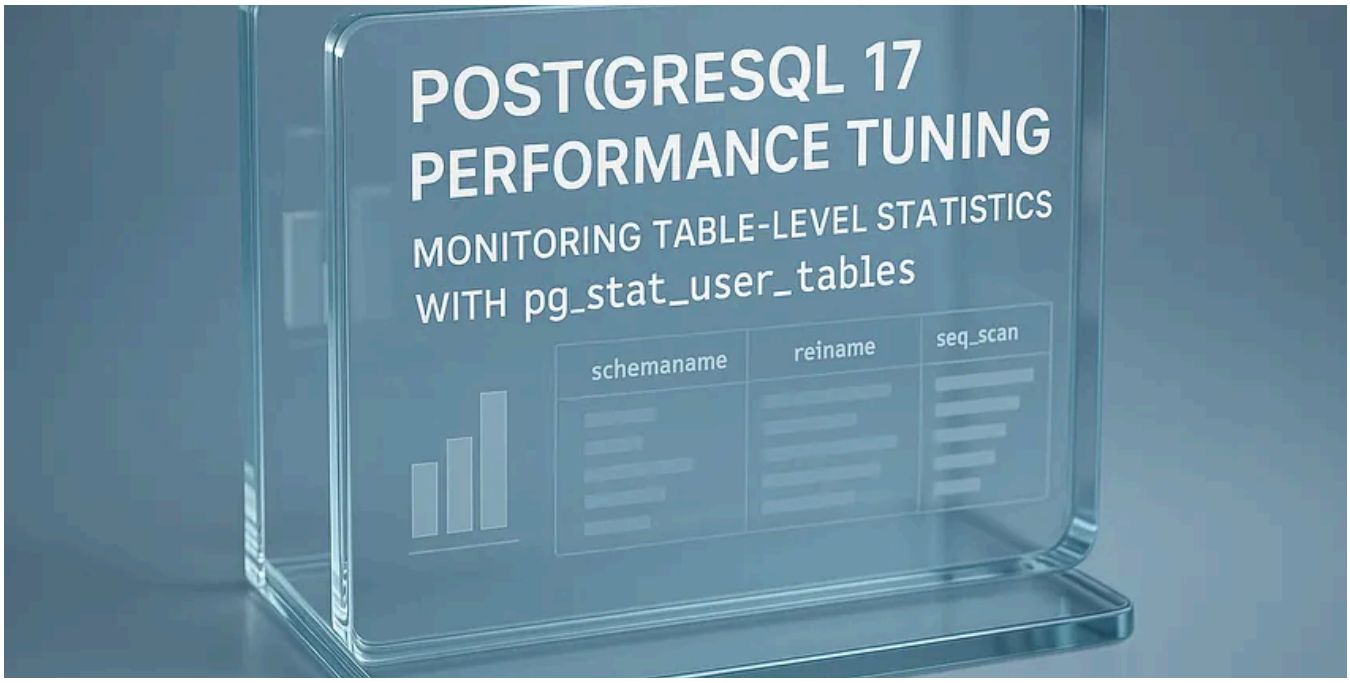


Jeyaram Ayyalusamy

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

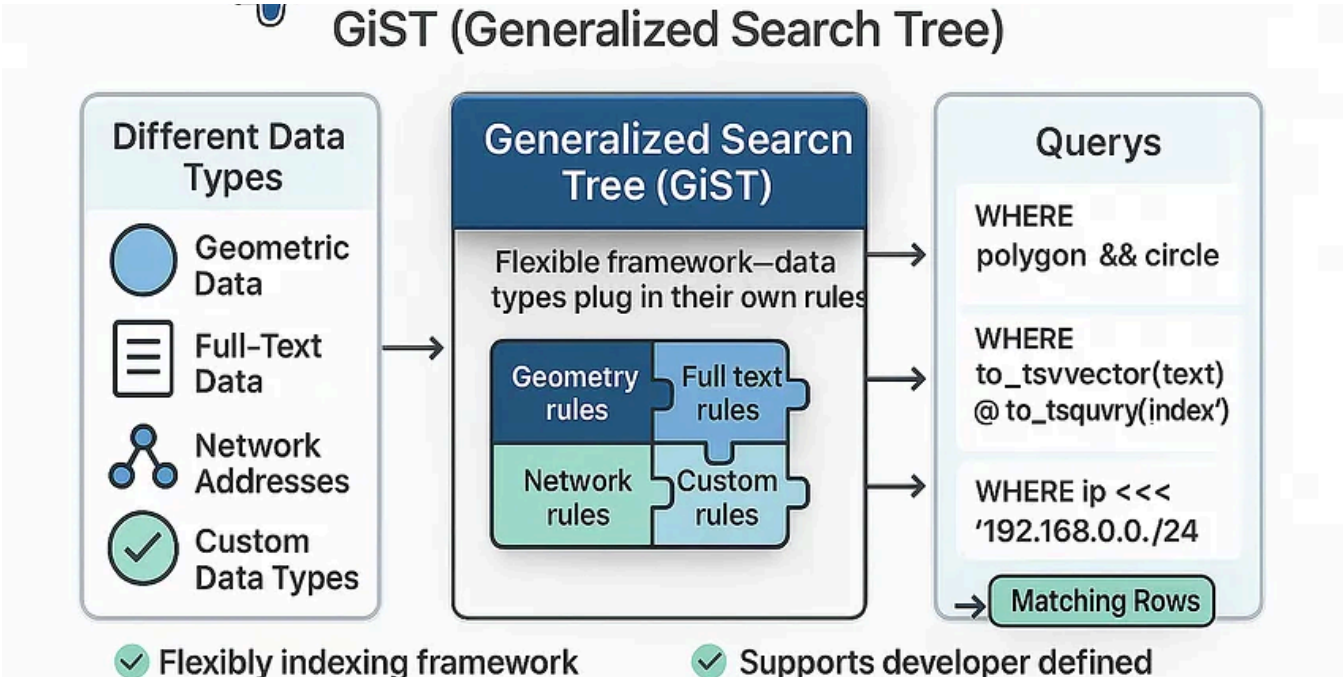


Jeyaram Ayyalusamy

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



Jeyaram Ayyalusamy

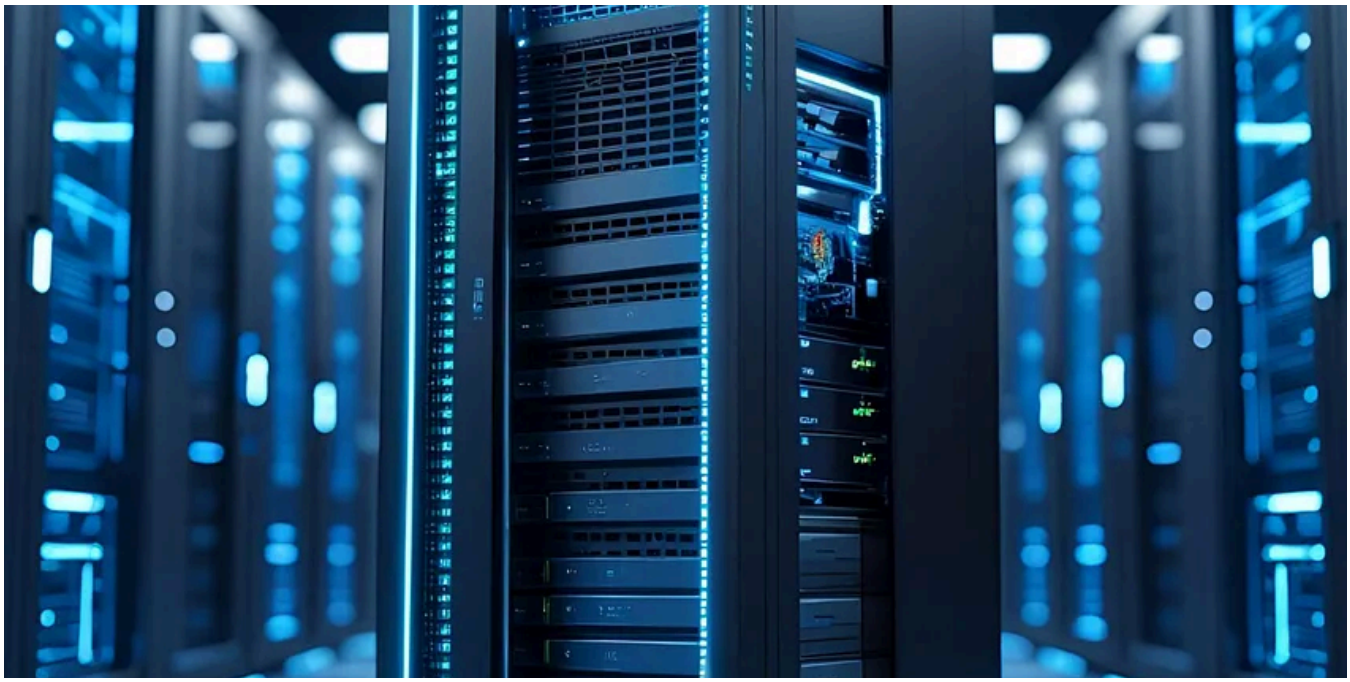
21—PostgreSQL 17 Performance Tuning: GiST (Generalized Search Tree)

GiST (Generalized Search Tree) in PostgreSQL is a flexible indexing framework that allows developers to build indexes for many different...

Sep 5 1

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

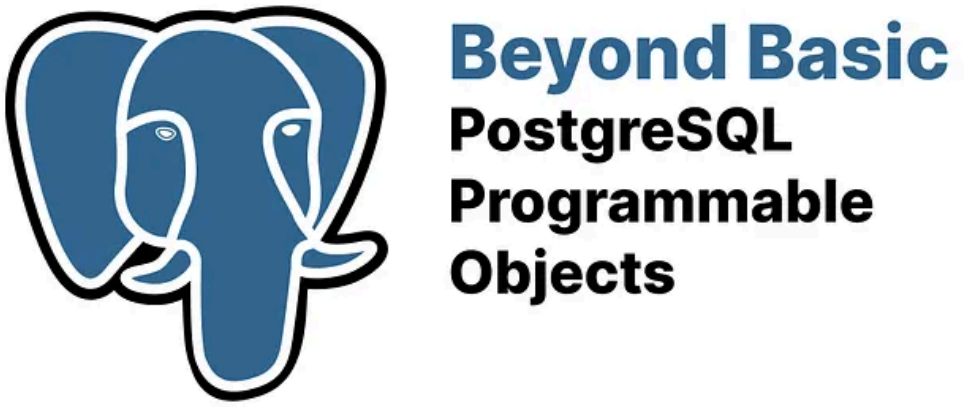
Sep 15 11 1

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

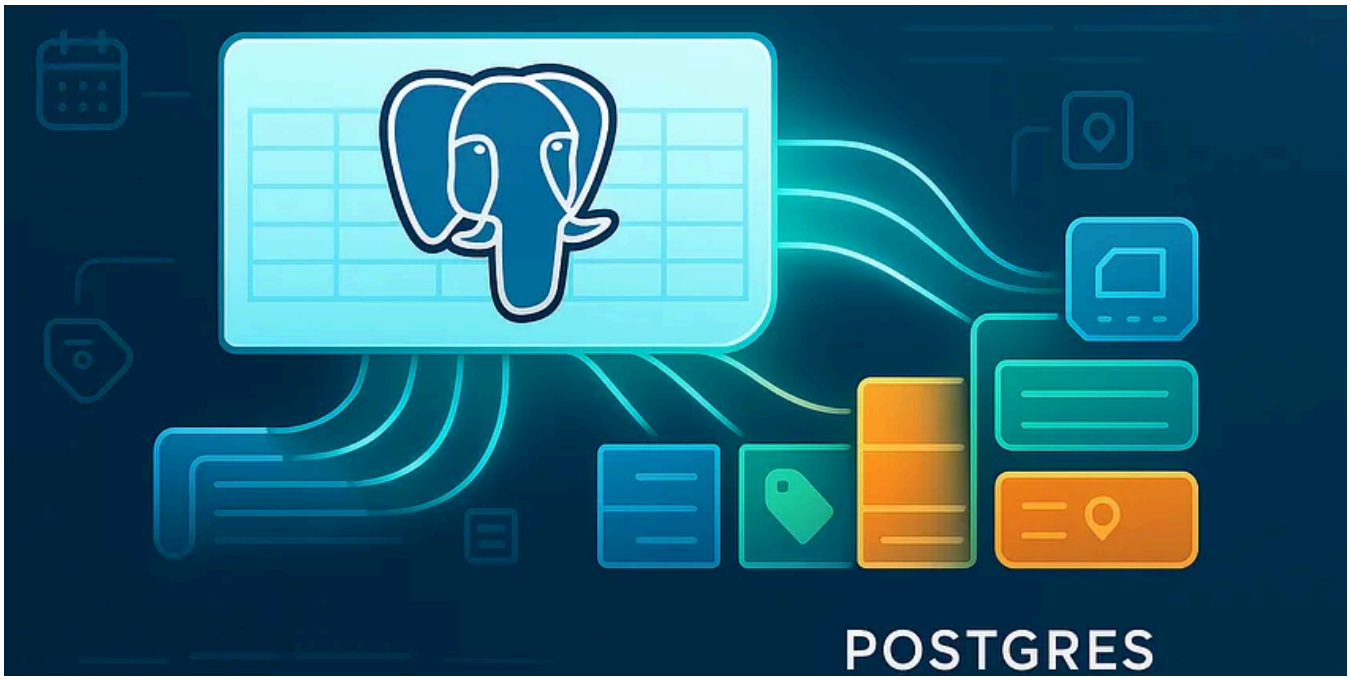


In Stackademic by bektiaw

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

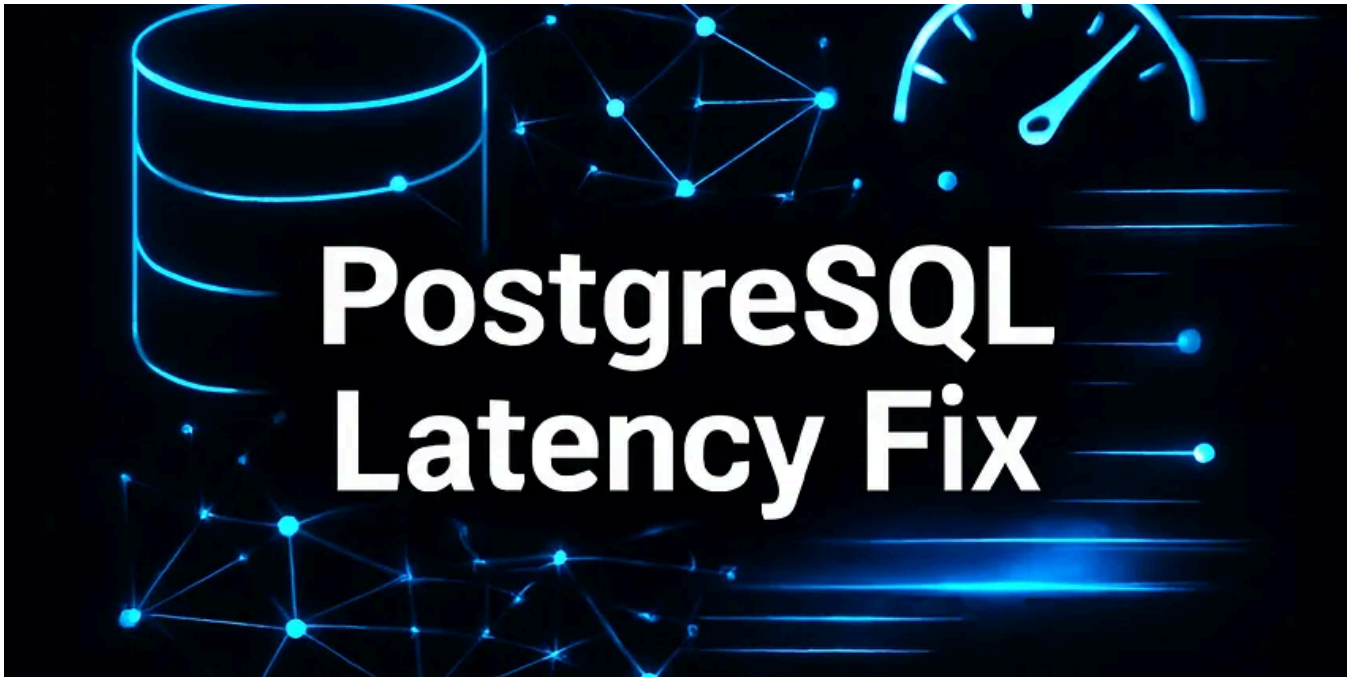


Thinking Loop

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

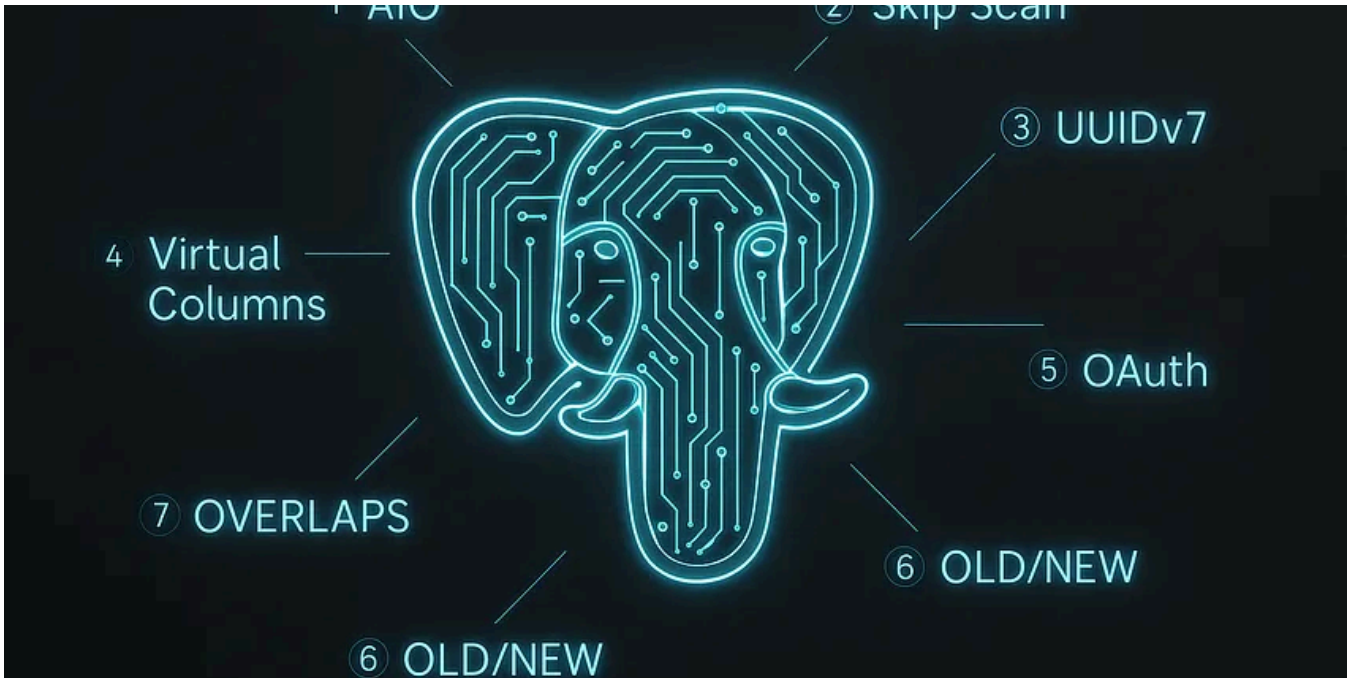


Chronicles

We Lost Users Because of PostgreSQL Latency—The Fix Wasn't More Hardware

We thought our PostgreSQL setup was rock solid. Queries were running, dashboards were loading, and everything "looked fine" in staging.

Aug 9 18



Thread Whisperer

Postgres 18 Arrives: 7 Features That Will Change Your Stack

A field-tested teardown of the Postgres 18 release for busy builders

Sep 13 307 8