PostgreSQL Tablespaces Explained: Complete Guide for PostgreSQL 17 DBAs

14 min read · Jun 11, 2025

J Jeyaram Ayyalusamy Following

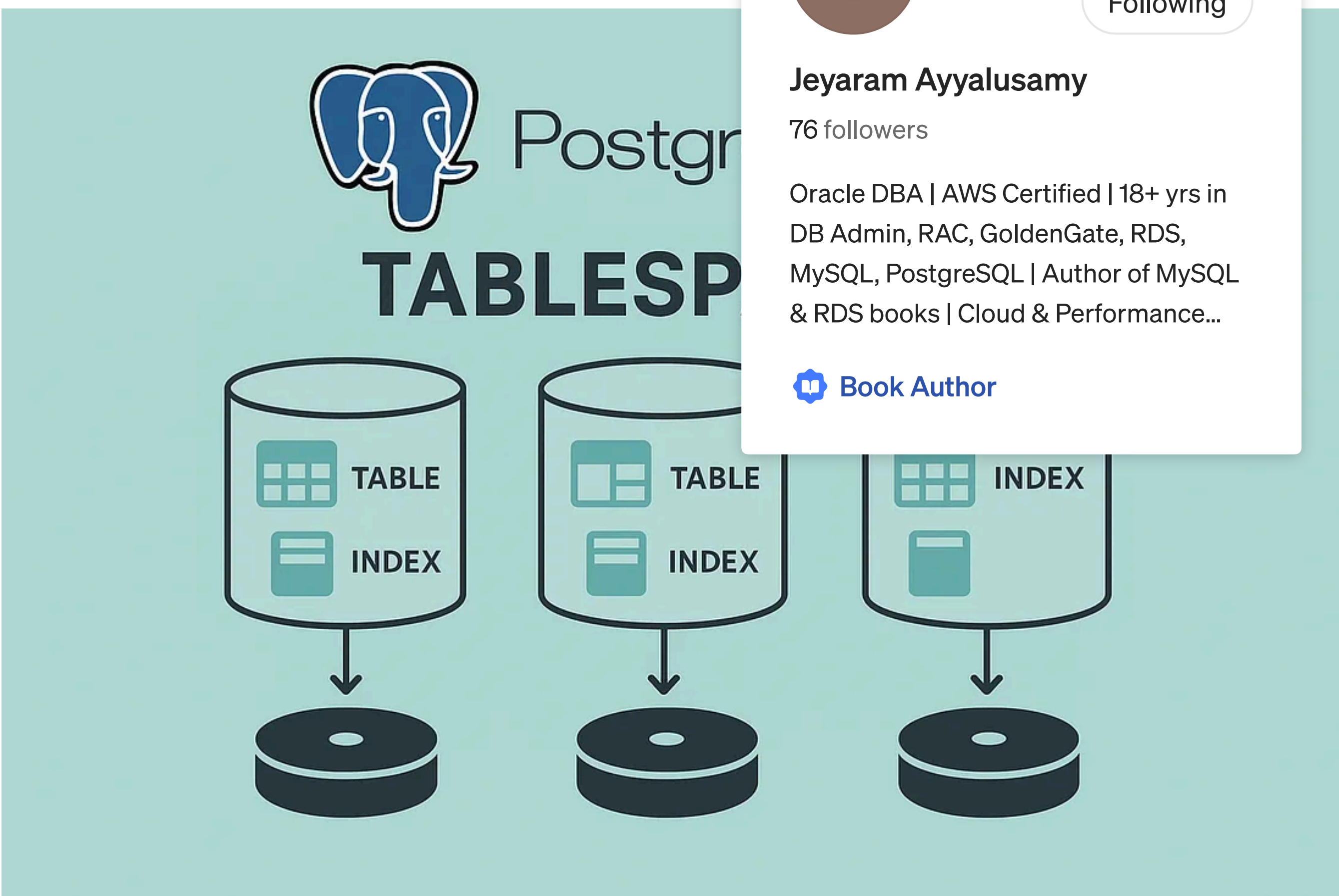
J Following

Jeyaram Ayyalusamy

76 followers

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

Book Author



PostgreSQL offers a powerful feature called **tablespaces**, allowing database administrators to take control of how and where data is physically stored. As databases grow, tablespaces help optimize performance, improve storage management, and give DBAs granular control over data placement.

In this article, we'll take a deep dive into PostgreSQL 17 tablespaces: what they are, how to create and manage them, and best practices for real-world use cases.

🚀 What Is a Tablespace in PostgreSQL?

A **tablespace** is simply a physical storage location on disk where PostgreSQL stores database objects (tables, indexes, etc.). While PostgreSQL uses a default tablespace out-of-the-box, advanced workloads benefit from placing different objects on different storage volumes.

✅ Default Tablespaces

Every PostgreSQL cluster starts with two built-in tablespaces:

Tablespace Description **pg_global** Stores share ... lt for user-created objects

By default, all tables and indexes are created in ... ur primary data directory (`$PGDATA`).

🔧 Creating a New PostgreSQL Cluster (

Introduction

PostgreSQL is known for its flexibility and robustness — and one of the key features that supports this is the ability to run **multiple clusters** side by side. Whether you're experimenting with tablespaces, testing replication, simulating standby behavior, or trying new configurations, spinning up a new cluster is often the safest and cleanest approach.

In this walkthrough, we'll explore how to create a **brand-new PostgreSQL 16 cluster** on a custom port using a custom data directory. You'll end up with an isolated instance that doesn't interfere with your default PostgreSQL setup.

Let's break it down step-by-step.

🧱 Step 1: Create the Cluster Directory

PostgreSQL stores all of a cluster's internal files, databases, transaction logs (WAL), and configuration files in a single directory known as the **data directory**. So the first step is creating that directory with the correct permissions.

```
sudo mkdir -p /postgres/data/instance1
sudo chown postgres:postgres /postgres/data/instance1
```

🔍 What You Need to Know:

- `-p` ensures that any missing parent directories (`/postgres/data/`) are created automatically.
- PostgreSQL requires that only the **`postgres`** o the data directory, hence the `chown` .

This directory will contain:

- System catalog tables
- Tablespace information
- WAL (Write-Ahead Log) segments
- Configuration files like `postgresql.conf` an

Creating a clean, dedicated folder makes your setup portable and avoids any interference with the default cluster.

⚙️ Step 2: Initialize the New Cluster

Use the built-in `pg_createcluster` command (available on Debian-based Linux systems) to initialize the cluster:

```
pg_createcluster 16 standby -d /postgres/data/instance1 -p 5434
```

✅ What This Command Does:

- Initializes a **PostgreSQL 16** instance with a cluster name `standby`
- Stores the cluster's data in the specified path `/postgres/data/instance1`
- Assigns a **custom port (5434)** to avoid conflict with the default PostgreSQL instance on port 5432

The result? A new standalone PostgreSQL environment is created with its own:

- Data files
- Configuration
- Port
- Logs

You can create and run **multiple clusters** simul
like a separate database server.

📊 Step 3: Verify Cluster Status

Now that the cluster is created, check whether

```
pg_ctlcluster 16 standby status
```

Output You Might See:

- `online` : PostgreSQL server is running for this cluster.
- `down` : Server is not yet started — you'll need to start it in the next step.

This command is especially useful in environments where multiple clusters are running. It allows you to manage them individually without confusion.

🔄 Step 4: Start PostgreSQL for the New Cluster

To start the newly created cluster, use `systemctl`, which controls system services on most modern Linux distributions:

```
sudo systemctl start postgresql@16-standby
```

🧠 What This Does:

- Activates the PostgreSQL server for the cluster named `standby` under version 16
- Starts background processes like the postmaster, autovacuum daemon, and stats collector for this specific cluster
- Listens for client connections on the define

To check that it started successfully:

```
sudo systemctl status postgresql@16-standb
```

You should see confirmation that the PostgreS

🧪 Step 5: Confirm the Data Directory

Finally, connect to the new PostgreSQL instance and verify that it's using the expected data directory.

```
psql -p 5434 -c "SHOW data_directory;"
```

This SQL query tells you exactly where PostgreSQL is reading and writing its data files. You should see:

```
data_directory
-------------------------
 /postgres/data/instance1
(1 row)
```

This step confirms that your instance is using the custom directory and not falling back to the default one (typically `/var/lib/postgresql/16/main` on Debian-based systems).

🧰 Why Set Up a Separate PostgreSQL Cluster?

Here are a few real-world use cases:

🧪 Testing New Features:

Want to test **tablespaces, partitioning, custom**
It's safer to try it in an isolated cluster.

🔄 High Availability Simulation:

Creating clusters with names like `primary` and
streaming replication or **hot standby** setups.

🧹 Upgrade Preparation:

You can prepare a new cluster on a higher vers
and test everything before switching over.

🏗️ Performance Benchmarks:

Want to compare configuration tuning (e.g., different `work_mem`, `shared_buffers`, WAL settings)? Run multiple clusters side-by-side under similar load.

🧩 What Makes a PostgreSQL Cluster Unique?

Each cluster has:

- A unique **data directory**

- A unique **configuration set**
- A dedicated **port number**
- Its own **log and monitoring context**
- Separate **databases, schemas, and users**

This means you can safely run multiple clusters for different purposes — even on the same machine — without any cross-talk or collision.

✅ Summary

By completing this tutorial, you've learned how

1. Set up a clean directory for PostgreSQL dat
2. Initialize a new cluster with `pg_createclust`
3. Assign it a custom port for isolation
4. Start and verify the cluster
5. Confirm its operational data path

You're now ready to move on to advanced Postg d environment.

🏗️ Creating a New Tablespace in PostgreSQL (Step-by-Step)

In PostgreSQL, a **tablespace** is a powerful feature that allows you to control **where** your data is stored on disk. While PostgreSQL by default stores all database objects (tables, indexes, etc.) in the main data directory, **tablespaces enable you to place specific objects in alternative directories or mount points** — offering more flexibility for performance tuning, storage tiering, and disk management.

In this step, we'll walk through creating a new tablespace for the PostgreSQL cluster we created earlier (running on port `5434`).

1️⃣ Create the Physical Directory

Before PostgreSQL can use a new tablespace, you need to create the underlying directory on the filesystem. This is where the actual data files for the tablespace will be stored.

```
mkdir -p /var/lib/postgresql/16/tbs1
```

✅ Key Points:

- `-p` ensures that parent directories are created if they don't already exist.
- Choose a location with sufficient storage —
  doing performance tuning.
- The directory **must be owned by the** **`postg`**

You may also need to set permissions:

```
chown postgres:postgres /var/lib/postgresc
chmod 700 /var/lib/postgresql/16/tbs1
```


2️⃣ Create the Tablespace in PostgreSQL

Once the physical directory exists, tell PostgreSQL to register it as a tablespace using the `CREATE TABLESPACE` SQL command.

```
psql -p 5434 -c "CREATE TABLESPACE tbs1 LOCATION '/var/lib/postgresql/16/tbs1';
```

🧠 What's Happening:

- You're connecting to the PostgreSQL instance on port `5434`.
- The `CREATE TABLESPACE` command registers a new logical tablespace named `tbs1`.
- PostgreSQL will internally create a **symbolic link** (`symlink`) in the cluster's data directory (`pg_tblspc`) pointing to the physical location.

You can now start creating tables or indexes using this tablespace with the `TABLESPACE` clause:

```
CREATE TABLE my_table (id INT, name TEXT) TABLESPACE tbs1;
```


3️⃣ List All Tablespaces

To confirm that the tablespace was successfull
command in `psql`:

```
psql -p 5434 -c "\db+"
```

This will show:

- The name of each tablespace
- Its owner
- The location path
- Access privileges

✅ You should see `tbs1` in the list with the correct file path you provided.

4️⃣ Validate the Symbolic Links

PostgreSQL does **not store tablespace data directly inside the `PGDATA` directory.** Instead, it uses symbolic links (symlinks) inside the `pg_tblspc/` directory, which point to the actual filesystem location of each tablespace.

To inspect this, set your `PGDATA` environment variable and check the symlinks:

```
export PGDATA=/postgres/data/instance1
ls -l $PGDATA/pg_tblspc/
```

🧠 What You Should See:

A symlink entry like this:

```
12345 -> /var/lib/postgresql/16/tbs1
```

Here, `12345` is an internal OID (object identifie `tbs1` tablespace.

This confirms:

- PostgreSQL successfully created the symbolic link
- The engine can resolve the custom storage path for reading and writing data

🧩 Why Use Tablespaces?

Here are a few compelling use cases:

- 🔄 **Storage tiering:** Store hot data on SSDs and cold/archive data on slower disks.
- 🚀 **Performance tuning:** Distribute large indexes or high-I/O tables across different physical disks.

- 🔒 **Disk quota enforcement:** Isolate large tables to prevent them from consuming shared storage.
- 📦 **Backup planning:** Back up or restore specific parts of the database by targeting certain tablespaces.

✅ Summary

You've now successfully:

1. Created a custom physical directory for data storage.
2. Registered a new tablespace in PostgreSQL.
3. Verified its presence using SQL commands
4. Inspected how PostgreSQL manages the loc

You're now ready to create tables, indexes, or e custom tablespace — giving you finer control o

🏗 Creating Databases and Tables Using

Once you've created a new tablespace in PostgreSQL, the next step is learning **how to use it effectively**. PostgreSQL allows you to assign specific **databases, tables**, or **indexes** to different tablespaces — giving you control over where your data physically resides on disk.

This feature is especially useful when you want to:

- Optimize storage across multiple disks
- Separate workloads
- Improve I/O performance
- Plan for future migrations

In this section, we'll explore how to assign databases and tables to a tablespace and how to **move them later if needed.**

1️⃣ Creating Databases with and without Tablespaces

Let's create two new databases: one using the default tablespace, and one using the custom tablespace we created earlier (`tbs1`).

```
psql -p 5434 -c "CREATE DATABASE db1;"
psql -p 5434 -c "CREATE DATABASE db2 TABLESPACE tbs1;"
```

🔍 Explanation:

- `db1` : This database will use the **default tabl** ... ts unless specified otherwise.
- `db2` : This database will use `tbs1` as the **def** ... tables or indexes) created inside it — unles ... assigned.

🚨 **Note:** A database's default tablespace deter ... **tablespace is specified** during object creation.

2️⃣ Creating Tables Within Tablespaces

Let's connect to `db1` and create a few tables — each with different storage strategies.

```
psql -p 5434 -c "\c db1"
```

Now inside `db1` , create the following:

```
CREATE TABLE tab1 (a int);  -- Stored in db1's default tablespace (pg_default)
CREATE TABLE tab2 (a int) TABLESPACE tbs1;  -- Explicitly stored in tbs1
CREATE TABLE tab3 (a int) TABLESPACE pg_default;  -- Explicitly stored in pg_de
```

🧠 Why It Matters:

- `tab1` : Uses whatever default tablespace is assigned to the database (`pg_default` unless overridden).
- `tab2` : Manually assigned to the custom tablespace `tbs1` . Useful for isolating performance-sensitive or high-growth tables.
- `tab3` : Explicitly stored in the system default tablespace, which may be important for standardization or administr

This ability to **control storage location per tabl** optimization and disk usage management.

🔄 Moving Tables Between Tablespaces

PostgreSQL also provides a simple way to **move** ur needs change — whether for performance tuni upgrades.

✅ Move a Single Table

You can move a single table to a different tablespace using the `ALTER TABLE` command:

```
ALTER TABLE tab2 SET TABLESPACE pg_default;
```

🔍 Use Case:

Let's say `tab2` was originally stored in `tbs1`, but you want to consolidate everything back into the default tablespace — this command will safely transfer all of the table's data and metadata.

✅ Move All Tables in a Tablespace

You can move **all objects** from one tablespace to another using a single command:

```
ALTER TABLE ALL IN TABLESPACE tbs1 SET TABLESPACE pg_default;
```

🧠 What It Does:

- Iterates over all user tables in `tbs1`
- Moves them to `pg_default`
- Preserves table names, schemas, and data i

🚨 **Important Note:** This command only affect objects like sequences. To move those, you'll ne statements for each object type.

🧩 When to Move Tables Between Tablespaces

You may want to move objects when:

- You're decommissioning or replacing storage devices
- You need to balance I/O workloads across disks
- Tablespace `tbs1` is filling up
- You want to consolidate data for easier backup or replication

PostgreSQL makes these migrations smooth with minimal downtime, though large tables can take time and require disk I/O capacity.

✅ Summary

With just a few SQL commands, you've learned how to:

- Create databases using specific tablespaces
- Create individual tables and assign them to targeted storage
- Move tables between tablespaces dynamically

Tablespaces in PostgreSQL are an often underused but powerful tool for **storage control**, **performance tuning**, and **long-term data management.** By mastering them, you're building the foundation for a scalable, production-ready database architecture.

🔧 Understanding PostgreSQL Tablespa… s, and Backup Strategies

PostgreSQL tablespaces give database administ… nd where data is stored. But beyond just separatin… also be **tuned, controlled at the session level**, a… are critical for managing large or high-perform…

In this post, we explore three advanced but practical features:

1. Setting cost parameters for query planning
2. Defining a default tablespace for session-level object creation
3. Backing up tablespaces correctly with `pg_basebackup`

Let's dive in.

🔧 Tablespace Properties: Tuning for Performance

Tablespaces in PostgreSQL aren't just about storage paths — they can also help inform the **query planner** about the cost of reading from disk. PostgreSQL allows you to assign **cost parameters** to tablespaces to optimize query performance based on the underlying hardware.

```
ALTER TABLESPACE space2 SET (seq_page_cost=0.5, random_page_cost=0.5);
```

🧠 What This Means:

- `seq_page_cost` : The estimated cost of reading a sequential page from this tablespace.
- `random_page_cost` : The estimated cost of reading a random page (e.g., an index lookup).

By default, PostgreSQL assumes mechanical sp… is expensive. But if you're using **fast SSDs or NVM**… cheaper — so lowering the cost estimates helps… .

✅ **Example Use Case:**
If you move an index-heavy table to an SSD-bac… `random_page_cost` can help PostgreSQL choose … - improving speed.

📄 Setting a Default Tablespace for a Session

There are cases where you want all new tables, indexes, or objects you create in a session to automatically go into a specific tablespace — without explicitly specifying `TABLESPACE` every time.

Here's how to set that up:

1️⃣ Create a Physical Directory

```
sudo mkdir -p /postgres/data/newts1
```

```
sudo chown postgres:postgres /postgres/data/newts1
```

This directory will host the new tablespace's files and must be owned by the `postgres` user.

2️⃣ Register the Tablespace

```
psql -p 5434 -c "CREATE TABLESPACE tsttbs1 LOCATION '/postgres/data/newts1';"
```

You now have a new tablespace `tsttbs1` availa
running on port 5434.

3️⃣ Set the Default Tablespace in the Ses

```
SET default_tablespace = tsttbs1;
CREATE TABLE foo (i int);
```

🔍 **Result:**
The table `foo` will be created in `tsttbs1` automatically, without having to specify the tablespace manually in the `CREATE TABLE` statement.

✅ **Use Case:**
This is useful during bulk table creation or temporary workloads where you want to isolate data in a specific storage location just for a session or script run.

🔄 Backing Up Tablespaces Safely

When working with custom tablespaces, backing up your database requires more than just copying files. You must ensure that **external tablespace paths** are correctly captured and restored.

PostgreSQL provides this capability via `pg_basebackup`, using the `--tablespace-mapping` option.

```
pg_basebackup --format=p \
--tablespace-mapping=/tmp/space2=/tmp/space2backup \
-D plainb
```

🔍 Breakdown:

- `--format=p` : Creates a **plain** backup format).
- `--tablespace-mapping` : Tells PostgreSQL to r n (`/tmp/space2`) to a different location (`/tmp/` process.
- `-D plainb` : Specifies the target directory fo

✅ Why This Matters:

Without tablespace mapping, a backup may fai original paths don't exist or aren't writable during recovery. With this feature, you can **relocate tablespaces during backup or restore** without data loss.

✅ Final Thoughts

PostgreSQL tablespaces offer more than just alternative storage paths — they unlock:

- **Query planner optimization** through cost tuning
- **Session-level control** for dynamic data placement
- **Safe, flexible backups** through tablespace mapping

By mastering these advanced features, you'll be able to fine-tune performance, control storage at scale, and confidently manage your PostgreSQL infrastructure across environments.

📊 Why Use Tablespaces in PostgreSQL?

As your PostgreSQL database grows in size and complexity, managing **storage layout** becomes increasingly important. That's where **tablespaces** come in — a feature that allows you to define **where** specific tables, indexes, or even entire databases are physically stored on disk.

Tablespaces give you more than just flexibility — they unlock powerful ways to optimize performance, organize data, and mar
effectively.

Here's a breakdown of **why you should use tab** r
high-performance PostgreSQL environments:

✅ Store Different Workloads on Differer

Not all data is created equal — some is read cor
accessed infrequently (cold). With tablespaces, you can **assign different types of data to different storage classes:**

- Use **SSD (Solid State Drives)** for indexes and high-frequency transactional tables to maximize IOPS (Input/Output Operations Per Second).
- Use **HDD (Hard Disk Drives)** for archival data, logs, or historical records that don't require frequent access.

This tiered approach helps optimize both **performance and cost** by using fast but expensive storage only where it matters most.

✅ Improve I/O Performance for Heavy Workloads

By strategically placing high-read or high-write tables on separate physical disks or partitions, you can **reduce contention** and improve I/O throughput.

Example:

- Place OLTP (Online Transaction Processing) tables on one tablespace.
- Place analytical reporting tables or staging data on another.

This separation ensures that heavy analytics jobs don't impact transactional performance — a common bottleneck in unified workloads.

✅ Control Disk Usage Across Multiple P

Tablespaces allow you to spread your data acro ,
or even mount points. This helps:

- Prevent single-disk exhaustion
- Maximize total available storage
- Simplify disk-level monitoring and alerting

You gain **granular control** over how storage is
environments with limited or shared storage resources.

✅ Logical Organization of Very Large Datasets

Beyond performance, tablespaces help with **data management** and **organization**. For example:

- Assign each client, department, or business unit their own tablespace.
- Create separate tablespaces for staging, production, and testing.
- Archive old tables or partitions to a designated archive tablespace.

This logical separation makes **backup, restore, and compliance processes easier** and more modular. In regulated industries, it even enables easier data segregation for audits or legal hold requirements.

⚠️ Important Notes on Tablespaces

While tablespaces are powerful, there are some critical best practices and limitations you must keep in mind to avoid performance degradation or data loss.

⚠️ Tablespaces Must Reside on the Same Server

PostgreSQL does **not support remote tablespac** e **located on the local file system** of the server w running.

✅ Why?

PostgreSQL relies heavily on **low-latency, cons** performance. Remote filesystems can introduc data corruption risks — especially during crash

⚠️ Avoid NFS or Remote Storage (Unless Highly Reliable)

Using **NFS, SMB, or network-mounted volumes** is strongly discouraged unless:

- You're using a **fully POSIX-compliant**, high-performance, and highly available setup.
- You've tested for crash safety, I/O consistency, and fsync behavior.

Even small inconsistencies or delays in remote filesystems can cause PostgreSQL to fail in subtle and dangerous ways.

⚠️ Ensure Proper Filesystem Permissions

PostgreSQL must have full read/write access to the tablespace directory. That means:

- The directory must be **owned by the `postgres` user**
- The permissions should typically be set to `0700` to ensure exclusive access

Failure to set correct ownership and permissions can cause:

- Errors during object creation
- Failure to access data at runtime
- Issues during backup or recovery

Before creating the tablespace, always run:

```
sudo chown postgres:postgres /path/to/tabl
chmod 700 /path/to/tablespace
```


✅ Final Thoughts

Tablespaces in PostgreSQL offer a **strategic advantage** in managing data at scale. From optimizing performance to organizing massive datasets across different storage types, they empower database administrators and developers alike to build smarter, faster, and more maintainable systems.

But like any powerful tool, tablespaces come with operational caveats. Stick to **local, reliable storage**, ensure **permissions are correct**, and avoid using them blindly without understanding your workload characteristics.

🏁 Summary

Tablespaces in PostgreSQL offer an excellent mechanism for advanced storage management. While often overlooked in smaller installations, large-scale production environments benefit significantly from tablespace planning — especially when balancing performance, storage costs, and data growth.

> PostgreSQL tablespaces = physical storage flexibility + logical data management.

👉 *If you found this guide helpful, follow me(medium) for more practical PostgreSQL tutorials, database architecture guides, and hands-on DBA content.*

🔗 Let's Connect!

If you enjoyed this post or would like to connec ... out to me on LinkedIn:
👉 Jeyaram Ayyalusamy

I regularly share content on **PostgreSQL, datab** **technologies, and data engineering.** Always ha discuss ideas!

Postgresql Oracle MySQL Open Sourc

J

Jeyaram Ayyalusamy

76 followers

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

Book Author

J

Following

Written by Jeyaram Ayyalusamy

76 followers · 2 following

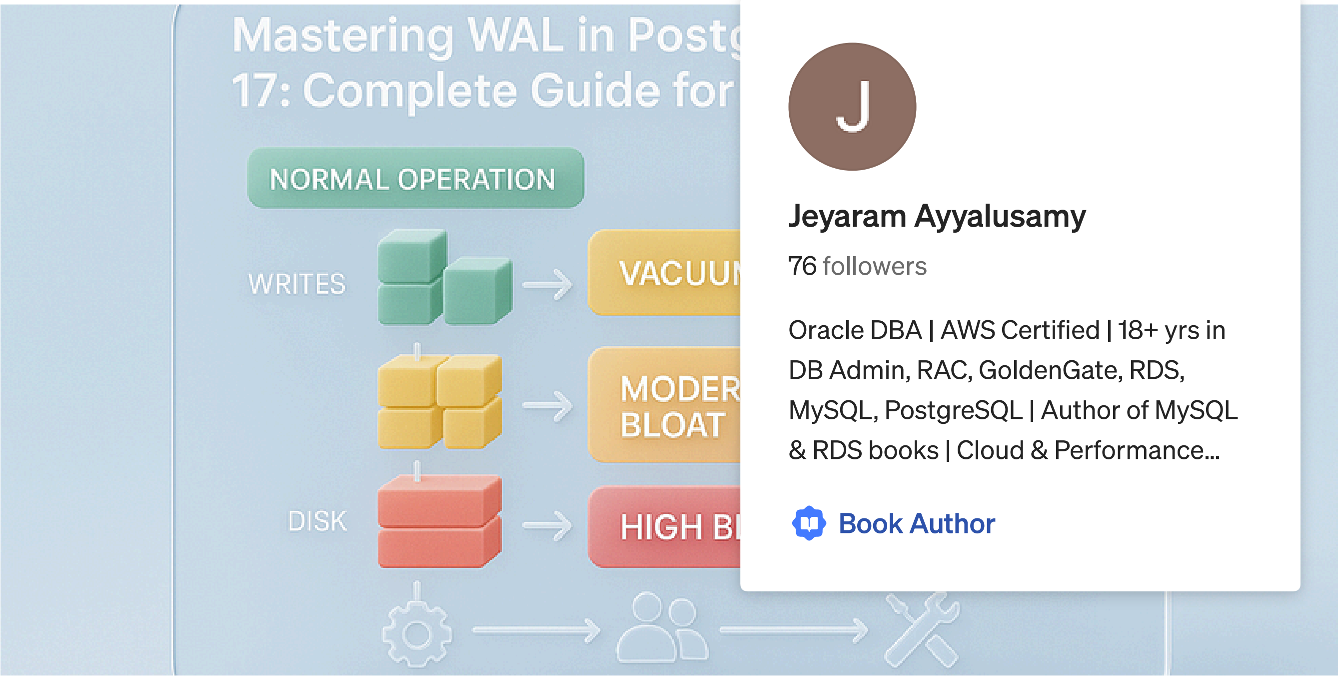
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance Expert

No responses yet

Gvadakte

What are your thoughts?

More from Jeyaram Ayyalusamy



J

Jeyaram Ayyalusamy

76 followers

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

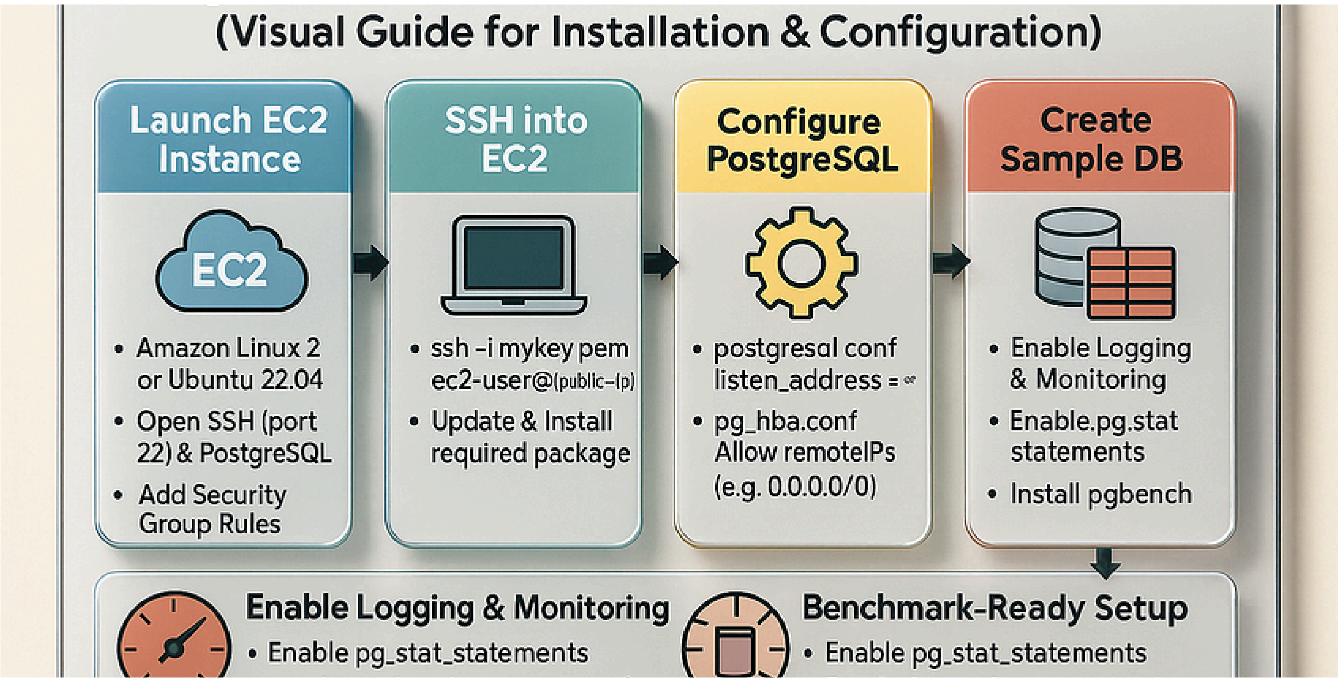
Book Author

J Jeyaram Ayyalusamy

Understanding and Managing Bloating in PostgreSQL: A Complete Guide for DBAs

PostgreSQL is a powerful, reliable, and feature-rich open-source relational database system. It's praised for its extensibility, ACID...

Jun 25 52



Jeyaram Ayyalusamy

PostgreSQL 17 on AWS EC2 — Full Installa Walkthrough

Setting up PostgreSQL 17 on AWS EC2 might seem c wn each component, it becomes an efficient and...

1d ago 50

Jeyaram Ayyalusamy

76 followers

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

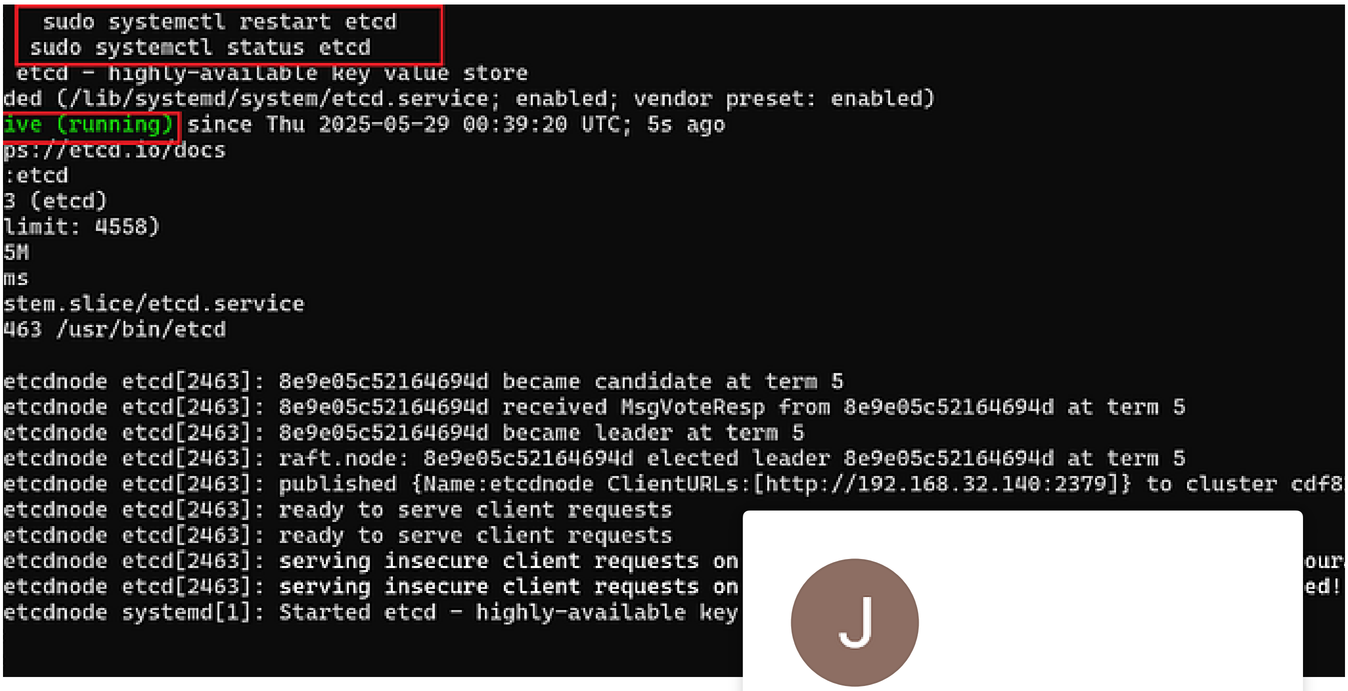
Book Author

Jeyaram Ayyalusamy

How to Install PostgreSQL 17 on Red Hat, Rocky, AlmaLinux, and Oracle Linux (Step-by-Step Guide)

PostgreSQL 17 is the latest release of one of the world’s most advanced open-source relational databases. If you’re using a Red Hat-based…

Jun 3



Jeyaram Ayyalusamy

Building a High Availability PostgreSQL C Step Guide

This will follow the standard instructional structure w your nodes: `node1`, `node2`, `etcdnode`…

May 29 1

Jeyaram Ayyalusamy

76 followers

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

Book Author

See all from Jeyaram Ayyalusamy

Recommended from Medium

Azlan Jamal

Stop Using SERIAL in PostgreSQL: Here’s

In PostgreSQL, there are several ways to create a PRI
usually involve different ways of generating the...

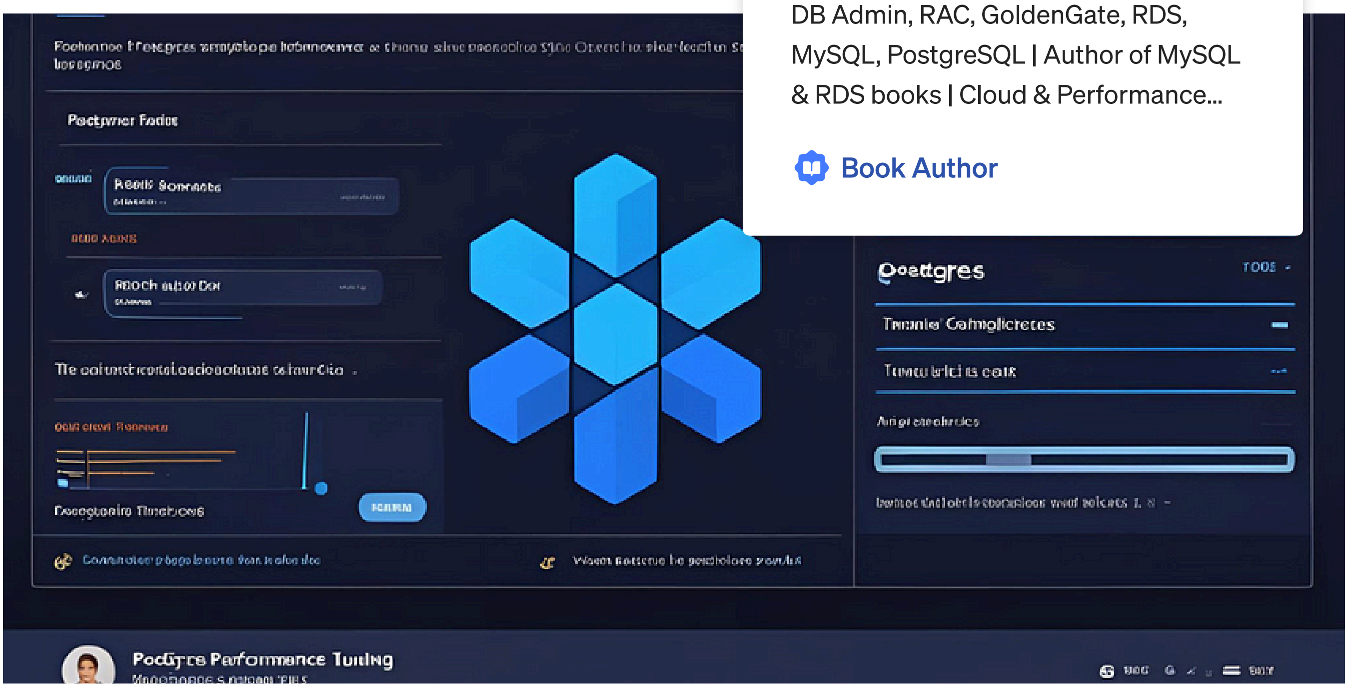
Jul 12 33

Jeyaram Ayyalusamy

76 followers

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

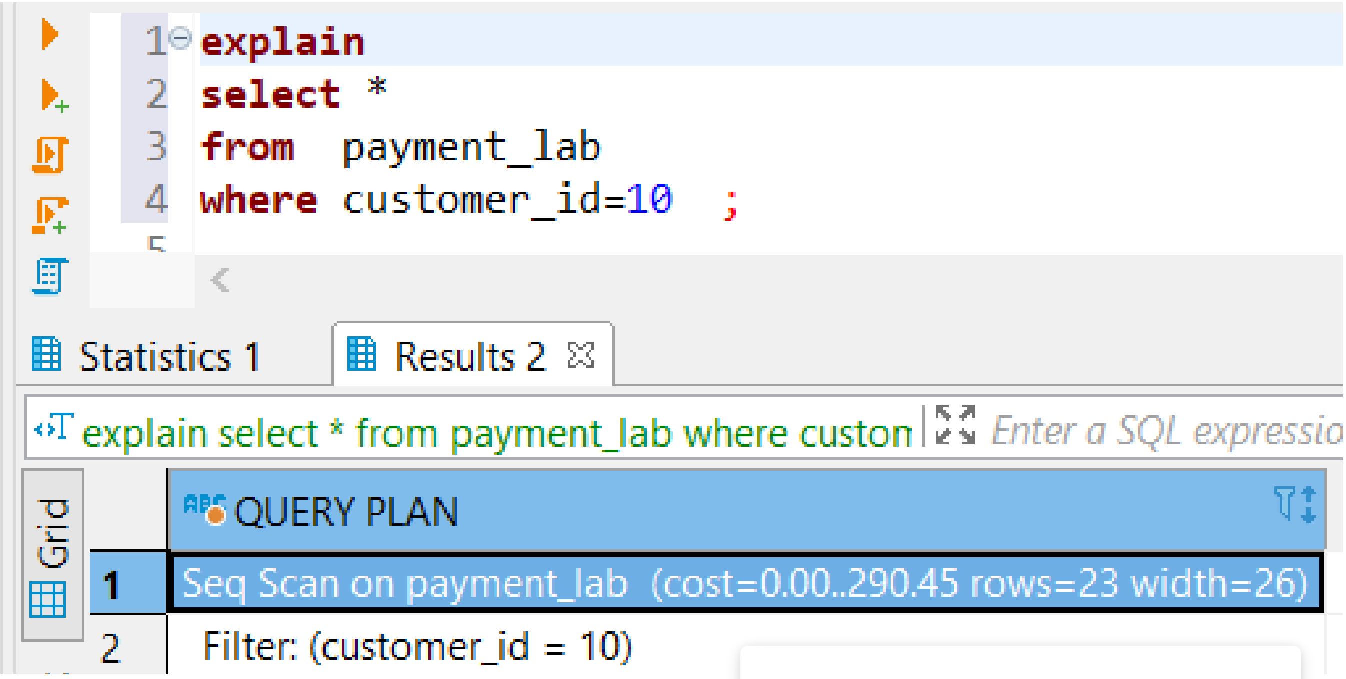
Book Author



Rizqi Mulki

Postgres Performance Tuning Like a Pro

The advanced techniques that separate database experts from everyone else



Muhammet Kurtoglu

Postgresql Query Performance Analysis

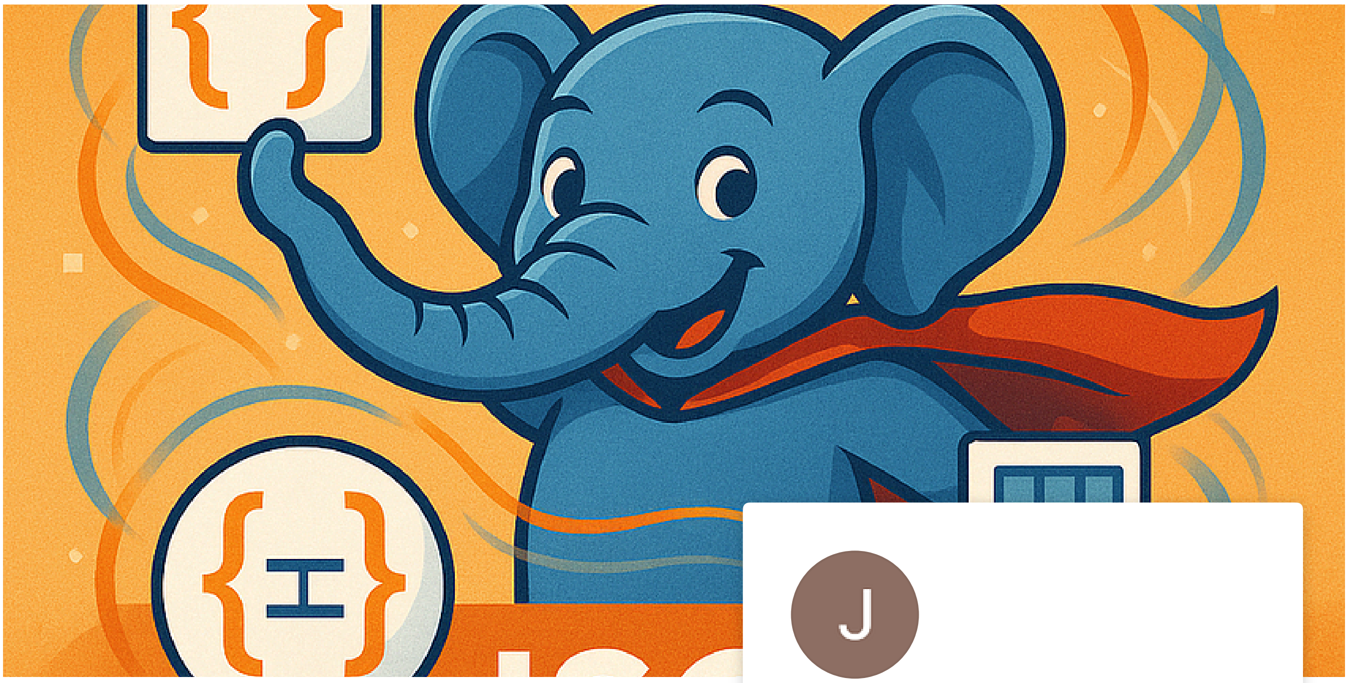
SQL tuning is critically important for ensuring databa ty.
In most cases, performance issues in a...

J
Jeyaram Ayyalusamy
76 followers
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance...
Book Author

Oz

Understanding PostgreSQL Page (Block) Structure

PostgreSQL stores data on disk in fixed-size units called pages or blocks. In this post, we'll explore how PostgreSQL organizes data...



Jeyaram Ayyalusamy

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

Book Author

Rick Hightower

JSONB: PostgreSQL's Secret Weapon for

Have you ever stared at your database schema and t ... n to this table, I'm going to lose it"? As you...

techWithNeeru

This SQL Query Took 20 Seconds. Now It Takes 20ms. Here's What I Changed

Last week, I was troubleshooting a performance issue that was driving our team crazy. Our analytics dashboard was timing out, users were...

Jul 10 66

See more recommendations

J

Jeyaram Ayyalusamy

76 followers

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

Book Author