03 Handling Large Databases

01 Large Database Backup Problems and Strategy (PostgreSQL)

In simple words

- Large databases change the whole game. Techniques that work fine on small databases start failing badly once data grows into hundreds of GBs or TBs. At that scale, backups are no longer just about running commands — they are about proper planning, understanding limits, and making smart trade-offs.

Why large databases are hard to back up

- **As database size increases:**
  - backup time increases linearly
  - restore time increases even more
  - I/O pressure affects live queries
  - disk space requirements explode
- **At large scale, the question is not:**
  - "Can I take a backup?"
- **It becomes:**
  - "Can I restore it fast enough when things break?"

The biggest mistake with large databases

- Using small-database thinking.
- **Common wrong assumptions:**
  - daily full logical dump is fine
  - restore time will be acceptable
  - disk space will somehow work
- These assumptions fail badly at scale.

Logical backups vs large databases

- **Logical backups on large databases:**
  - take many hours
  - create massive dump files
  - rebuild indexes during restore
  - cause long downtime
- Logical backups do work, but they are rarely the **primary recovery method**.
- **They are better suited for:**
  - migrations
  - partial restores
  - audits

Physical backups are mandatory at scale

- For large databases, physical backups become essential.
- **Why:**
  - file-level copy is much faster
  - restore does not rebuild indexes
  - WAL replay is faster than SQL replay
- **Large systems depend on:**
  - base backups
  - WAL archiving
  - point-in-time recovery

Backup window reality

- Every system has a backup window.
- **At scale:**
  - backups compete with production traffic
  - I/O saturation slows users
  - long backups increase risk
- A strategy must fit inside an acceptable time window.

Restore time is more important than backup time

- DBAs often optimize backup time.
- Senior DBAs optimize **restore time**.
- **Key question:**
  - "If the database dies at 2 AM, how fast can I bring it back?"
- **This decides:**
  - backup type
  - frequency
  - storage choice

Incremental thinking for large databases

- Large systems avoid full backups too frequently.
- **Typical strategy:**
  - occasional full base backup
  - continuous WAL archiving
  - incremental or differential layers
- **This reduces:**
  - backup time
  - storage pressure

I/O and performance impact

- Backups are I/O heavy.
- **Poor strategy causes:**
  - slow queries
  - replication lag
  - timeout errors
- **At scale, backup I/O must be:**
  - throttled
  - scheduled carefully
  - monitored

Storage planning matters

- **Large backups need:**
  - high throughput storage
  - fast restore access
  - off-host replication
- Backup stored on slow disks equals slow recovery.

Testing strategy at scale

- **Testing restore on large databases:**
  - takes time
  - needs separate infrastructure
  - cannot be skipped
- Even partial restore tests are valuable.
- Untested large backups are dangerous.

Real DBA strategy mindset

- **A good large-DB backup strategy balances:**
  - backup frequency
  - restore speed
  - storage cost
  - operational complexity
- There is no single perfect solution.

Final mental model

- Small DB thinking fails at scale
- Physical backups dominate
- Restore speed decides strategy
- Planning matters more than tools

One-line explanation

Large PostgreSQL databases require backup strategies focused on restore speed, I/O impact, and storage planning rather than simple full logical dumps.

02 Compression and Splitting Techniques for PostgreSQL Backups

In simple words

Compression and splitting exist to solve two real problems:

- backups are too big
- backups are hard to move and restore

At scale, storing and handling backups becomes as important as creating them.

Why compression is needed

- **Large databases produce:**
  - huge dump files
  - high storage usage
  - slow transfers
- **Compression reduces:**
  - disk space usage
  - network transfer time

But it always trades disk savings for CPU usage.

Compression with logical backups

External compression using gzip

```
pg_dump mydb | gzip > mydb.sql.gz
```

What happens:

- **pg_dump** outputs SQL
- **gzip** compresses the stream
- a smaller file is written

This is simple and widely used.

Compression with custom format

```
pg_dump -Fc mydb > mydb.dump
```

Custom format:

- uses internal compression
- avoids external gzip
- restores faster than plain SQL

For most production systems, this is the preferred option.

CPU impact of compression

Compression is CPU-heavy.

- **High compression levels:**
  - reduce file size
  - slow down backup
- **Low compression levels:**
  - faster backups
  - larger files

DBAs must balance CPU availability and backup windows.

Splitting large backups (file management)

Very large backup files are:

- difficult to move
- harder to store
- risky to handle

Splitting breaks a large backup into manageable chunks.

Splitting plain SQL dumps

```
pg_dump mydb | split -b 5G - mydb_part_
```

This creates multiple 5GB files.

During restore:

```
cat mydb_part_* | psql -d target_db
```

Splitting compressed dumps

```
pg_dump mydb | gzip | split -b 2G - mydb.gz_
```

Restore:

```
cat mydb.gz_* | gunzip | psql -d target_db
```

Splitting helps with storage and transfer limits.

Directory format avoids splitting issues

Using directory format:

```
pg_dump -Fd mydb -f mydb_dir
```

Advantages:

- files are already separated
- supports parallel restore
- easier to manage large datasets

This is the cleanest solution for very large databases.

Network transfer considerations

Compressed backups:

- reduce bandwidth usage
- increase CPU load

Uncompressed backups:

- faster CPU usage
- slower transfers

Network speed matters more than compression level.

Common mistakes

- compressing on already CPU-saturated servers
- using maximum compression blindly
- splitting without restore testing

Compression must be tested, not assumed.

DBA best practices

- prefer custom or directory formats
- compress when network or disk is limited
- avoid heavy compression during peak hours
- test restore using split files

Final mental model

- Compression saves space, costs CPU
- Splitting improves manageability
- Directory format scales best
- Restore testing validates everything

One-line explanation

Compression and splitting help manage large PostgreSQL backups by reducing storage size and improving transfer reliability, at the cost of additional CPU usage.

03 Parallel Backup and Restore in PostgreSQL

In simple words

- Parallel backup and restore means doing the work using multiple workers at the same time instead of one. It splits the load across CPU cores, which can drastically reduce total backup or restore time. When planned properly, it can save hours, but if used blindly, it can overload the system and hurt production performance.

Why parallelism exists

Single-threaded backups become too slow as databases grow.

Parallelism exists to:

- speed up restore time
- use modern multi-core CPUs
- reduce outage windows

But it increases CPU, I/O, and lock pressure.

Important rule (must remember)

> **`pg_dump` does NOT support parallel backup.**

Only **`pg_restore`** supports parallelism.

This surprises many DBAs.

Parallel restore using `pg_restore`

Parallel restore works with:

- custom format (`-Fc`)
- directory format (`-Fd`)

Example:

```
pg_restore -j 4 -d target_db backup.dump
```

Here:

- `-j 4` means 4 parallel workers

Each worker restores different objects.

Why directory format is best for parallel restore

Directory format stores objects as separate files.

This allows:

- maximum parallelism
- minimal contention
- fastest restore

For very large databases, directory format + parallel restore is the best combo.

What actually runs in parallel

Parallel restore can process:

- table data
- indexes
- constraints

Some objects are still restored serially:

- roles
- extensions
- dependencies

So parallelism is helpful, but not unlimited.

Choosing the right number of jobs

More jobs ≠ always faster.

Guidelines:

- start with CPU cores / 2
- monitor disk I/O
- avoid saturating production disks

Typical values: 4 to 8 jobs.

Parallel restore on production vs test

On test or DR servers:

- aggressive parallelism is fine

On production systems:

- restore `I/O` can starve other workloads
- parallel restore must be planned

Restore speed must not crash the system.

When parallel restore helps the most

Parallel restore is most effective when:

- database has many large tables
- many indexes exist
- directory format is used

It helps less with:

- small databases
- few tables

Common mistakes

- using `-j` too high
- ignoring disk limits
- running parallel restore on live production

These cause timeouts and system slowdown.

Testing parallel restore

Always test:

- different `-j` values
- restore duration
- system load

One-size-fits-all does not exist.

Final mental model

- Parallelism saves time
- `pg_restore` enables it
- directory format scales best
- limits must be respected

One-line explanation (interview ready)

Parallel restore in PostgreSQL uses multiple workers via `pg_restore` to speed up logical restores, mainly when using custom or directory formats.

04 Performance Impact During Backup in PostgreSQL

In simple words

Backups are not free.

Every backup consumes resources:

- disk I/O
- CPU
- memory

If not planned properly, backups slow down live users and can even cause outages.

Why backups affect performance

During a backup, PostgreSQL:

- reads large amounts of data
- scans tables sequentially
- competes with normal queries for I/O

On busy systems, this competition becomes visible to users.

Disk I/O is the biggest bottleneck

Most backup pain comes from disk I/O.

Symptoms of I/O saturation:

- queries become slow
- replication lag increases
- checkpoints take longer
- timeouts appear

CPU is rarely the first problem; disks usually are.

Impact of logical backups (`pg_dump`)

`pg_dump`:

- performs sequential scans
- generates continuous read load
- can evict useful pages from cache

On large databases, `pg_dump` can degrade query performance if run during peak hours.

Impact of physical backups

Physical backups:

- read raw data files
- create sustained I/O pressure
- may impact WAL write speed

Online physical backups depend heavily on storage speed.

Effect on replication

During backups:

- primary disk I/O increases
- WAL generation may increase
- replicas can lag

Replication lag during backup is a common production issue.

Backup timing strategy

Senior DBAs schedule backups:

- during low traffic windows
- when batch jobs are minimal
- outside peak business hours

Timing matters more than backup speed.

Throttling backup impact

Ways to reduce impact:

- limit parallel restore jobs
- reduce compression level
- use directory format wisely
- avoid running multiple backups at once

Gentle backups are better than fast ones that break production.

Using replicas for backups

A common strategy:

- run backups on a standby server
- offload I/O from primary

This reduces impact on live users.

But replication lag must be monitored.

Monitoring during backups

While backups run, I monitor:

- disk I/O metrics
- query latency
- replication lag
- PostgreSQL logs

Backups without monitoring are risky.

Common DBA mistakes

- running backups during peak hours
- ignoring I/O limits
- using maximum compression blindly
- backing up primary when replicas exist

Most incidents are avoidable.

Final mental model

- Backups consume resources
- I/O is the main enemy
- Timing reduces risk
- Monitoring keeps backups safe

One-line explanation

PostgreSQL backups impact performance mainly through disk I/O, so timing, throttling, and offloading backups are critical in production systems.

05 Disk Space Planning and Monitoring for PostgreSQL Backups

In simple words

- Most backup failures happen because the disk runs out of space. It's usually not PostgreSQL's fault and not a tool problem either. Proper disk space planning is what truly separates a safe, reliable backup strategy from a risky one.

Why disk space is critical for backups

Backups need space at multiple places:

- source server (read + temporary usage)
- backup destination
- WAL directory (especially during physical backups)

If **any one** of these fills up, backups and sometimes the database itself can crash.

The silent killer: WAL growth

During backups, especially physical backups:

- WAL generation increases
- WAL retention increases
- archive queue grows

If WAL directory fills up:

- PostgreSQL can stop
- write operations fail

This is a very real production outage scenario.

Planning disk space for logical backups

For logical backups, I plan:

- database size × compression ratio
- temporary space during dump
- restore-time space (often more than backup)

Important rule:

> Restore usually needs **more space** than backup.

Planning disk space for physical backups

Physical backups require space for:

- full base backup
- archived WAL files
- multiple backup generations

Physical backup storage grows continuously. Retention policy is mandatory.

Retention policies (must have)

Retention policy defines:

- how many backups to keep
- how long to keep WAL files

Without retention:

- disk fills silently
- oldest backups are never removed

Automation is critical here.

Monitoring disk usage

I always monitor:

- data directory usage
- WAL directory growth
- backup destination usage
- archive backlog

Disk monitoring should alert **before** space runs out.

Common disk-related backup failures

- backup stops due to disk full
- WAL archiving pauses
- restore fails mid-way
- primary database becomes read-only

Disk issues often appear suddenly but grow slowly.

Best practices for disk safety

- keep backups off the database server
- separate WAL and data disks
- monitor free space trends
- test retention cleanup

Disk space is cheap. Outages are not.

Real DBA mindset

Senior DBAs don't ask:

> "How big is the backup today?"

They ask:

> "How fast is disk filling every day?"

Trend matters more than snapshot.

Final mental model

- Backups multiply disk usage
- WAL growth is dangerous
- Retention prevents disasters
- Monitoring saves jobs

One-line explanation

Proper disk space planning and monitoring are essential for PostgreSQL backups to prevent WAL buildup, backup failures, and database outages.