01 Backup and Restore Basics

In simple words

- Backup means creating a safe copy of the database so that I can bring it back later.
- Restore means using that copy to rebuild the database when something goes wrong.

Why backup and restore exist

- Databases fail. This is not a question of *if*, only *when*.

- In real systems, things break because of:
  - human mistakes (DELETE or UPDATE gone wrong)
  - disk or server failure
  - VM or cloud instance deletion
  - filesystem corruption

Backup and restore exist to make sure data loss is **recoverable**, not permanent.

What a backup actually captures

- A backup is not just table data.

- Depending on the method, it can include:
  - tables and indexes
  - schema structure
  - ownership and permissions
  - transaction state
  - WAL history (for advanced recovery)

In simple terms, a backup captures a **known safe state** of the database.

Does backup require database downtime?

- No, not always.
- PostgreSQL supports online backups.
- Tools like `pg_dump` take a transaction-consistent snapshot while the database keeps running.
- Users can continue working and data remains consistent inside the backup.

What restore really means

- Restore is not just running one command.

- Restore means:
  - creating an empty or clean database
  - loading backup data into it
  - checking that objects, permissions, and data are correct
  - making sure performance is acceptable

A restore is considered complete only after verification.

Backup without restore is useless

- A backup that has never been restored is not trusted.

- Real-world rule:
  - An untested backup is equal to no backup.

DBAs must always test restores on non-production systems.

Backup and restore as risk management

- DBAs do not take backups because documentation says so.
- They take backups to manage risk.
- Two questions always matter:
  - How much data loss is acceptable? (RPO)
  - How fast must the database come back? (RTO)

Backup strategy is designed around these answers.

Simple example

- Backup taken at 12:00 AM
- Data deleted at 10:00 AM

With only daily backups:

- Maximum data loss = 10 hours

With WAL and point-in-time recovery:

- Data loss = a few seconds

This difference defines DBA decisions.

When I rely on backups

- I rely on backups when:
  - critical data exists
  - human mistakes are possible
  - infrastructure failure is not acceptable

Backups give me confidence, not convenience.

One-line explanation (interview ready)

Backup is the process of saving a safe copy of the database, and restore is the process of rebuilding the database from that copy after failure or data loss.

02 Types of Backups in PostgreSQL: Logical vs Physical

In simple words

- PostgreSQL backups are mainly of two types: logical backups and physical backups.

- Logical backup means rebuilding the database using SQL commands.
- Physical backup means copying the database files exactly as they are on disk.

What is a logical backup?

- A logical backup stores the database as SQL statements such as:
  - **`CREATE TABLE`**
  - **`CREATE INDEX`**
  - **`INSERT INTO`**
  - **`ALTER`** and **`GRANT`** commands
- When this backup is restored, PostgreSQL executes these SQL commands again to recreate the database.

- Tools commonly used:
  - **`pg_dump`**
  - **`pg_dumpall`**

How logical backup works internally

- The database remains online
- PostgreSQL takes a transaction-consistent snapshot
- Objects and data are read logically
- SQL statements are written to a dump file

Users can keep working during the backup without data corruption.

When I use logical backups

I use logical backups when: * migrating databases between servers * upgrading PostgreSQL versions * restoring only specific tables or schemas * moving data across different platforms

Logical backups are flexible and portable.

What is a physical backup?

- A physical backup copies PostgreSQL's actual data files from disk.

- This includes everything inside the data directory:
  - table files
  - index files
  - system catalogs
  - WAL files
  - control and metadata files
- Physical backups recreate the database exactly as it was.

How physical backup works internally

- Physical backups can be taken in two ways:

  - Offline method:
    - PostgreSQL is stopped
    - entire data directory is copied
    - consistency is guaranteed

- Online method:
  - PostgreSQL keeps running
  - tools like pg_basebackup are used
  - WAL files ensure consistency

No SQL is involved. Files are copied byte-by-byte.

Logical vs physical backup (core difference)

- Logical backup:
  - rebuilds database using SQL
  - slower restore
  - supports partial restore
  - works across PostgreSQL versions

- Physical backup:
  - clones the database files
  - very fast restore
  - no partial restore
  - must match PostgreSQL version and architecture

Restore behavior difference

- Logical restore:
  - executes SQL one statement at a time
  - rebuilds indexes
  - requires ANALYZE after restore

- Physical restore:
  - places files back on disk
  - PostgreSQL starts and replays WAL
  - database becomes usable quickly

Partial restore capability

- Logical backups allow:
  - restoring a single table
  - restoring a schema only
- Physical backups do not support partial restore.
- The entire cluster must be restored.

Downtime considerations

- Logical backup:
  - backup runs online
  - restore takes longer

- Physical backup:
  - offline backup requires downtime
  - online base backup avoids downtime
  - restore downtime is minimal

PostgreSQL version compatibility

- Logical backup:
  - can restore across PostgreSQL versions
  - safe for upgrades

- Physical backup:
  - must use the same PostgreSQL version
  - same architecture and layout expected

How a DBA chooses between them

- I choose logical backups when flexibility matters.
- I choose physical backups when recovery speed matters.

In real production systems, both are used together.

Final mental model

- Logical backup = rebuild
- Physical backup = clone
- Logical = portable and flexible
- Physical = fast and exact

One-line explanation (interview ready)

Logical backups store SQL instructions to recreate a database, while physical backups copy the actual database files for faster full recovery.

03 Transaction Consistency and Snapshots in PostgreSQL Backups

In simple words

- Transaction consistency means the backup represents a database state that could actually exist at one moment in time.
- Even if users are actively inserting, updating, or deleting data, the backup must not be broken or half-written.
- PostgreSQL guarantees this using **snapshots and WAL**.

Why transaction consistency matters

- A database is always changing.

- At any second:
  - some transactions are committed
  - some are running
  - some are rolled back
- If a backup captures a mix of these states, restore becomes useless.

- Transaction consistency ensures:
  - no partial transactions
  - no corrupted logic
  - restored database behaves like a real past moment

What a snapshot actually is

- A snapshot is PostgreSQL's internal view of the database at a specific instant.

- It answers three questions:
  - Which transactions are committed?
  - Which transactions are still running?
  - Which transactions must be ignored?

Once a snapshot is taken, PostgreSQL reads data **as if time is frozen** at that point.

How pg_dump uses snapshots

- When **pg_dump** starts:
  - PostgreSQL creates a transaction snapshot
  - **pg_dump** reads all tables using that snapshot
- Even if new data is inserted after the dump starts, **pg_dump** does not see it.

- This guarantees that:
  - all tables match the same point in time
  - foreign keys and references remain valid

Why users can keep working during backup

- **pg_dump** does not block reads or writes.

- It only:
  - reads committed data visible to its snapshot
  - ignores later changes
- That is why backups can run on live production systems without downtime.

Snapshot consistency across multiple tables

- Snapshots ensure that relationships stay intact.
- Example:
  - order exists
  - customer exists
  - both appear consistently in the backup
- Without snapshots, one table might reflect new data while another does not.

Physical backups and consistency

- Physical backups copy actual data files.

Consistency depends on how** the backup is taken:**

- Offline physical backup
  - PostgreSQL is stopped
  - no data changes
  - consistency is guaranteed

- Online physical backup
  - PostgreSQL is running
  - WAL records all changes
  - WAL replay fixes partial file copies

Why WAL is critical for consistency

- WAL (Write-Ahead Log) records every change before it reaches data files.

- During restore:
  - PostgreSQL replays WAL
  - incomplete pages are fixed
  - database reaches a consistent state
- Without WAL, online physical backups cannot be recovered.

Snapshots vs filesystem snapshots

- PostgreSQL snapshot:
  - logical view of transactions
  - used by pg_dump

- Filesystem snapshot:
  - OS or storage-level freeze
  - used for physical backups
- Both freeze time, but at different layers.

Common mistake to avoid

- Copying data files while PostgreSQL is running **without WAL or snapshot support** leads to:
  - broken backups
  - failed restores
  - silent corruption
- Never mix partial file copies with live databases.

When I rely on snapshot-based consistency

- I rely on snapshots when:
  - taking logical backups
  - running backups on live production
  - ensuring zero data corruption
- Snapshots are what make online backups safe.

Final mental model

- Snapshot = freeze database view
- WAL = fix incomplete writes
- Logical backup = snapshot-based
- Physical online backup = WAL-based

One-line explanation (interview ready)

Transaction consistency ensures a backup reflects a real, complete database state at a single point in time, achieved using snapshots and WAL.

4 Roles, Permissions, and Backup Requirements in PostgreSQL

In simple words

- Backups do not fail because of tools.
- They fail because of **permissions**.

- PostgreSQL backup tools only read what the connected role is allowed to see. If the role lacks access to even one required object, the backup can fail or become incomplete.

Why roles matter in backup and restore

- PostgreSQL is strict about access control.
- Every database object has an owner and permissions.

- When a backup tool connects:
  - it acts like a normal client
  - it obeys all role and permission rules

There is no special "backup mode" that bypasses security.

Common PostgreSQL backup tools and permissions

- `pg_dump` → backs up **one database**
- `pg_dumpall` → backs up **entire cluster metadata + all databases**
- `pg_basebackup` → takes **physical backups** of the cluster

Each tool requires different permission levels.

Role requirements for logical backups (`pg_dump`)

- To successfully dump a database:
  - the role must be able to connect to the database
  - the role must have read access to **all schemas, tables, views, and sequences** being dumped

If the role lacks access to even one table, `pg_dump` stops with an error.

Practical rule

- In production, `pg_dump` is usually run as:
  - database owner, or
  - a role with full read access, or
  - a superuser

Why `pg_dump` fails even though the database exists

- Common reasons:
  - role cannot access one schema
  - role cannot read a table or view
  - role cannot access a function

- `pg_dump` does not silently skip objects.
- If it cannot read something, it fails.

Role requirements for `pg_dumpall`

- `pg_dumpall` captures:
  - roles
  - role memberships
  - tablespaces
  - all databases
- This requires:
  - superuser privileges

A non-superuser cannot dump global objects.

Role requirements for physical backups

- Physical backups access raw database files.

- Tools like `pg_basebackup` require:
  - superuser, or
  - replication role with backup privilege (newer PostgreSQL versions)

Without sufficient privileges, the server refuses access.

Restore-side role requirements

- Restore often fails because roles do not exist on the target server.

- Logical backups may contain:
  - object ownership
  - GRANT statements
- If referenced roles are missing:
  - restore completes with errors
  - ownership and permissions break

Best practice:

Always restore roles **before** restoring databases.

Restoring as a different role

- Sometimes, original roles are not needed on the target system.

- Options:
  - use `--no-owner` during restore
  - use `--no-privileges` to skip GRANT statements

This avoids permission conflicts in test or staging systems.

Backups do not include everything

- Logical backups:
  - do not include PostgreSQL users unless pg_dumpall is used
  - do not include server configuration files

- Physical backups:
  - include cluster files
  - but may not include external scripts or OS-level configs

DBAs must know what is and is not covered.

Backup security is critical

- Backup files contain:
  - full data
  - sensitive information
  - passwords (indirectly)
- Best practices:
  - restrict file permissions
  - encrypt backups when possible
  - limit who can run backup tools

A leaked backup is a full data breach.

When I plan backup roles

- I ensure that:
  - backup roles have minimum required permissions
  - backups run non-interactively
  - role permissions are documented

Permissions should enable backups, not weaken security.

Final mental model

- Backup tools obey role permissions
- Superuser makes backups easy but risky
- Missing roles break restores
- Security applies even during backup

One-line explanation (interview ready)

PostgreSQL backup tools work under role permissions, so proper access rights are required to back up and restore database objects safely.