Replication Types and Modes in PostgreSQL

Data is a key part of any mission-critical application. Losing it can lead to serious issues, such as financial loss or harm to a business's reputation. A common way to protect against data loss is by taking regular backups, either manually or automatically. However, as data grows, backups can become large and take longer to complete. If these backups are not managed properly or tested often, a sudden system crash could result in permanent data loss.

To address this, many organizations use real-time data replication as a more reliable solution. In this setup, data from the primary server is continuously copied to one or more standby servers. If the primary server fails, there's no need to restore from a backup the standby server can be quickly promoted to take over as the new primary. This allows the application to resume with minimal downtime, reducing the Recovery Time Objective (RTO), and can bring the Recovery Point Objective (RPO) close to zero or even zero.

PostgreSQL also supports replication to keep standby servers in sync with the primary server using Write-Ahead Log (WAL) files. Every change made to the database is first recorded in these WAL files on the primary server. These logs are then continuously streamed to the standby server, which applies them to stay up to date. This method ensures that all standby servers stay in sync with the primary and are ready to be promoted in case the primary server fails.

In this blog, we will explore the different types and modes of replication available in PostgreSQL to help you understand which option best fits your business needs.

Types of Replication

PostgreSQL offers different types of replication, with each method designed to serve specific use cases.

- Physical Replication
- Logical Replication
- Cascading Replication

Physical Replication

Physical replication is the foundation for setting up high availability in active-passive clusters. In this replication, the primary server contains the complete data directory, which includes all database files and to set up a standby server, this data directory is copied from the primary server, typically using native tools like pg_basebackup. On the standby server, a few configuration settings are updated to identify it as a standby and to specify the primary server it should connect to. Once configured, the standby server is started and begins receiving data from the primary.

When changes occur on the primary, they are first written to Write-Ahead Log (WAL) files. These WAL files are then streamed to the standby server, which replays them to stay in sync with the primary server.

To manage WAL retention and ensure the primary doesn't remove WAL files before a standby has received them and actually replayed them too, replication slots are being used, which are created on the primary server and track how far each connected standby has replayed the WAL files. This prevents premature WAL file removal and ensures smooth, continuous replication even if the standby is temporarily disconnected and replication lag is generated.

Depending on the configured mode, synchronous or asynchronous (discussed in the next section), the primary may wait for confirmation from the standby before finalizing a transaction. This replication process enables near real-time data syncing and helps us minimize the data loss in real-world scenarios.

Note: In physical replication, standby servers are read-only. You cannot perform any write operations (INSERT, UPDATE, DELETE) on them. All write activity must go through the primary server. Also, physical replication cannot be used between different major PostgreSQL versions.

Physical (or streaming) replication in PostgreSQL offers two powerful modes

- Synchronous
- Asynchronous Replication
  These modes provide the flexibility to configure your setup based on your business needs, which allow you to balance between performance and data durability. you can control how much (if any) data loss is acceptable during a failure, with the help of the right mode

Synchronous (Physical/Streaming) Replication

In synchronous mode, data integrity is ensured by a confirmation process between the primary and standby servers

- When an application or user performs a write operation (INSERT, UPDATE, DELETE), the change is first written to a WAL file on the primary server.
- This WAL data is immediately streamed to the standby server.
- Once the standby receives the WAL data, it sends a confirmation back to the primary.
- Only after receiving this confirmation does the primary commit this change locally.
- Once this change is committed locally on the primary it then sends a success message to the application, confirming that the change has been safely recorded.
  This process is specifically designed to achieve an RPO (Recovery Point Objective) of zero, ensuring no data loss, even in the event of a failure.

Note: If multiple standby servers are connected to the primary, the primary will wait for a confirmation from one or more specific standby servers, depending on how the synchronous_standby_names setting is configured. Only after receiving the required confirmation(s) will the primary commit the change and acknowledge it to the application. To learn more about synchronous_standby_names please use the following link

Asynchronous Replication

In asynchronous mode, performance is prioritized over guaranteed data durability across nodes.

- When an application or user performs a write operation (INSERT, UPDATE, DELETE), the change is first written to a WAL file on the primary server.
- The primary server immediately commits the transaction locally
- The WAL data is streamed to the standby server.
- The primary server sends a success message back to the application without waiting for any confirmation from the standby server.
- The standby receives and applies these WAL files to stay as up-to-date as possible, but there may be a slight delay depending on network latency and system load.
  This setup results in a non-zero RPO, meaning a small amount of data loss is possible during a failover.

Note: Asynchronous replication is the default mode in PostgreSQL.

Quick Comparison: Synchronous vs Asynchronous

Feature	Synchronous Replication	Asynchronous Replication
Commit	The primary server	The transaction on the primary

Behaviour	confirms and commits data only after receiving approval from the standby server.	server is committed immediately after the WAL is sent to the standby server, without waiting for its approval.
Data Consistency	No data is lost even if the primary server crashes, ensuring strong consistency.	Possible data loss if primary server crashes
Performance	Performance drops due to standby approval delays.	Improved performance as the primary server doesn’t wait for standby approval.
Replication Lag	No replication lag from primary to standby	Low to high depending on several factors including latency etc
Use cases	Mission-critical apps: Banking, Finance, e-Commerce Payments	High-read, moderate-write apps: Analytics, News, E-learning

Logical Replication

As we know, Physical replication allows us to replicate an entire server to a standby means each database will be replicated, ensuring complete redundancy and data availability. However, what if we only want to replicate specific parts of a database such as selected rows or columns from a table?

This is where logical replication comes into play. logical replication offers the flexibility to replicate only the desired portion of data that you wish to back up or preserve.

It operates using a publisher-subscriber model, where the publisher (also known as the primary server) shares specific data, and the subscriber (the standby server) receives and maintains that subset. Also, a single publisher can have multiple subscribers, which allows the data to be replicated to multiple servers as needed.

Note: Logical replication can be configured across different PostgreSQL versions for example, replicating data from PostgreSQL 13 to PostgreSQL 15 making it a useful option for major version upgrades and cross-version data migration.

Cascading Replication

This is another supported method for replication in PostgreSQL in which the primary server writes data and logs changes in a WAL (Write-Ahead Log) file. This WAL file is first sent to Standby Server one, which applies the changes. Once updated, Standby Server One then forwards the same WAL file to Standby Server Two, and the process continues sequentially through the chain of standby servers. This method of replication is asynchronous, meaning the primary server does not wait for confirmation from the standby before writing changes locally.

Cascading replication ensures strong data backup across multiple servers and also distributes the load, preventing the primary server from becoming a bottleneck. Each standby server plays a dual role in receiving data from the previous node and forwarding it to the next until the final standby server in the chain has applied the changes. This structured and scalable approach enhances both performance and reliability in database systems.

Quick Comparison of Replication Types in PostgreSQL

Feature	Physical Replication	Logical Replication	Cascading Replication
Replication Type	Binary (file-level, WAL-based)	Row-level (table/data-based)	Binary (same as physical)
Cross-Version Support	No, Same major version only	Yes, Supports cross-version	No, Same major version only
Write Access on Replica	No, Read-only	Yes, Can write to subscriber tables	No, Read-only
Use Case	High availability, disaster recovery	Selective replication, data sharing	Load distribution from standby to other standbys
Sync/Async Mode	Yes, Supports both	No, Asynchronous only	No, Asynchronous only