

[Open in app ↗](#)

Search



★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Mastering PostgreSQL Configuration: A Deep Dive into postgresql.conf

10 min read · Jun 8, 2025



Jeyaram Ayyalusamy

Following

Listen

Share

More

PostgreSQL is one of the most powerful open-source databases in the world, but much of its true potential lies hidden in its configuration files. At the heart of PostgreSQL's server configuration sits a single, incredibly important file: `postgresql.conf`.

In this post, we'll break down everything you need to know about `postgresql.conf`, from its purpose to hands-on examples for modifying it — including full setup scripts to configure your PostgreSQL environment like a pro.



What is `postgresql.conf`? — The Heart of PostgreSQL Configuration

If you're working with PostgreSQL, you'll eventually meet one of the most important files behind the scenes:

`postgresql.conf` — the master configuration file that controls how PostgreSQL behaves.

Let's break it down in simple but powerful terms



📁 Why is postgresql.conf so important?

Every time your PostgreSQL server starts up, it reads the settings from this file to decide:

- How much memory to use
- How many connections to allow
- Which features to enable or disable

- 🚀 How to optimize for performance
- 📊 How much logging to produce
- 🌐 How to handle networking and authentication

In other words:

👉 *postgresql.conf is PostgreSQL's brain — it tells the server what to do.*

📍 Where to Find postgresql.conf in PostgreSQL

When you start tuning or troubleshooting your PostgreSQL server, one of the first tasks is to locate the `postgresql.conf` file — the primary configuration file controlling how PostgreSQL behaves.

Unlike some database systems that always store config files in the same place, PostgreSQL's location for `postgresql.conf` can vary depending on:

- 🔑 Your operating system (Linux, Windows, Mac)
- 📦 How PostgreSQL was installed (package manager, source build, containerized, managed cloud service)
- 💡 Custom configuration choices during installation

🔍 Typical Default Location on Linux

If PostgreSQL was installed using standard package managers like `apt` (Ubuntu/Debian) or `yum/dnf` (RHEL/CentOS), the `postgresql.conf` file is often located at:

```
/etc/postgresql/{version}/main/postgresql.conf
```

For example, on a PostgreSQL 16 installation, the full path might look like:

```
/etc/postgresql/17/main/postgresql.conf
```

🔍 Typical Default Location (Alternative Builds)

- On RPM-based distros (RHEL, CentOS, Rocky, AlmaLinux):

```
/var/lib/pgsql/{version}/data/postgresql.conf
```

- On Windows (default PostgreSQL installer):

```
C:\Program Files\PostgreSQL\{version}\data\postgresql.conf
```

- Inside Docker or container environments:

```
/var/lib/postgresql/data/postgresql.conf
```

🔍 How to Locate postgresql.conf Programmatically

Since location can vary, here are two reliable ways to find it no matter your environment:

✅ Method 1 — Using SQL (Always Works)

PostgreSQL itself knows where its configuration file is loaded from. You can ask it directly:

```
SHOW config_file;
```

Example Output:

```
/var/lib/pgsql/17/data/postgresql.conf
```

 **Pro Tip:** This is the most reliable method and works even inside remote or cloud-hosted PostgreSQL instances.

Method 2 — Using OS Shell Commands

You can search for the file directly at the OS level:

```
sudo find / -name postgre\*.conf
```

This searches your entire filesystem for any file starting with `postgre` and ending with `.conf`.

Example Output:

```
/etc/postgresql/17/main/postgresql.conf  
/var/lib/postgresql/17/main/postgresql.auto.conf
```

-  `postgresql.conf` : Main configuration file
-  `postgresql.auto.conf` : Automatically generated file for runtime configuration changes via `ALTER SYSTEM`

⚠ Why Knowing the Location is Important

- 🔧 To modify PostgreSQL behavior (memory, performance, replication, security)
- 🔒 To review security and access settings
- 🚀 To enable advanced features like `pg_stat_statements`, WAL archiving, or connection pooling
- 🐛 To troubleshoot performance issues



Summary Table

Environment	Likely Location
Ubuntu/Debian	<code>/etc/postgresql/{version}/main/postgresql.conf</code>
RHEL/CentOS	<code>/var/lib/pgsql/{version}/data/postgresql.conf</code>
Windows	<code>C:\Program Files\PostgreSQL\{version}\data\postgresql.conf</code>
Docker	<code>/var/lib/postgresql/data/postgresql.conf</code>
Any (Universal)	Run <code>SHOW config_file;</code> in <code>psql</code>

 **Remember:** Always make a backup before editing `postgresql.conf` and reload/restart PostgreSQL to apply changes.

⚙ Key Parameters in `postgresql.conf`: A Quick Guide for PostgreSQL Performance Tuning

The `postgresql.conf` file is the heart of PostgreSQL's configuration. With hundreds of tunable parameters, knowing which ones truly matter can save you hours of troubleshooting and greatly improve your database performance, reliability, and security.

Let's break down the most important settings — grouped into categories — so you know exactly what each one controls.

🔗 Connection Settings — Control How Clients Connect

1 max_connections

- **What it does:**

Defines the maximum number of concurrent client connections allowed.

- **Why it matters:**

Setting this too high can exhaust server memory; too low may reject client requests.

- **Example:**

```
max_connections = 200
```

2 statement_timeout

- **What it does:**

Terminates any SQL statement that runs longer than the specified time.

- **Why it matters:**

Prevents runaway queries from locking up resources.

- **Example:**

```
statement_timeout = 60000 # 60 seconds
```

3 ssl

- **What it does:**

Enables or disables SSL encryption for client connections.

- **Why it matters:**

Secures connections between clients and the database server.

- **Example:**

```
ssl = on
```

4 Resource Management — Control Memory & I/O Efficiency

4 shared_buffers

- **What it does:**

Allocates memory for PostgreSQL's internal cache of frequently accessed data.

- **Why it matters:**

One of the most impactful parameters for read performance.

- **Rule of thumb:** 15%–25% of total RAM.

- **Example:**

```
shared_buffers = 4GB
```

5 work_mem

- **What it does:**

Amount of memory used for complex operations like sorts and joins *per query*.

- **Why it matters:**

Affects performance for analytical queries.

- **Example:**

```
shared_buffers = 4GB
```

6 effective_io_concurrency

- **What it does:**

Tells PostgreSQL how many I/O operations can be performed in parallel.

- **Why it matters:**

Crucial for systems with SSDs or fast disks.

- **Example:**

```
effective_io_concurrency = 200
```

🚀 Query Performance — Optimize How Queries Are Executed

7 enable_seqscan

- **What it does:**

Allows or disallows sequential scans.

- **Why it matters:**

Sometimes disabling can help test index usage; generally best left enabled.

- **Example:**

```
enable_seqscan = on
```

8 autovacuum

- **What it does:**

Controls automatic cleanup of dead rows to prevent bloat.

- **Why it matters:**

Essential for healthy long-term performance.

- **Example:**

```
autovacuum = on
```

9 track_counts

- **What it does:**

Enables statistics collection for queries.

- **Why it matters:**

Required for performance monitoring tools like `pg_stat_statements` and autovacuum effectiveness.

- **Example:**

```
track_counts = on
```

Logging & Monitoring — Control What PostgreSQL Logs

10 log_destination

- **What it does:**

Sets where logs are written (e.g., stderr, csvlog, syslog).

- **Why it matters:**

Directs logs to the right place for analysis.

- **Example:**

```
log_destination = 'stderr'
```

1 1 log_statement

- **What it does:**

Controls which SQL statements are logged.

- **Options:** none, ddl, mod, all.

- **Example:**

```
log_statement = 'mod'
```

1 2 log_min_messages

- **What it does:**

Minimum severity level for messages to be logged.

- **Options:** DEBUG, INFO, NOTICE, WARNING, ERROR, FATAL, PANIC.

- **Example:**

```
log_min_messages = 'WARNING'
```

⌚ Replication Settings — Critical for High Availability & Disaster Recovery

1 3 wal_level

- **What it does:**

Sets how much information is written to the Write-Ahead Log.

- **Options:** minimal, replica, logical.

- **Why it matters:**

Required for streaming replication and logical replication.

- **Example:**

```
wal_level = replica
```

1 4 synchronous_commit

- **What it does:**

Controls whether transactions wait for WAL writes to complete.

- **Why it matters:**

Can improve performance but may trade off durability.

- **Example:**

```
synchronous_commit = on
```

1 5 max_wal_senders

- **What it does:**

Maximum number of concurrent WAL sender processes for replication.

- **Why it matters:**

Controls how many replicas you can maintain.

- **Example:**

```
max_wal_senders = 10
```



After changing `postgresql.conf`, always remember to reload or restart PostgreSQL for changes to take effect:

```
sudo systemctl reload postgresql
```

Why These Parameters Matter

-  Tuning these settings can dramatically improve query speed and resource usage.
-  Help you secure connections and control user access.
-  Allow better monitoring, logging, and troubleshooting.
-  Enable replication for high availability systems.

By mastering just these key parameters, you'll already be ahead of 80% of PostgreSQL admins — and your database will thank you for it.

How to Modify `postgresql.conf` in PostgreSQL

In PostgreSQL, the `postgresql.conf` file is the central configuration file that controls how your PostgreSQL server behaves. It includes critical parameters that affect performance, connections, security, logging, and more. Whenever you need to fine-tune PostgreSQL, you usually modify this file.

Let's walk through exactly how to modify `postgresql.conf` safely.

Step 1 — Edit the postgresql.conf File

The first step is to locate and open the file for editing. Depending on your PostgreSQL version and installation method, the file path may vary. For PostgreSQL 16 installed on a typical Linux system, it's often located here:

```
sudo vi /etc/postgresql/16/main/postgresql.conf
```

 If you're not comfortable with `vi`, you can also use `nano`:

```
sudo nano /etc/postgresql/16/main/postgresql.conf
```

Example Configuration Change

Suppose you want PostgreSQL to accept connections from any IP address (which is often required for remote access). You would change the following setting inside `postgresql.conf`:

```
listen_addresses = '*'
```

- By default, `listen_addresses` may be set to `localhost`, which only allows local connections.
- Changing it to `'*'` allows connections from any IP address (assuming you also properly configure `pg_hba.conf`).

 **Security Note:** Allowing all IPs requires additional network and firewall security. Only open PostgreSQL to trusted networks.

Step 2 — Backup the Configuration File (Best Practice)

Before making any changes to PostgreSQL configuration files, **always create a backup**. This ensures you can easily restore the previous working configuration if something goes wrong.

Run:

```
cp /etc/postgresql/16/main/postgresql.conf /etc/postgresql/16/main/postgresql.c
```



- This command creates a backup copy of the config file with today's date attached (using `date +%Y%m%d`), making it easy to identify when the backup was made.

Step 3 — Apply Changes by Restarting PostgreSQL

After modifying `postgresql.conf`, you need to restart PostgreSQL to apply the new configuration:

```
sudo systemctl restart postgresql
```

 **Tip:** After restarting, you can verify the server status to ensure everything started correctly:

```
sudo systemctl status postgresql
```

If there's a configuration error, PostgreSQL may fail to start, and you'll see helpful error messages in the log files (usually in `/var/log/postgresql/`).



Summary Checklist

Step	Action
1	Open and edit <code>postgresql.conf</code>
2	Backup before changes
3	Restart PostgreSQL

By following these steps carefully, you can modify PostgreSQL's behavior confidently and safely without risking server downtime.

🔗 Dynamic Configuration Changes in PostgreSQL Using `ALTER SYSTEM`

PostgreSQL offers a powerful feature that allows you to change many configuration settings **dynamically**, without directly editing the `postgresql.conf` file. This is done using the SQL command `ALTER SYSTEM`. It simplifies configuration management, especially for DBAs who prefer making changes directly from a SQL client like `psql`, `pgAdmin`, or automation scripts.

🚀 Why Use `ALTER SYSTEM`?

- You don't need shell or file system access to the server.
- Helps automate configuration changes.

- PostgreSQL validates the setting immediately.
- Minimizes manual file editing errors.

 Note: You still need sufficient PostgreSQL privileges to run `ALTER SYSTEM` (typically as the `postgres` superuser).

How to Use `ALTER SYSTEM`

Example: Change PostgreSQL Port

Let's say you want PostgreSQL to listen on port `5433` instead of the default `5432`. You can execute:

```
ALTER SYSTEM SET port = 5433;
```

- This writes the new setting into PostgreSQL's `postgresql.auto.conf` file (not directly into `postgresql.conf`).
- PostgreSQL will use this value on the next restart.

Apply Changes

Many `ALTER SYSTEM` changes require a server restart to take effect (especially for parameters like `port`, `shared_buffers`, etc.).

You can restart PostgreSQL with:

```
sudo systemctl restart postgresql
```

Always verify that PostgreSQL restarted successfully:

```
sudo systemctl status postgresql
```

🔍 Verify the Change

After restarting, you can check the current value using:

```
SHOW port;
```

This should return:

```
5433
```

♻️ How to Reset Settings

If you want to undo changes made via `ALTER SYSTEM`, you can reset individual settings or all settings:

✓ Reset a Single Setting

```
ALTER SYSTEM RESET port;
```

✓ Reset All Settings at Once

```
ALTER SYSTEM RESET ALL;
```

This clears all changes from `postgresql.auto.conf` and reverts to the original settings defined in `postgresql.conf` (or compiled defaults).

Behind the Scenes

- `ALTER SYSTEM` updates a separate file called:

`postgresql.auto.conf`

- Usually located inside your data directory.
- This file takes precedence over `postgresql.conf` during startup.
- Manual edits to `postgresql.auto.conf` are discouraged — always use `ALTER SYSTEM` for safe changes.

Key Benefits of `ALTER SYSTEM`

Benefit	Description
<input checked="" type="checkbox"/>	Centralized All changes managed from SQL clients
<input checked="" type="checkbox"/>	Safer PostgreSQL syntax validation
<input checked="" type="checkbox"/>	Easier Automation Useful for scripting/admin tools
<input checked="" type="checkbox"/>	No File Access No need to SSH into the server

Final Thoughts

`ALTER SYSTEM` is a DBA-friendly feature that makes PostgreSQL configuration safer and more flexible — especially in production environments where direct file access may be restricted. However, always document changes and test carefully before applying in mission-critical systems.

Temporary Session-Level Configuration Changes in PostgreSQL

PostgreSQL offers incredible flexibility when it comes to fine-tuning performance — and not all configuration changes require file edits or server restarts. In many cases, you may want to adjust settings **only for the duration of your current database session**. This is where session-level changes come into play.

What Are Session-Level Changes?

-  Changes apply only to your current database connection.
-  They do **not** affect other users or connections.
-  When you disconnect or close the session, the changes are automatically discarded.
-  Great for testing or temporarily optimizing certain queries.

How to Apply Session-Level Changes

PostgreSQL makes this very simple using the `SET` command.

Check the Current Value

Before making any change, it's good practice to check the current value of the parameter you're about to modify.

For example, to check the amount of memory allocated for sorting operations (`work_mem`):

```
SHOW work_mem;
```

Typical output might be:

```
4MB
```

Apply a Temporary Change

Let's say you want to increase `work_mem` to 20MB just for the current session:

```
SET work_mem = '20480kB';
```

Or equivalently:

```
SET work_mem = '20MB';
```



PostgreSQL supports both kilobytes (kB) and megabytes (MB) units.

Verify the New Setting

After applying the change, you can verify that it has taken effect:

```
SHOW work_mem;
```

Output:

```
20MB
```

- ✓ Done! The new setting is now active **only for your session**.

📝 Important Notes

- 🔒 Other users and sessions are completely unaffected.
- ⌚ The change lasts only as long as the session is open.
- 🔌 After disconnecting, the setting automatically reverts to the system default (`postgresql.conf` value).

🔍 Use Cases for Session-Level Changes

Scenario	Why Session-Level Change Helps	Query optimization	Temporarily allocate more memory for a complex sort or join	Testing	Safely evaluate new configuration settings	Development	Experiment with tuning parameters without impacting others	ETL/Batch Jobs	Allocate more resources for large one-time jobs
----------	--------------------------------	--------------------	---	---------	--	-------------	--	----------------	---

🚫 When NOT to Use

- ✗ Do not rely on session-level changes for permanent performance tuning.
- ✗ Do not use for changes that require consistency across multiple sessions.
- ✗ For long-term changes, modify `postgresql.conf` or use `ALTER SYSTEM`.

🏁 Summary

- Session-level changes in PostgreSQL give you **quick, isolated control** over performance parameters.
- Use `SET` to modify, and `SHOW` to verify.
- Changes automatically disappear at session end — no risk to global settings.

This flexibility makes PostgreSQL ideal for both production DBAs and developers who need fine-grained control during query tuning or troubleshooting.

🚀 Conclusion

The `postgresql.conf` file is your gateway to unlocking PostgreSQL's full power. Whether you're tuning for performance, enabling replication, or adjusting security settings — understanding this file gives you full control over your PostgreSQL database environment.

By learning how to locate, edit, and manage `postgresql.conf`, you can transform your PostgreSQL server into a finely tuned, high-performing data engine.

🔗 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

MySQL

Mongodb

Sql

J

Following ▾

Written by Jeyaram Ayyalusamy

50 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

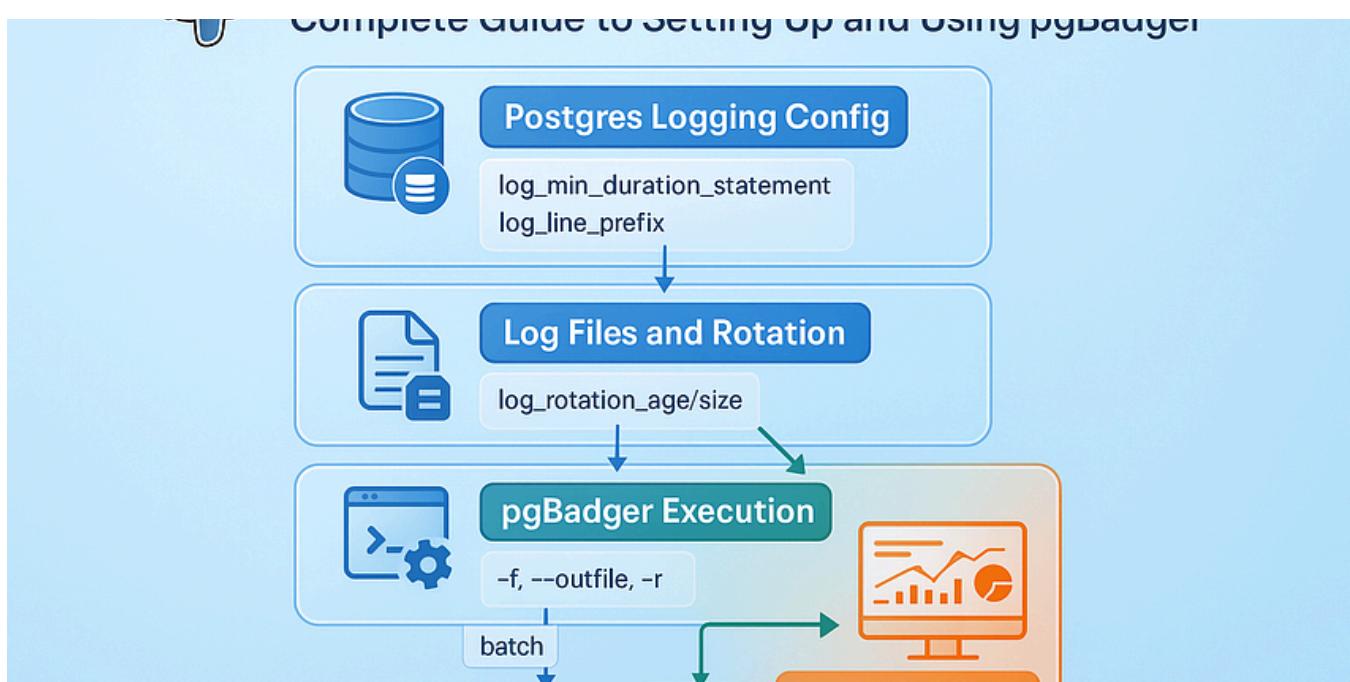
No responses yet



Gvadakte

What are your thoughts?

More from Jeyaram Ayyalusamy



J Jeyaram Ayyalusamy 

PostgreSQL 17 Log Analysis Made Easy: Complete Guide to Setting Up and Using pgBadger

Managing a production-grade PostgreSQL instance isn't just about keeping data safe—it's about monitoring performance, identifying...

Jun 23  52

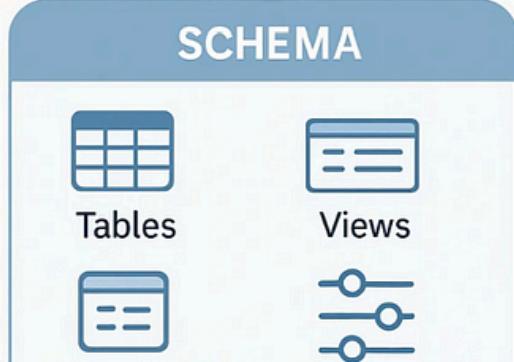


...

A Deep Dive into How PostgreSQL Organizes Data

① SchemaSQL: Logical Structure

Every PostgreSQL database can contain multiple schemas.



PostgreSQL Physical Structure



PostgreSQL Physical Storage

- Database data files
- Transaction logs
- System metadata
- Cluster configuration
- Replication and recovery state

J Jeyaram Ayyalusamy 

The Internal Structure of PostgreSQL: A Deep Dive into How PostgreSQL Organizes Data

PostgreSQL is one of the most powerful and popular open-source relational database systems used in production today. But while most people...

Jun 1  2



...



J Jeyaram Ayyalusamy

PostgreSQL 17 Administration: Mastering Schemas, Databases, and Roles

PostgreSQL has evolved into one of the most feature-rich relational databases available today. With version 17, its administrative...

Jun 9 3



...



J 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...

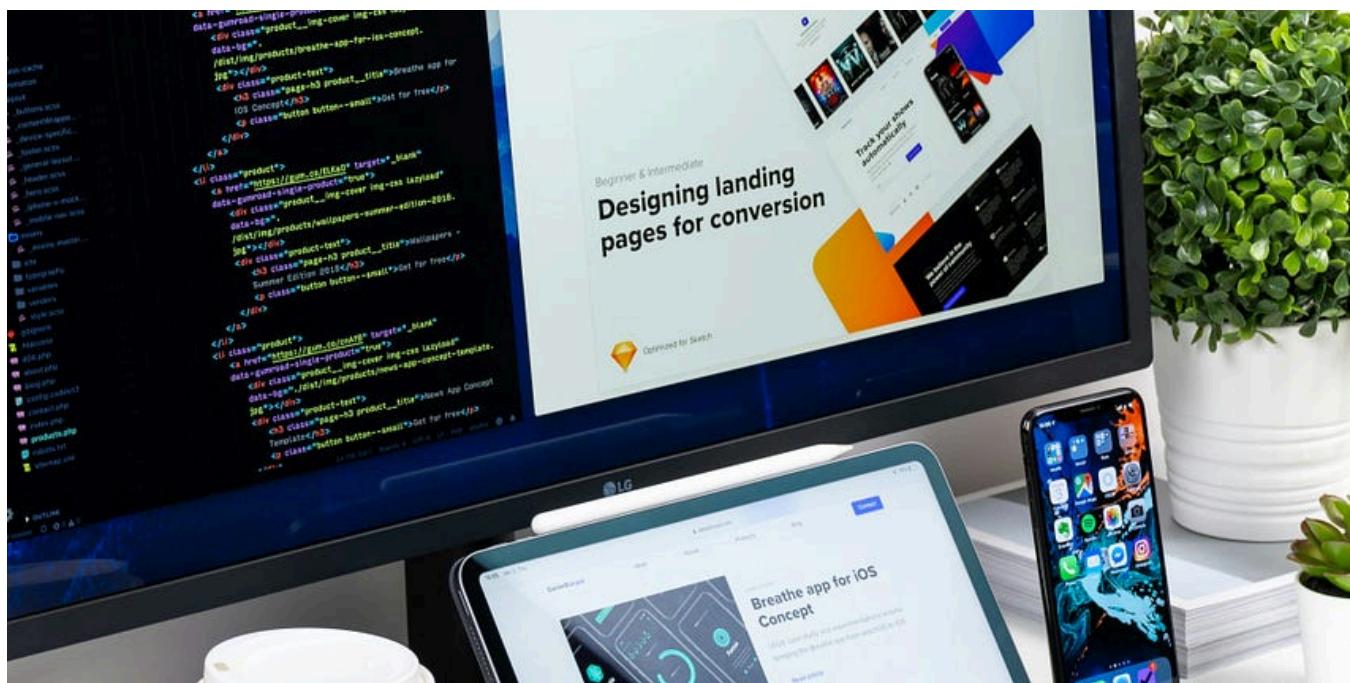
Jun 3



...

See all from Jeyaram Ayyalusamy

Recommended from Medium



 Sandesh | DevOps | CI/CD | K8s

“Why PostgreSQL is Beating MongoDB in 2025 (And Why I Switched Back)”

(Spoiler: My NoSQL experiment cost us 3 months of debugging headaches)

 Jun 11

 10

 1



...

View Table

Purpose	Data Distribution	Use Case	Key Benefit
Scale horizontally	Split across servers	Huge user base (e.g. Twitter)	Handle large data volumes
Increase availability & backup	Duplicate data	Read-heavy apps (e.g. blogs)	High availability and tolerance
Manage large tables efficiently	Split within one server	Time-series logs	Better performance for large tables

 Ravi Jaisawal

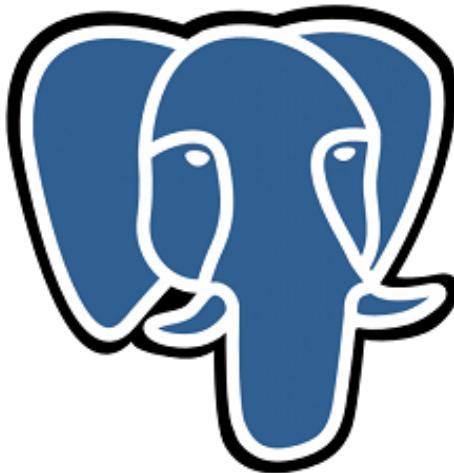
Sharding vs Replication vs Partitioning in Databases

Differences between Sharding, Replication, and Partitioning in databases — with practical examples and real-world use cases

Jun 11  4



...



Postare.

 Sohail Saifi

Postgres Hidden Features That Make MongoDB Completely Obsolete (From an Ex-NoSQL Evangelist)

For six years, I was that developer.

May 26

200

11



...



Deniz GÜRSOY

Select For Update in Postgres

The SELECT FOR UPDATE statement enables you to acquire exclusive locks on rows within PostgreSQL databases. An exclusive lock ensures that...

Jun 25

1



...



ThreadSafe Diaries

PostgreSQL 18 Just Rewrote the Rulebook, Groundbreaking Features You Can't Ignore

From parallel ingestion to native JSON table mapping, this release doesn't just evolve Postgres, it future-proofs it.

Jun 27 34



ACID Compliance	Yes (strong data integrity)	Yes (since 4.0, at document level, and with transactions)
Schema Flexibility	Strict schema	Flexible schema
Scalability	Vertical and horizontal	Horizontal
Query Language	SQL (declarative)	MongoDB Query Language (MQL)
Transactions	Full ACID transactions	Multi-document transactions (ACID compliant since 4.0)
Performance	Optimized for complex queries and joins	Optimized for high-throughput and simple queries
Community & Support	Large and mature community, enterprise support	Large and active community, enterprise support
Typical Use Cases	Financial apps, data warehousing, complex business logic	Content management, real-time analytics, mobile apps, IoT

A Advaitlachake

vs PostgreSQL vs MongoDB: Which Database Should You Choose in 2025?

In the ever-expanding world of data, your database choice can define your product's future.

Jun 26



See more recommendations