1. **What is a Data Base Backup?**

A.

* A copy of your database that can be used to restore data if the original is lost or corrupted.

Key Characteristics

* Not the same as replication (backups are static snapshots)
* Not automatic - must be configured deliberately
* Should be stored separately from the live database

Uses:

* Data loss prevention (hardware failure, human error)
* Disaster recovery (fires, floods, ransomware)
* Regulatory compliance (many laws require backups)
* Migration/testing (copy data to new environments)

1. **What are the Different Types of Backups?**

A.

1. Physical Backups
2. Full backups: Complete copy of all database files
3. Incremental backups: Only changes since last backup (full / incremental)
4. Differential / Cumulative backups: All changes since last full backup

Tools: pg\_basebackup, filesystem snapshots, Barman, WAL Archiving

1. Logical Backups

* Extract data as SQL statements or delimited format
* Does not include transaction logs (WAL), physical storage details
* Can restore to different postgre SQL versions

Tools: SQL dumps (e.g., mysqldump, pg\_dump), Export/import utilities

1. Snapshot-Based Backups

* Storage-level snapshots (LVM, SAN, VM)
* Point-in-time copies of database volumes
* pgBackRest or Barman that support both physical and logical backups

1. **What are Physical backups?**

A.

* Physical backups involve copying the actual files that PostgreSQL uses to store your database.
* Copy the actual database files like data files, WAL files, configuration files, tablespace files, control files, archived redo logs many more instead of logical SQL dumps
* Physical data backup is kept in the cloud, offline storage, magnetic tape or on a disc.
* It is beneficial to provide details of transactions and changes made in databases.
* Faster to restore especially large databases
* Supports PITR
* Must restore to same postgre SQL version
* Requires more storage than logical backups
* This slows down database operations
* Entire Database Cluster Will Be Backed Up, not database or schema specific
* server has to be stopped while restoring
* cluster wise backup & restore

There are two methods to perform a physical backup:

1. Operating system utilities
2. Recovery manager

Physical Backup & Restore Process:

Backup process:

Postgre SQL uses WAL to ensure integrity

* Base backup = full copy of data files
* Wal files = transaction logs

Restore process:

* Restore base backup
* Replay WAL logs to reach desired state

1. **Full Base Back Up:**

* As the name implies, backs up each and every file and folder (hard drives and more) on the system.
* Complete copy of all database files.
* Foundation for incremental / differential backups
* It needs large storage, and time consuming
* Simple and fast to restore
* Using pg\_basebackup we will back up the Full Base Backup
* If we want to continue with Incremental or differential backups then we have to made some changes in this process
* Accepting Only Plain, Tar formats

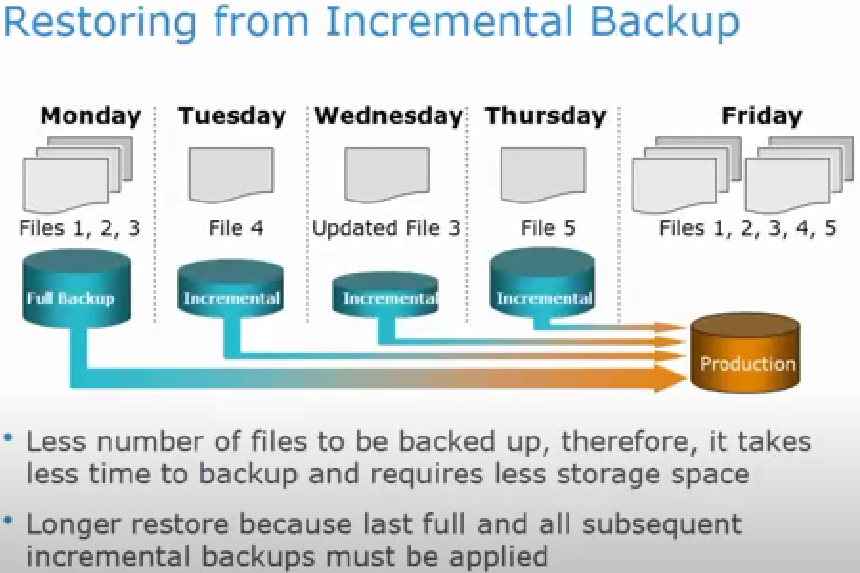
Ex: pg\_basebackup -U postgres -D "C:\DB\_Backup" -F p -X stream -P -v

1. **Incremental Back Up:**

* Backups Only changes since last backup (full / incremental)
* it only includes the data that has changed since the last backup
* Requires continuous WAL archiving
* Smaller, faster
* Restore = full backup + all incremental backups in correct order
* WAL and differential backups do not need to be stored until next full backup, because all the changes will reflect in the next full backup \*\*\*

Ex: pg\_basebackup --incremental="C:\DBFullBackup\backup\_manifest" -D "C:\BDIncBackup" -U kjs

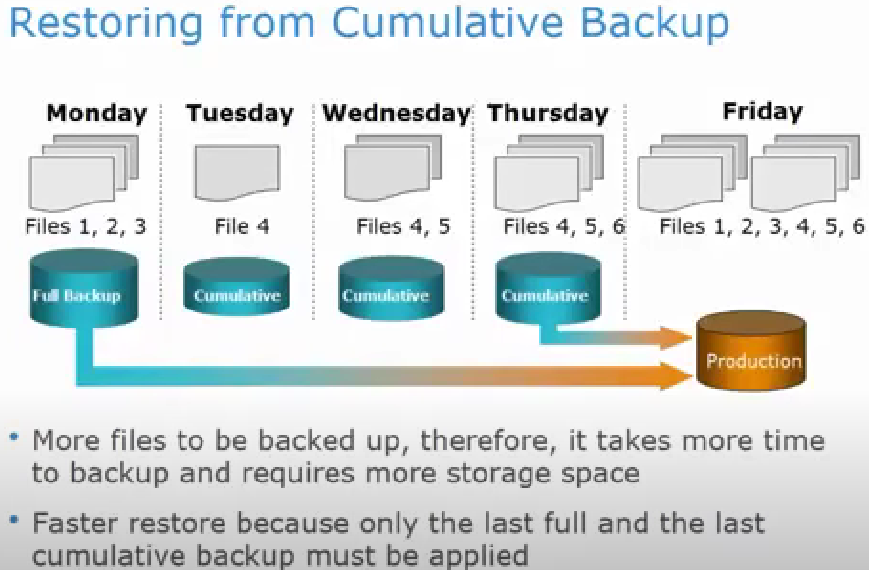
a company did full back up on Monday, and did incremental backups from Tuesday to Friday, so Tuesday will back up only Tuesday’s data.



1. **Differential / Cumulative Back Up:**

* These backups begin with a full backup, which saves all of your files. Then differential backups are performed, which save only the data that has changed since the previous full backup.
* It backups only save changes to files that changed as compared to the previous backup and it also makes a copy if new files if any.
* It also provides for a speedier restore time than incremental although it requires more storage space.
* Backs up the data that has been changed since last full backup
* Requires continuous WAL archiving since last full backup.
* Restore = Full + last differential backup.

Ex: a company does a full back up on Monday, and did differential backup from Tuesday to Friday, as data being added to database on these week days, the differential backup will back up the data that has been changed since Monday’s full back up, so Tuesday will have Tuesday’s data, Wednesday will have Tuesday and Wednesday data, Thursday will have Wednesday (Tue + Wed) and Thursday data, Friday will have Thursday (Tue + Wed + Thu) and Friday’s data



1. Network Backup:

* It backs up a file system from one machine onto a backup device connected to another machine.
* It is referred to as a remote or network backup.

1. Transaction Log Backup:

* Every individual change recorded
* Every 15-60 minutes
* Point-in-time recovery possible
* Complex management

Ex: BACKUP LOG SalesDB TO DISK = 'C:\Backups\SalesDB\_Log.trn' \*

|  |  |  |
| --- | --- | --- |
| Back up Type | Data that backs up | Restore procedure |
| Full | All data (Longest) | Full back up only (Fastest) |
| Incremental | Data that has been changed since the last full or incremental back up (fastest) | Full and incremental backups in the correct order (Longest) |
| Differential | Data that has been changed since the last full backup (medium) | Full and last differential (medium) |

Differential ≠ Incremental

* Incremental: Changes since last any type of backup
* Differential: All changes since last full backup

Practical Example

* Suppose you take a full back up on Sunday.
* On Monday:
  + Incremental backup copies only Monday’s changes since Sunday.
  + Differential backup copies all changes from Sunday to Monday (same as incremental in first backup).
* On Tuesday:
  + Incremental backup copies only Tuesday’s changes since Monday’s backup.
  + Differential backup copies all changes since Sunday (i.e., Monday and Tuesday's changes).
* On Wednesday:
  + Incremental backup copies only Wednesday’s changes since Tuesday.
  + Differential backup copies all changes since Sunday (Monday, Tuesday, Wednesday).

1. **What Is Logical Backup?**

A.

* It contains logical data which is retrieved from the database. It contains a view, procedure, function, and table.
* This is useful When users want to restore or transfer a copy of the database to a different location
* Logical backups are not as secure as physical backups in preventing data loss.
* It only provides structural details
* Logical backups are used as a supplement to a physical backup
* This is useful when the user needs to restore the complete database to the last time.
* It was more complex and provides granular recovery capabilities.
* Slow for large databases, does not capture transactions
* We can backup only structure, only data or both structure and data
* We Can backup entire database or only specific schema from the DB
* We can do it by CMD, GUI
* No need to stop the server while restoring the data
* Used in migrations

BACKUP: pg\_dump, pg\_dumpall

RESTORE: psql (for plain SQL dumps), pg\_restore (for custom or tar format)

1. **What are the basic backup methods?**

A.

1. Native Tools: Built-in utilities

Ex: mysqldump, pg\_dump

1. GUI Tools: Visual interfaces for backup/restore

Ex: MySQL Workbench, DBeaver

1. Schedule Jobs: Automated backups via:

Ex:

* Cron jobs (Linux)
* Task Scheduler (Windows)
* Database agent jobs

1. **What is pg\_basebackup?**

A.

* pg\_basebackup is a PostgreSQL command-line tool used to take a full, physical backup of a running PostgreSQL database cluster while the server is online and accessible to other users
* Creates a physical (binary) backup of the entire database cluster (i.e., all databases managed by one server instance)
* You can take the backup while PostgreSQL is running and serving clients
* The tool puts the server in backup mode automatically, makes sure the backup is consistent, and manages transaction logs (WAL files) as needed
* You can choose to include Write-Ahead Log files (WAL) in the backup, enabling point-in-time recovery (PITR).

It will located in : "C:\Program Files\PostgreSQL\17\bin\pg\_basebackup.exe"

Limitations:

* Only full cluster backups; you cannot use it for individual databases or tables (use pg\_dump for that)
* Strictly works at the instance (cluster) level, no table-level or schema-level backups

Process:

* In the entire process we will take a back into a folder and we will replace it with data folder in postgre SQL
* First, we will restore base backup file and then WAL files if present
* If your backup is a tar archive (e.g., backup.tar), extract it:
* Use a tool like 7-Zip or WinRAR to extract the contents into your data directory.

1. **What is pg\_dump?**

A.

* pg\_dump is used to take a backup (dump) of a single PostgreSQL database. The output is a logical file that contains SQL commands—or a special binary format—that can recreate the database’s schema and data.
* Can output as plain text SQL files or custom, directory, and tar formats
* Plain sql is Easy to edit/read with a text editor; restore with psql.
* Custom/Directory/Tar are Compressed formats; restores via pg\_restore for more control.
* Can back up individual tables, schemas, or exclude certain objects.
* Can Restore to another server/version

1. **What is pg\_dumpall?**

A.

* pg\_dumpall is used for a cluster-wide backup, meaning it dumps all databases in your PostgreSQL server instance at once, along with all global objects (roles, tablespaces, and configuration settings).
* pg\_dumpall will back up only one PostgreSQL server’s cluster at a time, not all clusters across multiple servers within a network or management tool.
* Only outputs as plain text SQL (no custom, directory, or tar).
* Can Restore to another server/version

RESTORE: pg\_restore

1. **What is pg\_restore?**

A.

* Used to restore a PostgreSQL database from an archive file created by pg\_dump in one of the non-plain-text formats (custom, directory, tar).
* It can selectively restore specific database objects (tables, schemas, etc.) from the backup
* Supports options to create the database (-C), clean (drop objects before restoring, -c), and restore in parallel (-j).
* If the backup is a custom or directory format dump, pg\_restore is the appropriate tool.

RESTORE: pg\_restore - for pg\_dump custom or tar or directory format, parallee jobs

1. **What is psql?**

A.

* Used to execute SQL scripts or plain-text dumps generated by pg\_dump with the plain text format
* It restores a database by executing the SQL commands contained in the dump file.
* It cannot selectively restore parts of the dump; it simply runs all SQL statements sequentially.
* Usually simpler but less flexible and may be slower for large databases since it does not support parallel restore.
* Suitable for plain-text SQL backup files or when the entire dump is to be restored as-is.

RESTORE: psql - for plain pg\_dump, pg\_dumpall

This distinction is important because pg\_restore cannot handle plain SQL dumps, and psql does not understand archive formats.

1. **What are the Backup components?**

A.

1. Data Files:

* Contains actual table data
* Largest part of the backup

Ex: .mdf (SQL Server), .idb (My SQL)

1. Transaction Logs:

* Records all changes
* Enables point-in-time recovery

Ex: .ldf (SQL Server), ib\_logfile (My SQL)

1. Schema / Structure:

* Database design (tables, relationships)
* Often stored separately from data
* Crucial for rebuilding the database

1. **What are Backup Devices?**

A.

1. CD / DVD:

* Because they have a small capacity ranging from 1000's of MB to a few GB, they are utilized for home/personal use where users can save their papers, primarily personal or office-related papers.

1. USB Sticks:

* USB sticks are small in size and cost, but they're big in storage capacity; you can obtain up to 128 gigabytes on a USB stick.
* They are small in size but have a good transfer speed.

1. USB Drivers:

* This sort of drive has a size range of 500MB to 2TB and is compatible, and normally includes backup and recovery software.
* Encryption, convenient automatic backups, and a cloud backup option are all included in many models.
* The cost is high.

1. SSD:

* They are more expensive than hard drives, but they're also more reliable, smaller, faster, and consume less power
* SSDs are ideal for applications where a speed improvement is worthwhile, such as system files or multimedia production.

1. NAS (Network-Attached Storage):

* A network-attached storage is a file storage device that delivers centralized, consolidated disc storage to local-area network (LAN) users via a normal Ethernet connection.
* NAS allows a network with servers to add more hard disc storage capacity without having to shut them down for maintenance and updates.

1. **What is Data Recovery?**

A.

* Data recovery, often known as a restore, is required when data of any sort is no longer readable or has been corrupted by a malicious alteration.
* The act, process, or occurrence of recovering data following inadvertent loss or corruption is known as recovery.
* The cost of recovering data is high.

 The following are the most prevalent reasons for data loss:

1. Virus/Spyware/Malware attack.
2. Natural calamities
3. Hardware Failure
4. Human errors.
5. Manipulation in software etc.
6. **Essential Terminology**

A.

1. Recovery Time Objective (RTO):

* How fast you can recover
* Determines backup frequency

1. Recovery Point Objective (RPO):

* How much data you can lose
* Affects backup type choice

1. Hot OR Online Backup:

* Backup while database is running
* Application is up and running, with users accessing their data during backup
* Open file agent can be used to backup open files
* Critical for 24/7 systems

1. Cool OR Offline Backup:

* Backup while database is offline
* Requires application to be shut down during the backup process
* Users can not access the data during the backup
* More reliable but causes downtime

Common Mistakes

1. Only backing up data files (forgetting transaction logs)
2. Storing backups on same server (defeats purpose if server fails)
3. Never testing restores (backups are useless if they don't work)
4. No retention policy (keeping either too many or too few backups)
5. **What s Fault Tolerance?**

A. Fault Tolerance is the prevention of data loss if a component fails

1. **What is disaster recovery?**

A. Disaster recovery is the process of rebuilding an organization’s data after a disaster such as data loss

1. **What is Point In Time Recovery (PITR)?**

A.

* It allows you to restore your database to a specific moment in time
* Requires backing up live database files and archiving the Write Ahead Log (WAL)
* Cannot be used to upgrade postgre SQL DB
* Can only backup and restore a whole cluster
* Can be done while the DB is online (hot backup)

1. **What is WAL Archiving Process?**

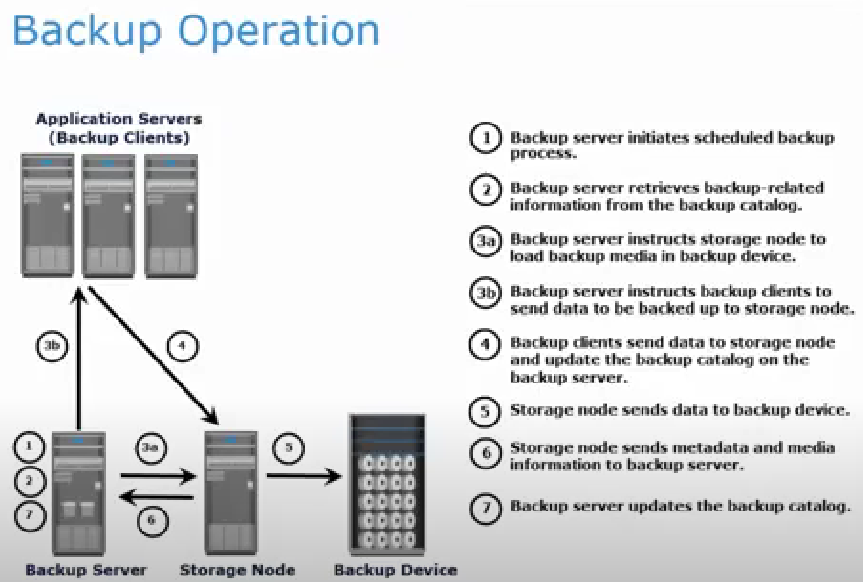
A.

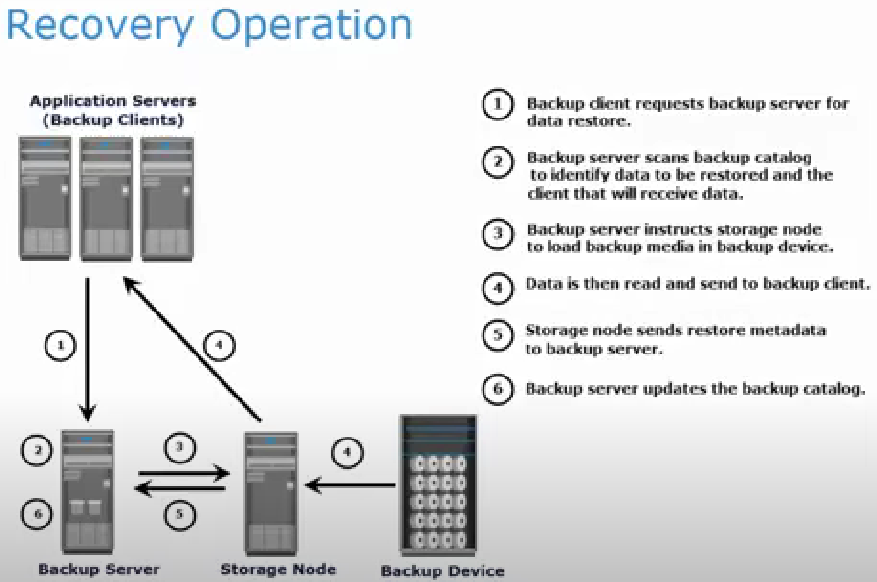
* WAL Stands for Write Ahead Log
* It is a physical backup
* it’s how PostgreSQL logs every change, before writing to the data files.
* WAL archiving is the process of saving (archiving) completed WAL files (segments) to a separate, safe location outside the live database folder.
* Regular backups alone only give you a snapshot. Any changes after the backup will be lost if you restore.
* With WAL archiving, you can recover the database to any moment between your last backup and the current time (or just before a problem happened).
* WAL archiving captures changes to the entire database cluster, not individual databases, or tables.
* It helps in PITR and crash recovery
* Uses built-in DB setting in postgreSQL archive\_command

Why is WAL Backup Needed?

* Point-in-Time Recovery (PITR): Pairing a base physical backup with the sequence of WAL files generated afterward lets you restore the database to any specific point in time (right up to just before a failure or data loss event).
* Crash Recovery: If the database crashes, applying the WAL files after restoring a base backup ensures all transactions are replayed, bringing the system back to a consistent state

Backup Operations





SQL Dump

dump method is to generate a file with SQL commands that, when fed back to the server, will recreate the database in the same state as it was at the time of the dump. PostgreSQL provides the utility program [pg\_dump](https://www.postgresql.org/docs/current/app-pgdump.html) for this purpose. The basic usage of this command is:

Ex: pg\_dump db\_name > dumpfile

* pg\_dump is a regular PostgreSQL client application
* This means that you can perform this backup procedure from any remote host that has access to the database
* it must have read access to all tables that you want to back up, so in order to back up the entire database you almost always have to run it as a database superuser.
* If you do not have sufficient privileges to back up the entire database, you can still back up portions of the database to which you do have access using options such as -n schema or -t table.
* To specify which database server pg\_dump should contact, use the command line options -h host and -p port
* The default host is the local host or whatever your PGHOST environment variable specifies.
* Similarly, the default port is indicated by the PGPORT environment variable or, failing that, by the compiled-in default.
* PostgreSQL client application, pg\_dump will by default connect with the database user name that is equal to the current operating system user name.
* To override this, either specify the -U option or set the environment variable PGUSER.
* An important advantage of pg\_dump over the other backup methods described later is that pg\_dump's output can generally be re-loaded into newer versions of PostgreSQL, whereas file-level backups and continuous archiving are both extremely server-version-specific.
* pg\_dump is also the only method that will work when transferring a database to a different machine architecture, such as going from a 32-bit to a 64-bit server.
* Dumps created by pg\_dump are internally consistent, meaning, the dump represents a snapshot of the database at the time pg\_dump began running. pg\_dump does not block other operations on the database while it is working

Restoring the Dump

Text files created by pg\_dump are intended to be read by the psql program using its default settings. The general command form to restore a text dump is

Ex: psql -X dbname < dumpfile

where dumpfile is the file output by the pg\_dump command

The database dbname will not be created by this command, so you must create it yourself from template0 before executing psql

 To ensure psql runs with its default settings, use the -X option

psql supports options similar to pg\_dump for specifying the database server to connect to and the user name to use.

Non-text file dumps should be restored using the [pg\_restore](https://www.postgresql.org/docs/current/app-pgrestore.html) utility.

Before restoring an SQL dump, all the users who own objects or were granted permissions on objects in the dumped database must already exist. If they do not, the restore will fail to recreate the objects with the original ownership and/or permissions

By default, the psql script will continue to execute after an SQL error is encountered. You might wish to run psql with the ON\_ERROR\_STOP variable set to alter that behavior and have psql exit with an exit status of 3 if an SQL error occurs:

psql -X --set ON\_ERROR\_STOP=on dbname < dumpfile

Either way, you will only have a partially restored database. Alternatively, you can specify that the whole dump should be restored as a single transaction, so the restore is either fully completed or fully rolled back. This mode can be specified by passing the -1 or --single-transaction command-line options to psql

When using this mode, be aware that even a minor error can rollback a restore that has already run for many hours. However, that might still be preferable to manually cleaning up a complex database after a partially restored dump.

The ability of pg\_dump and psql to write to or read from pipes makes it possible to dump a database directly from one server to another, for example:

pg\_dump -h host1 dbname | psql -X -h host2 dbname

======================================================================================================

1. **What is Verbose Output?**

A.

* The -v (or --verbose) flag in pg\_basebackup enables verbose output, which means the tool will show you more detailed information about what it is doing during the backup process.
* Use -v to get more detailed, step-by-step messages about the backup operations performed by pg\_basebackup

1. **What is 2>&1?**

**A.**

* 2>&1 is a shell redirection command
* 2 represents standard error
* 1 represents standard output
* >& is an operator that redirects one output stream to another
* 2>&1 means redirect standard error (stderr) to wherever standard output (stdout) is currently going

**Ex: pg\_basebackup -h localhost -U postgres -D $BACKUP\_PATH -Fp -Xs -v -P > $LOG\_FILE 2>&1**

* + **> $LOG\_FILE** redirects standard output to the log file
  + **2>&1** redirects error to the same place as standard output (the log file)
  + It ensures both normal output and error messages from pg\_basebackup go into the same log file. Without 2>&1, errors would still appear on the console or another place, separate from the logs

1. **What is a Cluster?**

A.

* A cluster in PostgreSQL refers to a collection of one or more databases managed by a single instance of the PostgreSQL database server.
* The term "cluster" in PostgreSQL describes all the databases that share the same physical data directory and are controlled by a single server process.
* Each PostgreSQL cluster contains multiple databases (such as postgres, template1, template0). These databases are logically separated from each other, but they share resources such as user roles and tablespaces managed at the cluster level.
* unlike some other database systems, in PostgreSQL the word "cluster" does not refer to a group of servers by default; it refers to a group of databases on a single server instance.
* In simple way we can say that the number of databases that are part of a sever is called a cluster (databases tab under server tab)

1. **What is pg\_hba.conf file?**

A.

* It is the PostgreSQL "Host-Based Authentication" file.
* It controls which clients (machines, users) are allowed to connect, for what purpose (database access, replication etc.), and with which authentication method
* controlling which users from which client IPs can connect, and how they authenticate.
* To allow replication connections, you must add specific entries for the replication database type.
* When setting up replication (such as master-slave, or streaming replication), the secondary ("slave" or "replica") server must connect to the primary ("master") server in replication mode.
* If the primary does not have a pg\_hba.conf entry allowing this exact connection, the attempt will be rejected with errors like the one you saw.
* A typical pg\_hba.conf line for replication looks like

Syntax:

Host/ hostssl replication <username> client\_ip/subnet auth\_method

Ex:

host replication postgres 49.43.227.147/32 md5

**Host / Host SSL**: applies to TCP/IP connections

**Replication**:  The database field; here, it means replication rather than a database name.

**Username**: The username allowed to connect for replication. PostgreSQL user with replication privilege

**client\_ip**: IP address or subnet of the client machine initiating the connection

**Auth\_method**: authentication method (password, hashed) usually md5 alternatives trust, peer, scram-sha-256, etc.

For testing, trust allows no-password connection but is insecure.

**SSL/Encryption control:**

* Use hostssl if you want the connection to require SSL encryption.
* Use host if either SSL or non-SSL is allowed.

**To set up replication securely and professionally**

1. **Create a replication user**

For security, create a specific user for replication

CREATE ROLE replicator WITH REPLICATION LOGIN ENCRYPTED PASSWORD 'yourpassword';

1. **Edit pg\_hba.conf on the master (primary) server:**

Locate the pg\_hba.conf file in postgreSQL server Data folder

C:\Program Files\PostgreSQL\<version>\data\pg\_hba.conf

Add a line for replication

Only allow your replica(s) to connect, ideally by specific IP

Use hostssl instead of host to enforce SSL encryption if desired

Never use trust in production

1. **Reload or restart postgresql for changes to effect**

**Either reload configuration**: pg\_ctl reload / pg\_ctl reload -D "C:\Program Files\PostgreSQL\<version>\data"

**Or restart the service**: systemctl restart postgresql

1. **Set replication parameters in postgresql.conf**

On your master:

wal\_level = replica

max\_wal\_senders = 5 # Number of replicas that can connect

hot\_standby = on # Needed for standby to accept connections

1. **On the replica, use correct connection parameters:**

When running pg\_basebackup -h primary\_ip -U replicator -D /path/to/data/dir -P --wal-method=fetch ensure password is provided for replication user

**Process:**

1. Locate the pg\_hba.conf file in postgresql server Data folder

C:\Program Files\PostgreSQL\<version>\data\pg\_hba.conf

1. Open the file with text editor
2. Add a line for replication
3. The format should be

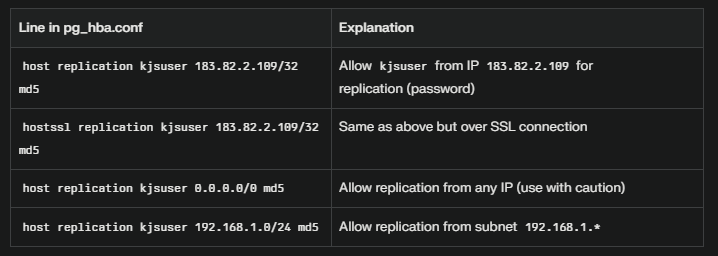
Host/ hostssl replication <username> client\_ip/subnet auth\_method

* host = connection type over TCP/IP
* replication = special database type for replication connections
* [username] = your PostgreSQL user with replication privilege (e.g., kjsuser or postgres)
* [client\_IP\_address/subnet] = IP address or subnet of the client machine initiating the connection (e.g., 183.82.2.109/32 for a single IP)
* [authentication\_method] = usually md5 (password authentication) or scram-sha-256. For testing, trust allows no-password connection but is insecure.

Ex: host replication kjsuser 183.82.2.109/32 md5

1. This allows the user kjsuser connecting from IP 183.82.2.109 to connect for replication role using password authentication.
2. Save changes reload the postgresql configuration

pg\_ctl reload -D "C:\Program Files\PostgreSQL\<version>\data"



1. **What is pgpass.conf file?**

A.

**Process**

1. Open a plain text editor like Notepad.
2. Add one line for each database/server connection with the following format

hostname:port:database:username:password

1. To specify all databases on a server, use \* for the database name:

localhost:5432:\*:postgres:your\_password1

1. Save the file as pgpass.conf (not .pgpass) in the directory:

C:\Users\<YourUserName>\AppData\Roaming\postgresql\pgpass.conf

1. If the folder postgresql does not exist in %APPDATA%, create it manually.
2. Ensure that the pgpass.conf file is readable only by your user account and protected to avoid unauthorized access.
3. **What is pgAgent?**

A.

* pgAgent is an open-source job scheduling agent for PostgreSQL databases.
* It allows you to automate periodic tasks such as database backups, maintenance, sending reports, running scripts, etc.
* You manage pgAgent jobs mainly through the pgAdmin graphical interface, but you can also use SQL for programmatic control.
* Execute several scripts in sequence, either SQL or system batch/shell commands
* Decide what happens on job/step failure—stop, skip, or continue.
* Can run multiple jobs at the same time
* Easy to use via pgAdmin GUI or command line
* Keeps logs of job runs and errors for auditing and troubleshooting
* If the system is sleep or shut down then at that time the schedule will not be done, soon after the system comes alive it executes
* Schedule jobs on a machine/server that remains powered on and always has pgAgent running.

**How It Works:**

* + Runs as a background process (a service/daemon) on your database server or another system.
  + Periodically checks the database for scheduled jobs and executes them automatically.
  + Each job consists of:
    - Steps (each step can be a SQL script or shell/batch script)
    - Schedules (when and how often the job runs)
  + Jobs can be run sequentially (step by step) or in parallel (if scheduled to run at the same time)

1. **What is Replication?**

A.

* Replication is the process of copying data from one database server (called the "primary" or "master") to one or more other servers (called "standbys" or "replicas").

ALTER ROLE KJS WITH REPLICATION;

**Uses:**

* + If the main database crashes, a replica can take over, minimizing downtime and preventing data loss.
  + Replicas can handle complex or read-heavy queries, allowing the primary to focus on updates and writes.
  + Replicated data means you always have a backup copy for emergencies.

**Types of Replications in PostgreSQL:**

1. **Physical (Streaming) Replication:**

* Copies the database cluster by transferring physical changes (file blocks, WAL logs) from primary to replicas in real time.
* Keeps entire cluster in sync. Ideal for fault tolerance and scaling reads.
* Replicas are *read-only* and cannot accept writes.

1. **Logical Replication:**

* Replicates data at the table level using a publish/subscribe model.
* Allows replication of specific tables, supports schema changes, and enables partial data sharing, aggregation, migrations
* Replicas can accept writes; supports transformations and more flexibility

**Replication Modes:**

1. **Synchronous Replication:**

* Primary waits for replica(s) to confirm data receipt before committing. Guarantees zero data loss but may reduce speed.
* Guarantees zero data loss but may reduce speed.

1. **Asynchronous Replication:**

* Primary commits instantly, then transmits changes to replicas.
* Faster, but risks minor data loss during failure (called replication lag).

1. **What is PgBackRest?**

**A.**

* pgBackRest is a robust, open-source backup and restore tool for PostgreSQL. It’s built to handle simple to very large or mission-critical databases, offering:
* Highly reliable and automated **full, incremental, and differential backups**
* Compression, parallelization, encryption, data checks, and automated retention
* Direct backup to local, remote, or cloud storage locations
* Easy recovery and restore, including point-in-time recovery

**Key Concepts in pgBackRest**

* **Repository**: Where your backup files live (can be local disk, network, cloud, or remote server).
* **Stanza**: A named configuration for each PostgreSQL cluster/database you want to back up.
* **Archive**-**push**: Saving WAL (Write-Ahead Log) files to your repo (for point-in-time-recovery).
* **Incremental** **Backup**: Stores only changed files since last full backup.
* **Differential** **Backup**: Stores changes since last full backup (not previous incremental).
* **Remote** **Repository**: A server (Linux/Windows/Cloud) where data is stored directly, without first writing to the DB server’s disk.