

A PLAYBOOK

OCTOBER 2026

THE DATABASE ENGINEER'S

PRODUCTION PLAYBOOK

Best practices for reliability, performance and safety

10	20	35
Disciplines	Concepts	Questions

Mouad Krikbou
Systems Architect & Engineer

Engineer by trade, artist by habit.

INTRODUCTION

DISCIPLINE
OVER TECHNIQUE

A dependable database is not the most sophisticated one. It is the one run with discipline, stripped of unnecessary risk, keeping the qualities the system genuinely needs.

I reduce database engineering to what I reduce architecture to: a handful of principles that, respected, make a system dependable. Everything else is technique. The techniques matter, but they change. The principles do not.

Have you ever run a schema change on a busy table and nobody noticed? Then someone hid the difficulty well. If instead you are waiting on a lock, or asking why a migration took three hours, the opposite happened.

The guide has two parts. **Part I** is ten operational disciplines: what to do, each with a story and the question I ask in a review. **Part II** is twenty concepts: what you are actually touching when you do it.

THREE IDEAS IT RESTS ON



Every change is a deployment

With a window, a rollback, a validation and someone on call.



Old and new, together

The schema serves both applications for as long as the rollout takes.



Evidence, not hope

A restore, a failover, a rollback: tested, or it is a hypothesis.

10
Disciplines
The operational part, each ending with a question.

20
Concepts
What lies underneath, to master and to revisit.

35
Questions
A checklist for before a deploy or a migration.

CONTENTS

WHAT'S INSIDE

01 One guide, many engines 04

02 Mindset and operating principles 06

03 Zero-downtime migrations 07

04 Locks, timeouts and long transactions 09

05 Backups, restores and recovery 10

06 Monitoring and observability 11

07 Performance and capacity 12

08 Security and least privilege 13

09 High availability and failover 14

10 Data quality and migration safety 16

11 Operational hygiene 17

12 Runbooks and change management 18

13 How the principles work together 19

14 Twenty concepts to master and revisit 21

15 The pre-deploy checklist 25

16 References 27

HOW TO READ IT

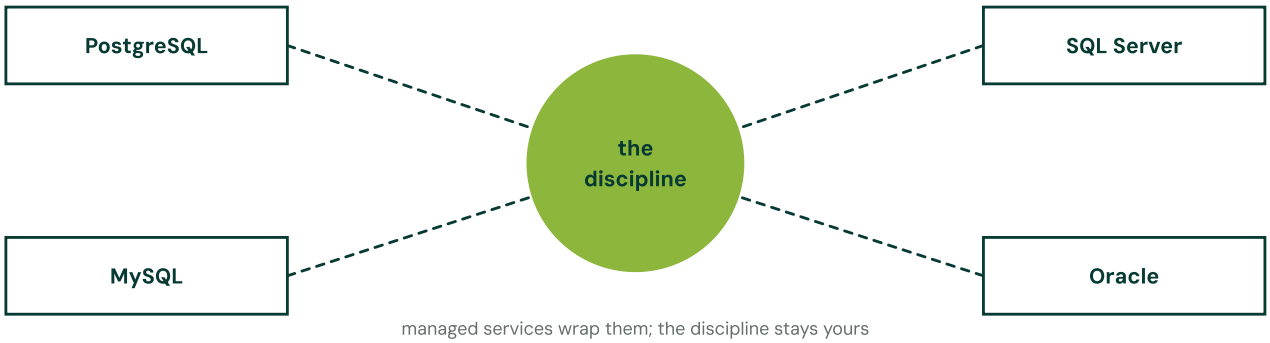
Part I in order, the first time; after that, the chapter you need before a change. Part II when a problem in Part I needs explaining. The italic questions are the ones I ask in a review: ask them of your own system.

CHAPTER 01

ONE GUIDE, MANY ENGINES

The examples are PostgreSQL-shaped; the discipline is not.

PostgreSQL makes the examples concrete because its mechanisms are open and well documented. But this is about running a relational database in production, and that work is much the same on MySQL, SQL Server, Oracle, or a managed service that wraps one.



managed services wrap them; the discipline stays yours

Area	What transfers: the principle	What changes: the mechanism
Locking	A schema change must never queue behind a long read	Lock modes, queues, escalation
Replication	A choice between data loss and latency	Topology and tooling
Backups	Physical and logical; restores tested	Restore and recovery commands
Maintenance	Dead space accumulates and must be reclaimed	Vacuum, purge, ghost records
Planner	The plan is the truth; statistics feed it	Plan output, hints, statistics
Isolation	The level is a contract, matched to the business	Same names, different anomalies

On PostgreSQL

Read it as written; the mechanisms match your daily work.

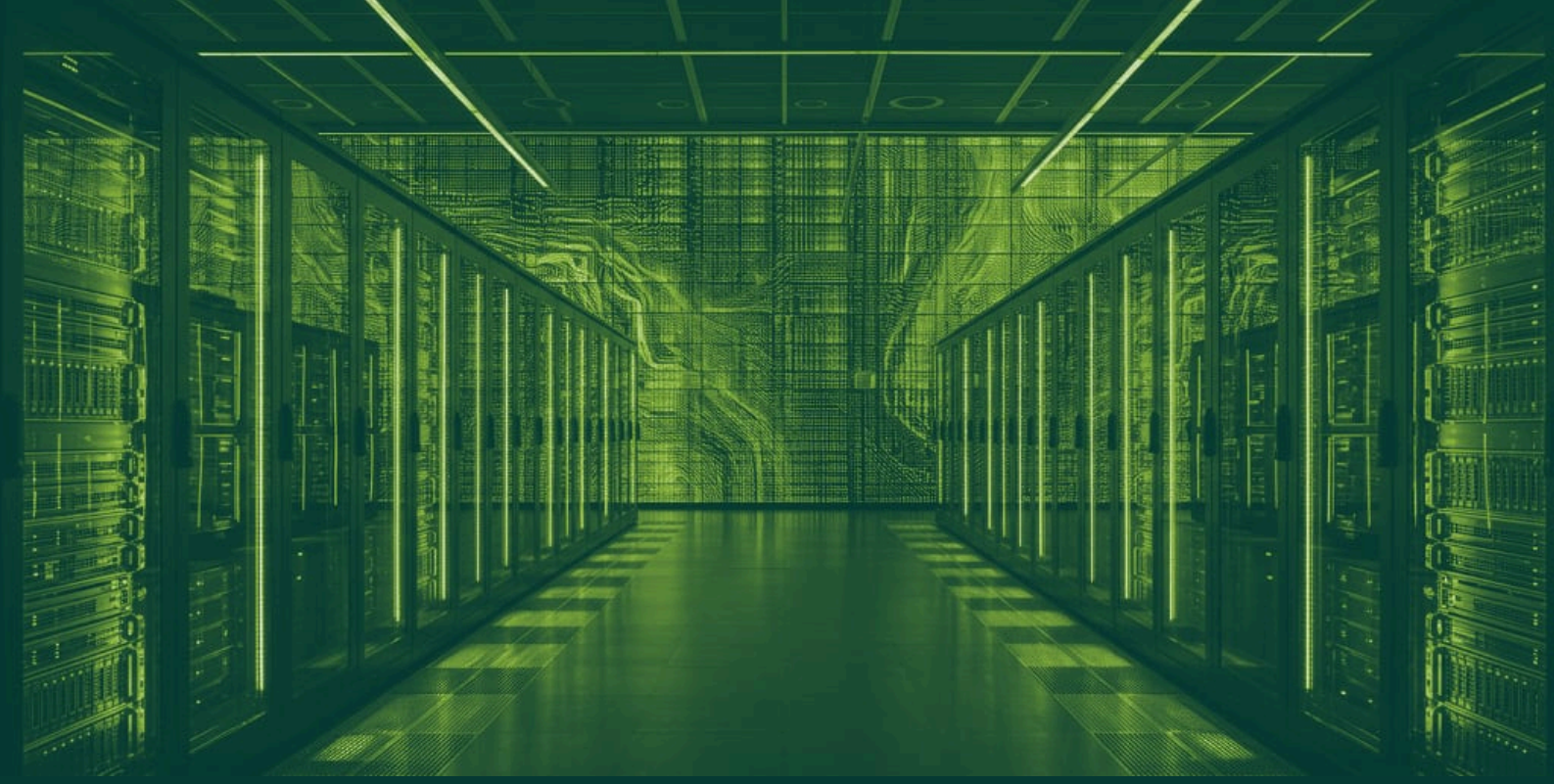
On another engine

Read Part I as a list of disciplines and translate each mechanism. Part II is a map of concepts, not commands.

On a managed service

Add one question to every chapter: which part does the provider own, and which part is still mine?

> *Which parts of this guide does my engine change, and which does it leave exactly as written?*



01

PART ONE

OPERATIONS

TEN DISCIPLINES

What to do before, during and after a change, so a risky operation becomes a routine one.

- Mindset and operating principles
- Zero-downtime migrations
- Locks, timeouts and long transactions
- Backups, restores and recovery
- Data quality and migration safety
- Monitoring and observability
- Performance and capacity
- Security and least privilege
- High availability and failover
- Hygiene, runbooks and change

CHAPTER 02

MINDSET
AND OPERATING PRINCIPLES

Most outages come from breaking changes and untested assumptions, not from hardware.

Before choosing a migration strategy, a backup tool or a failover design, understand what failure looks like. A disciplined approach turns risky operations into repeatable, low-stress procedures.

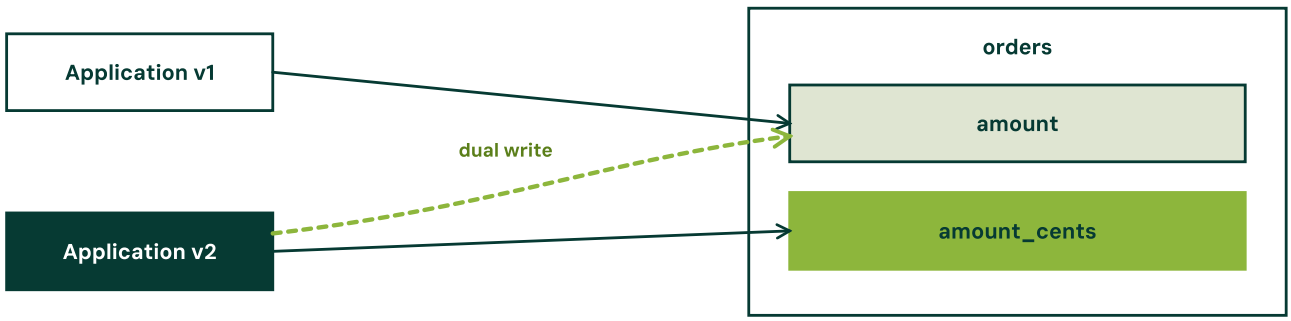
2.1 Treat every change as a deployment

A schema change is not a quick fix: it has a window, a rollback path, a validation plan and someone to tell. A column added on a quiet afternoon took under a second on staging and eleven minutes on forty million production rows, while checkout failed. The next one used expand and contract, with a short lock timeout, and nobody noticed it.

> *If this change goes wrong at 2 a.m., who is on call, and what do they do?*

2.2 Design for compatibility

The application and the database evolve independently: the schema serves the old and the new application at the same time, for as long as the rollout takes. This is the most important shift in the guide; every other chapter depends on it. To rename `amount` to `amount_cents`, a payments team added the new column, wrote both for two weeks, moved reads gradually, and dropped the old one a month later. No user saw an error.



both work, for as long as the rollout takes

> *Can the old application and the new one run against this schema at the same time?*

2.3 Assume things will fail

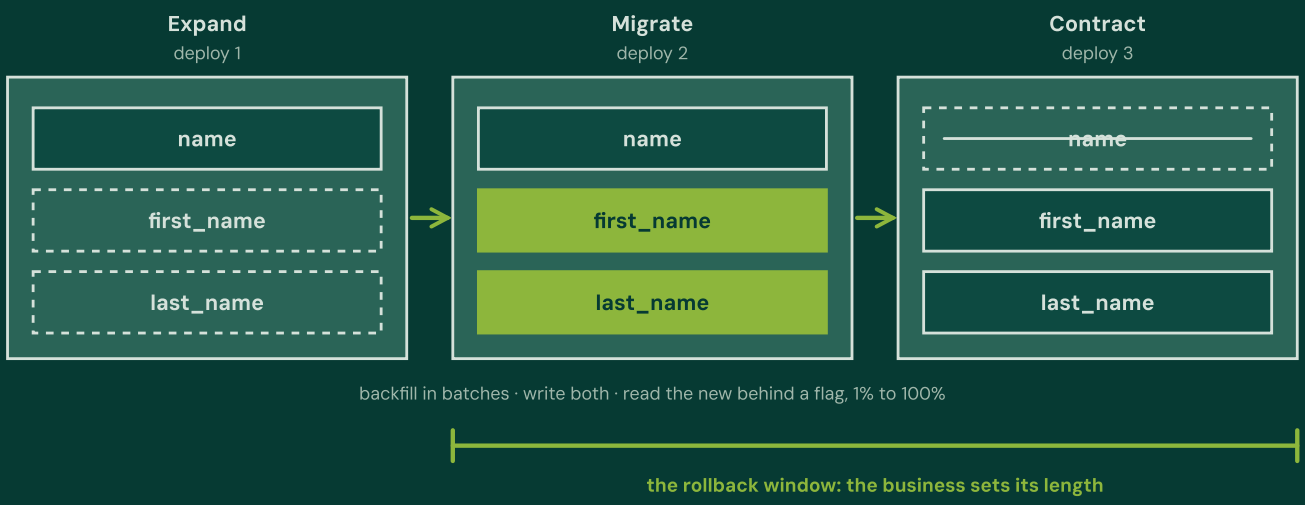
The migration will be interrupted, the backfill will run slow, the primary will fail mid-deploy. I watched an unbatched backfill run six hours and hit the write-ahead log limit; it rolled back and started over. In batches of ten thousand with a pause between, it finished in ninety minutes and never held a long transaction.

> *What is the rollback plan, and when was it last tested?*

CHAPTER 03

ZERO-DOWNTIME MIGRATIONS

Change the schema without stopping the application: three phases, three deployments, and the discipline is in never skipping one.



E Expand

Add the new columns or tables beside the old, nullable, unconstrained. Drop nothing. Set a short lock timeout first, so a blocked change fails fast instead of freezing the table.

M Migrate

Backfill in small batches. Ship code that writes both, reads the old, then reads the new behind a flag: one percent, ten, all, over a week.

C Contract

When nothing reads the old, drop it in its own deployment, with its own review. The smallest change, and the only one you cannot undo the same day.

I have run this pattern on real systems, and what mattered most both times was keeping the three phases in three deployments. The temptation to combine them is strong, and it is exactly the temptation that causes outages.

> *Which phase are we in, and is it really finished?*

3.2 Safe DDL practices

Some changes are cheap, some expensive, some lock the table. Keep the expensive ones from blocking the cheap ones:

- Add constraints as not valid, then validate separately, under a weaker lock.
- Build large indexes concurrently, so writes carry on.
- Set lock and statement timeouts in every migration script.
- Batch and throttle backfills, so production traffic is never starved.

Requiring an email on every user, the team added the constraint as not valid, then validated it: still a full scan, but with a lock the application could live with.

> *What does this operation lock, and for how long?*

3.3 The rollback window

Every migration has a window in which the old system must stay recoverable, and the business sets it, not the engineer's confidence. Inside it the old schema stays readable, the old data intact, the rollback tested.

I saw a migration declared a success and the old columns dropped the same day. Two days later a rounding rule turned out to have been lost in the backfill, and with no rollback path left, the team spent a week rebuilding the old data from backups and logs.

> *How long does the business need the old system recoverable, and is that respected?*

A migration is finished when the old system can be let go, *not when the new one starts.*

< 1 s
On staging
A few thousand rows: the change looked free.

11 min
On production
Forty million rows, rewritten under a lock.

2 wk
Dual writes
Both columns, while every instance caught up.

0
Errors seen
The same change, done in three deploys.

The figures are from the stories in chapters 2 and 3.

A schema change…	Usually costs	The safer way
Add a nullable column	**Metadata only**	As it is
Build an index	Blocks writes while it builds	Build it concurrently
Add a constraint	Scans the table under a strong lock	Not valid first, validate later
Change a column's type	Often rewrites the whole table	A new column, expand and contract
Rename or drop a column	Breaks every instance still using it	The contract phase, last

What a change costs differs by engine and version: check yours before relying on a row of this table.

CHAPTER 04

LOCKS, TIMEOUTS
AND LONG TRANSACTIONS

Locks protect the database from inconsistency, and they turn a small problem into a large one. One long transaction can freeze a table.

Without a lock timeout: everything queues behind the change



With a 2 s lock timeout: the change gives up, the table keeps moving



the change failed after 2 s and was retried 15 s later, when the read had finished

4.1 Timeouts on everything

Lock and statement timeouts on every migration, statement timeouts on every query, idle-in-transaction timeouts on every connection. A report that grew from two seconds to minutes held its locks until the nightly batch queued behind it; with thirty seconds it would have failed loudly and been fixed.

> *How long may this wait, and then what?*

4.2 Fail fast, retry later

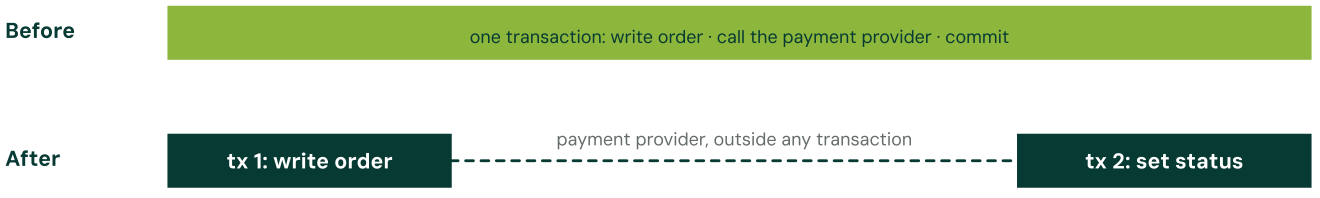
A blocked query holds its own locks, and the queue behind it grows until the table is frozen. A failed query releases everything. With a two-second lock timeout, a migration behind an analytics read failed at once and succeeded on retry fifteen seconds later. The failure was the feature.

> *Should this wait, or give up?*

4.3 Keep transactions short

Wait for nothing inside a transaction that you could wait for outside it. A checkout that called the payment provider inside its transaction held order and inventory locks for as long as the provider was slow, and every other checkout queued.

> *Is anything in here waiting on something outside the database?*



the same work; locks held for a fraction of the time

CHAPTER 05

BACKUPS
RESTORES AND RECOVERY

Backups are not enough. Recovery must be tested.

5.1 Physical and logical, both

Physical backups restore the whole state fast; logical backups restore a part, portably. When a careless delete removed one tenant's rows, the physical backup could only roll every tenant back; the daily logical one restored just that tenant.

Recovering from…	Physical	Logical
The disaster: the whole database	**Fast, complete**	Slow
The mistake: one tenant, one table	Rolls back everyone	**Just what was lost**

The disaster is rare. The mistake is common.

5.2 Test restores, end to end

An unrestorable backup is false confidence. My restore test came up clean, connected, and then failed on a table newer than the procedure, whose indexes it never rebuilt. Found in a test, not an incident.

> *When was the last restore, and did it follow the procedure end to end?*

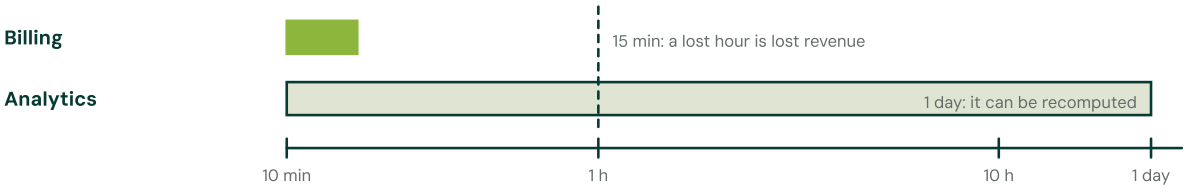
5.3 Verify that it boots

Some teams restore the backup into an isolated machine on a schedule, boot it, and capture a screenshot of what comes up. One such check caught three weeks of silently truncated backups that every job had reported as a success.

> *Would the restored system actually come up?*

5.4 Recovery objectives, per service

How much data may be lost (RPO) and how long recovery may take (RTO) are business decisions. One hourly policy for every service was too loose for billing and needlessly dear for analytics; split, both were served, and the total cost went down.



the one hourly policy: too loose for one, too dear for the other

CHAPTER 06

MONITORING AND OBSERVABILITY

You cannot fix what you cannot see. Good signals shorten recovery; missing ones turn a small problem into a long incident.

6.1 The minimum set of signals

Up or down | **Replication lag** | Query latency | Error rate | Lock waits | Deadlocks | Disk usage | Log growth | Table and index bloat

A dashboard was green for a month while users complained: latency, errors and CPU were steady on the primary, but the replicas serving the reads had fallen seconds behind, and nobody was watching them. With lag on the dashboard, it lit up in minutes.

> *If this started failing now, which signal would tell us first?*

6.2 Alert on what users feel

An alert on a metric nobody cares about trains a team to ignore alerts. An alert on CPU above seventy percent fired several times a week and never meant anything; two alerts, on replication lag and on error rate, fired less than once a month and always mattered.



four weeks: twenty-one pages for nothing, against one that mattered

> *Would this alert wake someone for a reason they would agree with?*

6.3 One identifier, through everything

A correlation id in the database, application and proxy logs. A slow request then took ten minutes to explain: a query waiting on a lock, held by a transaction waiting on a payment provider, slow because of an unrelated outage. Without the id it would have taken days.

> *Could an operator reconstruct this request from the logs alone?*

CHAPTER 07

PERFORMANCE AND CAPACITY

Slow queries lengthen locks and lag, and growth surprises only the team that never measured it. Both yield to the same habit: understand the workload before tuning it.

7.1 The plan is the truth

Read plans with actual timings and buffers. A query took 200 ms for small customers and 20 s for large ones: the source was fine, the planner's nested loop was not. One index on the join column brought every customer under 100 ms.

> *Where does the time actually go, according to the plan?*

7.2 Index with discipline

Every index costs every write. The mistake I see most is not the missing index but the one added for a report and never removed when the report was. An events table taking five thousand inserts a second carried fourteen:



four serve a query in use; ten served reports long retired

dropping the ten doubled the write throughput

> *Which query does this index serve, and what does it cost on writes?*

7.3 Settings that match the workload

Memory and autovacuum settings are not universal. Moved from 64 GB to 256 GB of memory, a database got slower: its settings still described the old machine, and the planner assumed disk reads it no longer needed.

> *What is the workload, and do these settings match it?*

7.4 Capacity, with realistic load

A load test passed against a hundred thousand rows; production had forty million, and the plan changed at that scale. Test against data shaped like production, and keep doing it as production grows.

100 K

Rows in the test
Well within budget.

40 M

Rows in production
A different plan.

30×

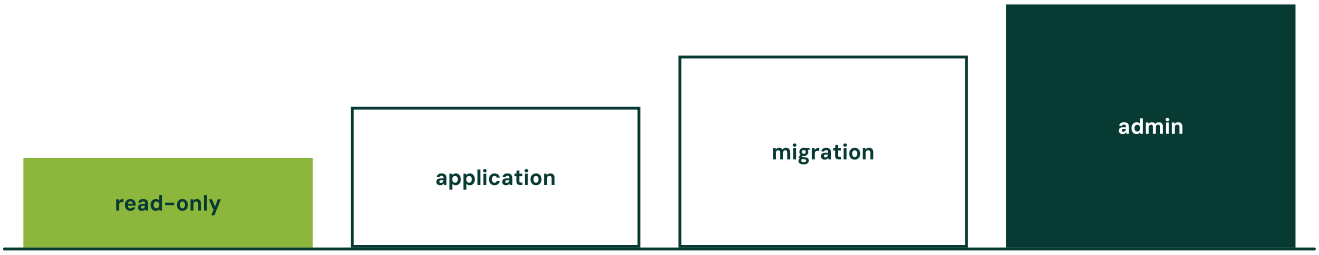
Slower
What does this look like at three times today's load?

CHAPTER 08

SECURITY
AND LEAST PRIVILEGE

Over-privileged roles magnify every bug and breach. A strong posture is many small, disciplined choices, not one large control.

The separation of roles is not distrust. It limits the blast radius when something goes wrong, and something always does.



each role only as tall as its job; the reporting service needed the smallest

8.1 Roles, least privilege

Read-only, application, migration and admin roles, each with only what it needs. An audit where every service used the superuser found that none needed it; the read-only reporting API had kept it for two years because that was how it first worked. Fixing it took an afternoon.

> *What is the smallest permission this role can do its job with?*

8.2 TLS, and rotation

Encrypt every connection and rotate credentials; rotation is the part that gets skipped. A password committed to a private repository travelled with it when the repository was shared. Rotating took an hour; auditing everything that credential had done took weeks.

> *How old is this credential, and what if it leaked today?*

8.3 Audit what matters

Schema changes, permission changes, exports. Asked who dropped a column a report relied on, a team with rotated logs needed three days to reconstruct it. The audit log need not be elaborate; it must exist, with the right fields, for long enough.

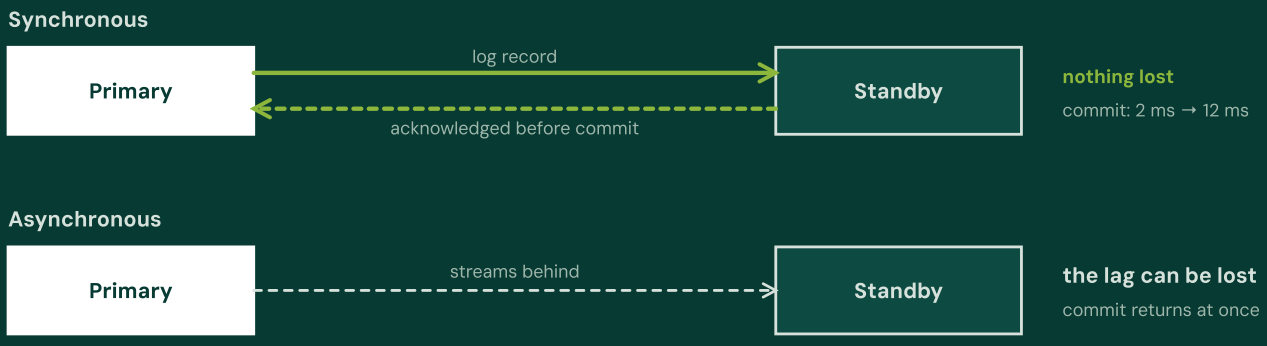
> *If it happened last night, would we know who did it?*

A credential in a repository is shared *with everyone the repository is ever shared with.*

CHAPTER 09

HIGH AVAILABILITY AND FAILOVER

Your availability design is your real uptime and your real data loss, and it rests on two numbers: how much you may lose, and how long you may be down.



9.1 Choose replication by acceptable loss

Streaming the write-ahead log is the usual baseline; logical replication, shared disks, block-level mirroring and triggers each trade something else. The decision that matters is synchronous or not. A ledger team took 2 ms to 12 ms commits for zero loss, and a failover later proved it. A year on, another team asked to switch it off, and nobody could find why it was on.

> *What loss is acceptable on failover, and which technique delivers it?*

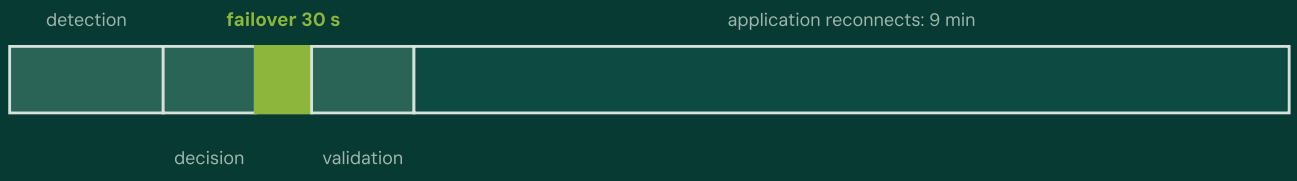
9.2 Own the whole failover

Replication does not fail over; a cluster manager detects, promotes and reconfigures, whether you run one yourself (Patroni, repmgr, Stolon) or a provider does. Running it yourself is control that asks for discipline; a managed service asks for less, and owns less. One provider promoted in two seconds; the application, pinned to a hard-coded host, took nine minutes to follow.

> *Who owns detection, promotion and reconnection, and is it written down?*

9.5 RTO is end to end

Recovery time is detection, decision, failover, validation and reconnection together. One system failed over in thirty seconds and still took ten minutes, because the application retried every thirty seconds, twenty times, before it noticed.

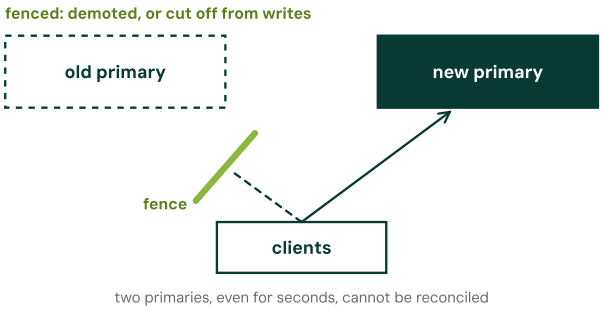


the database did its job in thirty seconds; the system took ten minutes

9.3 Split-brain and fencing

The hard part is not promotion but making sure the old primary stops. A drill once showed it taking writes for seconds after the switch: the partition that caused the failover had also cut off the fencing signal.

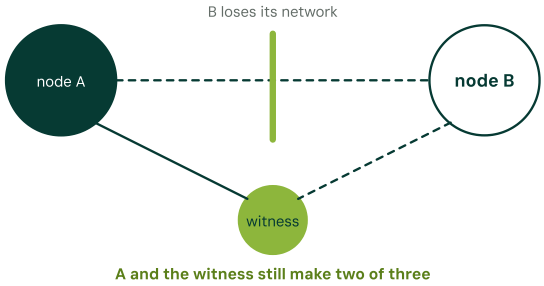
> *Cut off from the cluster, could the old primary still take writes?*



9.4 Quorum

An odd number of consensus nodes, and a witness when the database nodes are even. A two-node cluster lost one node's network and could not elect a leader, though both were healthy; a witness in a third zone made the same failure a non-event.

> *How many nodes can fail before no leader can be elected?*



9.6 Drill, until it is boring

Quarterly drills, recorded. The point of the fourth, clean drill was the three before it:

1. A firewall rule missing
2. A DNS update missing
3. Reconnection too slow
4. Clean

A failover path that has never been exercised *is a hypothesis, not a plan.*

CHAPTER 10

DATA QUALITY
AND MIGRATION SAFETY

Silent corruption, or half a migration, can be worse than downtime.

Trust depends on integrity, not only uptime, and integrity is protected by validation before the cutover, not after it.

10.1 Compare old and new

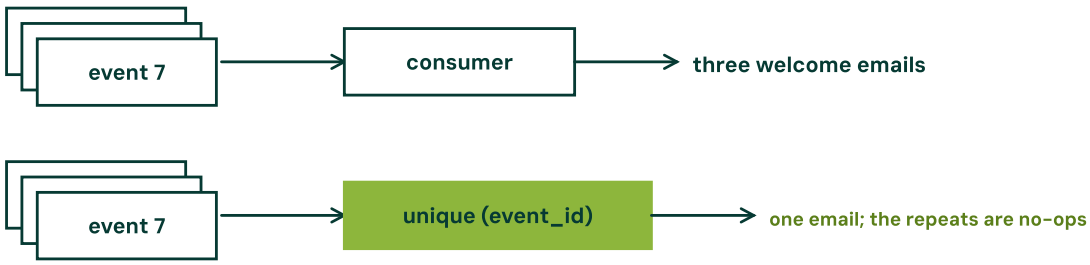
Row counts, checksums and key business queries, automated and reviewed. A billing migration matched row for row and was signed off; a week later the monthly totals drifted, because a tax rule for a few customers had been lost. Every row was there. The values were not.

Row count	Checksum
matched exactly	would have caught it in minutes

> *What evidence says the new system holds the same data?*

10.2 Change data capture delivers at least once

The lag is the visible part; the duplicates are the incident. A consumer that was not idempotent handled one event three times after a network blip. One unique constraint on the event's id fixed it: anything that can be retried must be safe to retry.



10.3 Write the criteria first

Decide "successful" and "roll back now" before the migration starts. One team wrote three criteria; at cutover two passed and one key query differed. It was written as a rollback trigger, so they rolled back, fixed it within the week, and succeeded on the second attempt, with no debate at 2 a.m.

- Every table: rows and checksums match
- Key business queries: same answers
- Errors under threshold for an hour

> *What would make us roll back, and is it written down?*

A row count says the rows are there. *A checksum says they are right.*

CHAPTER 11

OPERATIONAL HYGIENE

Neglected vacuuming, bloat and old versions arrive as performance cliffs. Routine work, done before it is urgent, is far cheaper than an emergency.

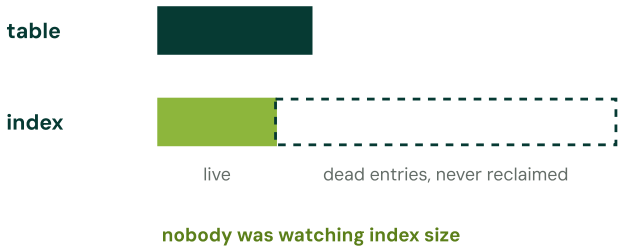
11.1 Vacuum and analyze

Autovacuum is a background process, and not always right. A heavily updated table grew while its row count stayed flat; its thresholds were set for a smaller, slower table. No error, just a query slower every week. A lower scale factor and a higher cost limit, for that table only, let vacuum keep up.

> *Is autovacuum keeping up with this table, and if not, why?*

11.2 Measure the dead space

Bloat is not an emergency the day it is noticed; it is one the day it stops being noticed. An index three times its table's size was reading needless pages and crowding the cache. An online reindex fixed it at once.



> *How much of this is live data, and how much dead space?*

11.3 Rehearse upgrades on a production-shaped load

The upgrade is not the hard part; the rehearsal is. A major version upgrade passed on staging, then on the night chose a different plan for a key query under production's workload: ten times slower, rolled back. The second rehearsal used a production-shaped load, caught the plan, and the upgrade went through.

> *Has this been rehearsed on a system that matches production?*

A rehearsal without realistic load *is a rehearsal of half the system.*

CHAPTER 12

RUNBOOKS
AND CHANGE MANAGEMENT

Documentation is not a compliance artefact. It is the team's memory, and memory is what outlives the people.

In an incident, a good runbook reduces panic and mistakes, and a new engineer can operate safely without tribal knowledge.

12.1 Runbooks, tested

Architecture and data flows, migration steps with checks and rollback, incident playbooks. A restore runbook two years old pointed at a script that no longer existed; a one-hour restore took three. Test the runbook like the backup it describes.

> *Could someone new recover this from the runbook alone?*

12.2 Decision records

Synchronous replication, a rollback window's length, a backup strategy: each has context and rejected alternatives. Unrecorded, the replication choice of chapter 9 took two weeks to defend; recorded, five minutes.

> *Why was this decided, and what was rejected?*

12.3 Reviews with context

A review is a second pair of eyes, not a gate. A not-null constraint was approved in five minutes by someone who did not know the column had never been filled for existing rows. A reviewer who knew the data would have caught it in one.

> *Who reviewed this, and did they know the data?*

The record of a decision, in five lines.

Decision Synchronous replication for the ledger

Costs commit latency, 2 ms → 12 ms

Revisit if the loss tolerance changes

Because no committed transaction may be lost

Rejected asynchronous, for the lag it can lose

The memory of a team is shorter than the life of its system; *the written record closes the gap.*

CHAPTER 13

HOW IT ALL WORKS TOGETHER

The disciplines form a sequence, each resting on the one before.

01
Purpose
What the database is for, before how it is built.

02
Compatibility
Old and new application, one schema.

03
Bounded locks
Timeouts, short transactions, fail fast.

04
Protected data
Both backups, tested restores, objectives per service.

05
Observed
The signals that matter, alerts users would agree with.

06
Tuned
Plans read, indexes justified, load realistic.

07
Secured
Least privilege, TLS, rotation, audit.

08
Failure-ready
Replication by acceptable loss, fencing, quorum, drills.

09
Validated
Old against new; criteria written first.

10
Maintained
Vacuum, bloat, rehearsed upgrades, current runbooks.

A dependable database is a disciplined database: compatible changes, bounded locks, protected data, an observed system, limited access, *expected failure, validated data, routine operations.*

I have seen the opposite: migrations as quick fixes, untested backups, missing monitoring, over-privileged access, failover never tried, operations improvised. They worked, for a while, and then they did not. These are their lessons, written down so I do not have to learn them twice.



02

PART TWO

CONCEPTS
TO MASTER AND REVISIT

Part I says what to do. This part says what you are touching when you do it, from the storage layer upward, the order in which a production problem usually reveals itself.

The query is slow because the plan is wrong, the plan is wrong because the statistics are stale, the statistics are stale because autovacuum is behind, and autovacuum is behind because the table is bloated.

CHAPTER 14 · STORAGE AND TRANSACTIONS

TWENTY CONCEPTS

Each with an example, because a concept without one is a definition I will have forgotten by next month, and a line on what to revisit.

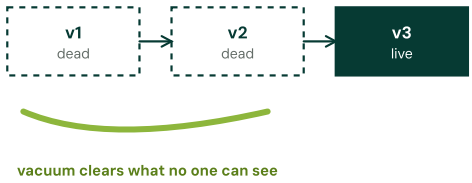
14.1 Transactions and isolation

The isolation level is a contract. Under read committed, two users booked the same seat: the check and the insert were not atomic. Serializable made it impossible, at the price of a few retries.

REVISIT which level each critical transaction uses, and whether the application handles the retries. The names transfer between engines; the behaviour does not.

14.2 MVCC and vacuum

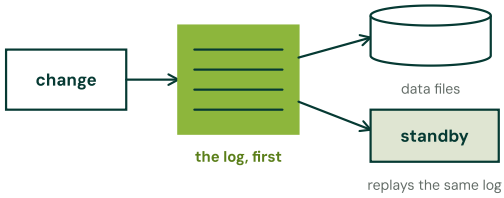
Readers and writers do not block each other because every update leaves a new version; vacuum removes the ones nobody can see.



REVISIT whether vacuum keeps up with the write pattern, and whether bloat is measured.

14.3 Write-ahead log and durability

Every change is logged before it is applied; the log is replayed after a crash, and by the standby. A bulk import with synchronous commit off lost its last rows to a power cut: a real trade, never written down.



REVISIT which operations run without synchronous commit, and whether their loss window is accepted on paper.

14.4 Planner and statistics

The plan is chosen from statistics, approximate by design. After a bulk import changed a status column's distribution, a rare value was rare no longer; one analyze and the right plan returned.

REVISIT whether analyze runs often enough for the write rate.

14.5 Index types and costs

B-tree, hash, GIN, GiST, BRIN: each a different trade, all paid on every write. A GIN index on JSON made a search fast and every insert slow; a scheduled materialized view put the cost on the other side.

REVISIT which query each index serves, and its write cost.

14.6 Normalization, and when not

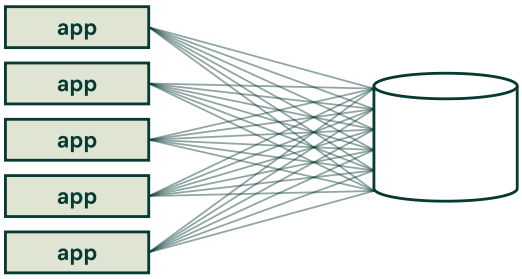
One fact in one place, or a deliberate copy to save joins. A seven-table report became a single-table read, maintained by a trigger: a second source of truth, named and chosen.

REVISIT where a fact lives twice, and whether that is deliberate.

14.7 Partitioning

A bet on the access pattern. Monthly partitions made retention a metadata operation and time-filtered queries faster; queries by user still scanned every partition.

REVISIT whether queries filter on the partition key.

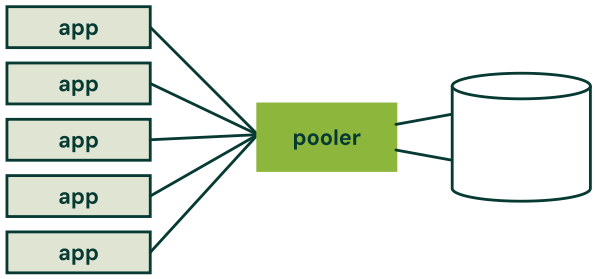


10 × 200 = 2,000 connections

14.8 Connections and pooling

Two hundred connections per instance, ten instances: fine at idle, collapse under load. Twenty per instance behind a pooler was stable, and faster.

REVISIT the total across all instances, against what the database can serve.



10 × 20, multiplexed: work, not context switches

14.9 Replication lag and read consistency

A replica is behind by contract. Reports moved to one became fast, and wrong, by seconds at peak. Fresh-data reports went back to the primary.

REVISIT which reads can tolerate staleness, and whether lag is measured.

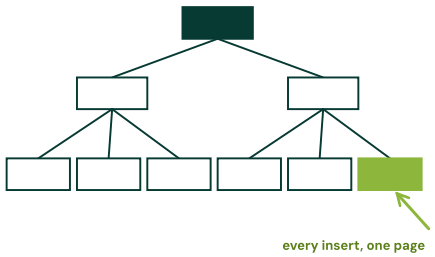
14.10 Lock modes and queues

The queue is the problem: a waiting exclusive lock blocks even the requests compatible with the current holder. Chapter 4 is the remedy.

REVISIT whether every migration sets a short lock timeout.

14.11 B-tree pages and splits

A strictly increasing key sends every insert to the rightmost page, which splits again and again. A small spread in the key restored the write throughput.



REVISIT whether the primary key is monotonic, and where inserts land.

14.12 Cardinality and selectivity

With ninety-five percent of rows active, a sequential scan was right; a hint forcing the index made it slower. The planner had been right, and was overruled on intuition.

REVISIT whether the planner is trusted, or overruled without evidence.

14.13 Covering indexes

Answer from the index alone, if it holds every column and the visibility map is current. Adding status to an existing index made the dashboard's slowest query one of its fastest.

CHAPTER 14 · DISTRIBUTION AND RECOVERY

14.14 Two-phase commit

Correct and expensive. A partition left transactions prepared and their rows locked until a person decided which to commit. It does not avoid thinking about failure; it makes the failure explicit.

REVISIT whether prepared transactions are monitored.

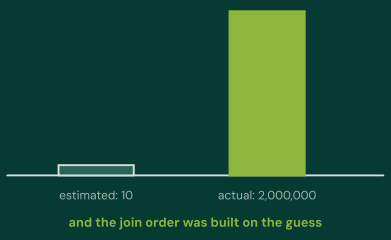
14.15 Idempotency keys

Exactly-once is built, not delivered: a key stored with its result, and a unique constraint. A retried payment request without one charged a customer twice.

REVISIT which operations can be retried, and whether each is safe.

14.16 Estimates and join order

Errors compound through a join. A five-table query took two seconds, not twenty milliseconds, on a guess like this:



REVISIT the estimates in slow plans, read rather than assumed.

14.17 Caches and invalidation

A cache has a staleness window, and it is a contract. Prices cached for an hour made price changes hourly; invalidating on change restored what the business assumed.

REVISIT the window, and whether the business agreed to it.

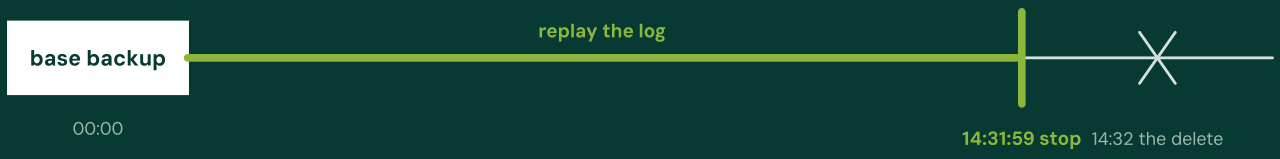
14.19 Schema versioning

The migration table is a record; bypass it once, by hand during an incident, and the next deploy fails on a column that already exists.

REVISIT whether every change goes through the tool.

14.18 Point-in-time recovery

A base backup and the log since it. A delete without a where clause at 14:32; replayed to 14:31:59.



forty minutes; no committed work lost

CHAPTER 14 · WHERE TO START

14.20 The cost of a query

Rows read and returned, pages from cache and from disk, CPU per row, time holding locks. A query that read ten rows took two seconds: ten serialized disk reads, the cache full of other data. Read in pages, not in source.

REVISIT slow queries in rows, pages, cache and disk.

MASTER, REVISIT, RECOGNISE

Not every concept deserves the same attention.

MASTER FIRST

They explain most of the surprises I have seen in production.

- Transactions and isolation
- MVCC and vacuum
- Planner and statistics
- Indexes and their costs
- Locks and lock queues

REVISIT OFTEN

Their right setting drifts with the workload: re-measure on every change.

- Autovacuum and bloat
- Connection pooling
- Lag and read consistency
- Cache invalidation
- Point-in-time recovery

RECOGNISE

They matter rarely, and a great deal when they do.

- Partitioning
- Two-phase commit
- B-tree internals

HOW A PROBLEM USUALLY UNWINDS



Which concept does this incident actually involve, *and which one am I assuming?*

CHAPTER 15

BEFORE YOU DEPLOY

A pre-deploy and pre-migration checklist. A memory aid, not a religion: ask the questions that matter for the change at hand.

PURPOSE

- Is the capability this change supports stated?
- Is the rollback window agreed with the business?
- Is the on-call engineer named?

DATA PROTECTION

- Is the backup recent?
- Has a restore been tested off production?
- Has the restored system been seen to boot?
- Do RTO and RPO match the business agreement?

PERFORMANCE

- Has the plan been reviewed?
- Does each index serve a real query, worth its write cost?
- Do memory and autovacuum settings match the workload?
- Was it load tested on production-like data?

MIGRATION

- Is the change backward-compatible?
- Which phase is it in, and is that phase finished?
- Does the script set lock and statement timeouts?
- Is the backfill batched, resumable and throttled?
- Is the rollback plan written and tested?

OBSERVABILITY

- Are latency, errors, locks, lag and disk alerting?
- Do requests carry a correlation id?
- Would each alert wake someone for a good reason?

SECURITY

- Does every role have least privilege?
- Is TLS enforced, and are credentials rotated?
- Are sensitive operations audited?

DATA QUALITY

- Are old and new compared, checksums included?
- Is change data capture checked for lag and duplicates?
- Are success and rollback criteria written?

DOCUMENTATION

- Is the runbook current?
- Are important decisions recorded?
- Has the change been reviewed, by someone who knows the data?

HIGH AVAILABILITY

- Does replication match the acceptable data loss?
- Is failover automated, with fencing?
- Is there quorum?
- Has failover been drilled recently?

OPERATIONS

- Is autovacuum keeping up?
- Is bloat tracked and managed?
- Has the next upgrade been rehearsed?

Print these two pages. *Ask the questions that matter for this change.*

CHAPTER 16

REFERENCES
AND FURTHER READING

MIGRATIONS: EXPAND AND CONTRACT

1. Sato, D. (2014). *Parallel change*. martinfowler.com. The expand, migrate, contract pattern by name. martinfowler.com/bliki/ParallelChange.html
2. Harness. *Zero-downtime database migrations: safe schema changes*. harness.io/blog
3. SchemaSmith. *Zero-downtime database migrations guide*. schemasmith.com/guides
4. IaaStha. *Zero-downtime database migrations*. iaastha.com/engineering
5. jholt1055. *How I migrated a 25 TB production database with zero downtime*. Medium. medium.com/@jholt1055
6. Chattopadhyay, S. *Zero-downtime database migrations*. LinkedIn. linkedin.com/posts

LARGE MIGRATIONS, CHANGE DATA CAPTURE, VALIDATION

7. Ghosh, A. *Zero-downtime database migration: the definitive guide*. DEV. dev.to/ari-ghosh
8. SystemDR. *Database migration strategies: zero downtime*. systemdr.systemdrd.com
9. Mydbops. *Zero-downtime database migrations in high-traffic systems*. mydbops.com/blog
10. Srivastav, S. *Data engineering interview: database migration*. LinkedIn. linkedin.com/posts

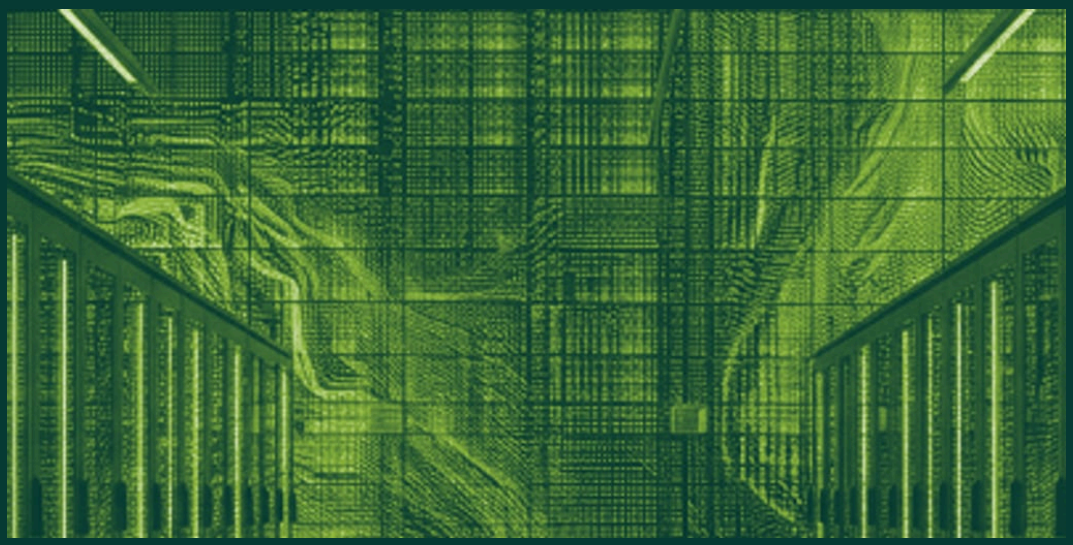
OPERATIONS, AVAILABILITY, OBSERVABILITY

11. MinervaDB. *PostgreSQL DBA daily checklist*. minervadb.xyz
12. Beyer, B. et al. (2016). *Site Reliability Engineering*, ch. 6, Monitoring distributed systems: alert on symptoms. O'Reilly. sre.google/sre-book
13. Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly. Replication, isolation, distributed transactions.
14. *Patroni documentation*. patroni.readthedocs.io
15. Leach, B. (2017). *Designing robust and predictable APIs with idempotency*. Stripe. stripe.com/blog/idempotency

POSTGRESQL DOCUMENTATION, FOR PART II

16. *Concurrency control*. postgresql.org/docs/current/mvcc.html
17. *Write-ahead logging*. postgresql.org/docs/current/wal.html
18. *Using EXPLAIN*. postgresql.org/docs/current/using-explain.html
19. *Index types*. postgresql.org/docs/current/indexes-types.html
20. *Routine vacuuming*. postgresql.org/docs/current/routine-vacuuming.html
21. *Continuous archiving and point-in-time recovery*. postgresql.org/docs/current/continuous-archiving.html
22. *Explicit locking*. postgresql.org/docs/current/explicit-locking.html

The Database Engineer's Production Playbook, first edition, October 2026. Written by Mouad Krikbou. AI helped with formatting and proofreading. Cover photograph supplied by the author; its variations and the diagrams were made for this guide.



Engineer by trade, artist by habit.