KEDBYTE

SOFTWARE · AI · AUTOMATION

HOW DATA WORKS

From a Written Record to a Global Database

The plain-English guide to data systems,
followed by the engineer's version of the same thing.

WRITTEN BY

Shikhar Singh

Founder & Chairman

KedByte Technologies Private Limited

Volume H · Chapters 47–52 · First Edition
Running Data Systems

HOW DATA WORKS

From a Written Record to a Global Database

Author	Shikhar Singh
Designation	Founder & Chairman, KedByte Technologies Private Limited
Published by	KedByte Technologies Private Limited
Imprint	KedByte Press
Edition	First Edition, September 2026
Subject	Data systems, databases, software engineering and general reference
Extent	Eight parts, fifty-two chapters

Examples. Mira's Corner and all shop records, identifiers, prices, people and events are invented teaching material. They are not customer data or observations of a live commercial system. New examples state their own assumptions and do not silently replace earlier facts.

Accuracy and currency. Technical references identify versions or consultation dates where available. Software and documentation change. Local tests exercise stated examples in the recorded environment; they do not verify every sentence, implement every production safeguard or establish compatibility with every software version.

Educational scope. This book and its code are for learning. They are not a production service, legal opinion, security certification or promise of recovery from every failure. Use isolated practice environments and obtain appropriate authority before inspecting or changing a real system.

Preparation. Prepared with AI assistance for Shikhar Singh and KedByte Press. The publisher's accompanying quality report records the checks performed and their limits. Independent technical, legal and accessibility certification is not claimed.

VOLUME H

Running Data Systems

Operate, change, protect and retire a data system, then assemble the complete case study.

This volume contains Chapters 47–52 of the complete book. Chapter and section identifiers remain unchanged. Its local page numbers differ from the complete edition. The volume glossary covers its own chapter vocabulary; the source register is shared across the series.

How to read this book

The shape of every section

Every main teaching section uses the same six blocks. The labels describe ways to approach an idea, not different classes of reader.

Block	What it is for
PLAIN — in simple words	The problem and central idea in ordinary language.
PLAIN — a picture in your head	An everyday comparison, followed by its explicit limits.
PLAIN — a worked example	Concrete records, quantities or steps that can be checked.
PLAIN — what is really happening inside	The actual mechanism, explained without a jump in assumed knowledge.
TECHNICAL — the engineer's version	Precise vocabulary, implementation details, assumptions and source references.
WORDS — remember these	Each term connected to both an everyday and a technical meaning.

Two ways to read it

1. **One continuous pass.** Start at Chapter 1 and read every block. The later engineering treatment grows out of the example already introduced.
2. **Two passes.** Read the PLAIN and WORDS blocks first, then return for the TECHNICAL blocks. The browser reader provides modes for both routes. Practice and chapter summaries remain visible in either mode.
3. **Question-led reference.** Use the contents, chapter search or glossary to locate a subject. Read the nearby assumptions before borrowing a formula, query or design pattern.

Conventions used throughout

1. Main sections have stable identifiers such as 26.4. Their six learning blocks extend that identifier, for example 26.4.5 for the engineering treatment. Chapter and section numbers stay constant across the complete edition, individual volumes and website.
2. Numbered paragraphs separate ideas. An analogy includes a statement of where it breaks. The technical treatment should deepen the simple explanation rather than silently contradict it.
3. A product-specific behavior is not presented as a universal database rule. PostgreSQL examples refer to the documented version; SQLite examples identify their separate implementation and tested runtime.
4. A source marker such as [S25] points to the source register. Consultation dates belong to those records. Unversioned documentation can change; the included test report identifies the environment actually exercised.

5. Worked shop examples are synthetic. A table of amounts is not evidence of payment settlement, real customer activity or a production incident.
6. Every chapter ends with practice and worked answers, common wrong ideas, and a summary of twenty numbered entries. A summary entry may wrap onto more than one printed line.
7. Code blocks may be complete executable fragments, queries requiring the stated fixture, or explicitly labeled conceptual traces. The companion-lab guide identifies the implemented exercises and their boundaries.
8. A successful local test establishes only its asserted property. It is not independent review, complete security assurance or certification of a production service.

One shop, growing demands

Mira's Corner is a fictional stationery shop. Mira owns it and Dev helps at the counter. The canonical four agreed order lines comprise six items and 28,650 paise: O-1042 totals 17,100 and O-1043 totals 11,550. Taxes, discounts, delivery fees and settlement records are deliberately outside that initial fixture.

The later catalogue-price example does not rewrite historical agreements. The earlier expected-stock-10/observed-stock-9 discrepancy remains unexplained. Later race conditions and capstone transactions are separate demonstrations, not invented explanations of that discrepancy.

The capstone adds synthetic tenant identifiers T-A and T-B. These represent separate organizational scopes; a branch identifier is not a substitute for tenant authorization. CAP-001 is a separate order whose two notebooks and one pen happen to total the same 17,100 paise as the opening example.

Where to find things

1. Chapters 1–6 establish meaning, records, representations, measurements, identity and history.
2. Chapters 7–13 develop files, databases, schemas, constraints, SQL and relationships.
3. Chapters 14–20 follow queries through scans, indexes, plans, reports and performance measurements.
4. Chapters 21–26 examine transactions, overlapping work, isolation, locks, versions and idempotency.
5. Chapters 27–32 trace physical storage, logs, compaction, durability, backup and repair.
6. Chapters 33–39 explain replication, failover, partitioning, consistency, consensus, distributed transactions, caches and queues.
7. Chapters 40–46 develop pipelines, streams, analytical storage, text and vector search, and careful interpretation of results.
8. Chapters 47–52 cover permissions, retention, migrations, operation, the integrated case study and the reference desk.
9. The A–Z glossary, companion-lab guide and source register follow the chapters. The browser edition also offers keyword and full chapter-text search.

Edition and evidence

This first edition contains all 52 chapters and was prepared with AI assistance for Shikhar Singh and KedByte Press. The quality report records local checks, not independent certification. The PDF fixes the layout; Word and browser pages reflow. Use stable chapter and section identifiers across formats.

Contents

PART H

Running Data Systems

Operate, change, protect and retire a data system, then assemble the complete case study.

47

CHAPTER

Permissions, Secrets and Tenant Boundaries

47.0 What this chapter gives you

1. Finding a record and being allowed to read it are different questions. A correct query can return information to the wrong person. This chapter adds an explicit permission decision to the data machinery developed so far.
2. You will distinguish identity verification, action permission, database privileges and tenant separation. You will trace a request across the application, database, cache, export and background worker rather than treating the login screen as the whole boundary.
3. The examples extend the fictional shop into a teaching service with two tenants, T-A and T-B. They are independent organisations in this example, not automatically two branches of the same organisation. A branch restriction may be an additional rule inside a tenant.
4. The companion capstone trusts a small principal object supplied by its tests. It demonstrates scoped operations in an isolated SQLite database. It does not implement login, session verification, PostgreSQL row security or a production security perimeter.

47.1 Who may do what

47.1.1 PLAIN — in simple words

1. Authentication asks who is making a request. Authorisation asks whether that person or program may perform this particular action on this particular information now. Knowing a person's name does not settle the second question.
2. An order may be readable by a shop assistant but not changeable after acceptance. An accountant may see totals without seeing delivery notes. A support operator may investigate a failed job without downloading every customer's records.
3. Write permissions as specific actions. "Can access orders" is incomplete: reading one order, listing all orders, exporting them and changing their prices have different consequences.
4. A safe decision needs a trustworthy identity, the requested action, the target resource and relevant context. When essential context is absent or invalid, do not invent a permissive interpretation.

47.1.2 PLAIN — a picture in your head

1. Imagine a building receptionist who recognises Dev. Recognition lets the receptionist establish who arrived; it does not give Dev the keys to every room.
2. A separate assignment says which rooms Dev may enter and what work he may perform there. The assignment can expire even though Dev remains the same person.

3. **Where the comparison breaks:** software has many entrances that do not look like doors: APIs, exports, scheduled jobs, search endpoints and cached responses. Hiding a button blocks none of those routes by itself.

47.1.3 PLAIN — a worked example

1. Give the synthetic principal dev-a membership in T-A and the actions order.read and order.create. Give reviewer-a order.read only. Neither principal belongs to T-B.
2. dev-a reading T-A order CAP-001 is permitted if the order exists and any additional branch rule passes. reviewer-a creating a new order is denied even inside T-A. dev-a reading T-B CAP-001 is also denied, despite an identical local order identifier.
3. Now revoke dev-a's membership. The next operation must use the service's declared revocation policy. A cached membership valid for another hour would create a one-hour revocation delay; that delay must not be described as immediate revocation.
4. A compact decision table should include a valid same-tenant action, a forbidden action, a wrong tenant, an absent principal, an expired assignment and an unknown resource. These are different cases, not six names for one test.

47.1.4 PLAIN — what is really happening inside

1. The application receives a credential or authenticated session, resolves a principal and evaluates a policy. It then performs the action using only the allowed scope. The client-supplied URL or form is not authoritative evidence of that scope.
2. A policy can depend on role, ownership, membership, branch, lifecycle state or an explicit grant. Each fact has an origin and a freshness requirement. Conflicting or stale facts need a defined decision, not whichever answer is convenient.
3. Decisions and actions can be separated by time. If the target or assignment changes between them, the earlier decision may no longer justify the later action. Where that matters, recheck relevant conditions at the protected operation or bind them to a versioned transaction boundary.

47.1.5 TECHNICAL — the engineer's version

1. OWASP distinguishes authentication from authorisation and recommends default denial, permission checks on every request and tests for access-control logic. Those principles motivate, but do not fully specify, our original permission matrix. [S185]
2. Model a decision as allow(principal, action, resource, context). A successful decision is not transferable to a different resource, operation or tenant unless the policy explicitly grants that wider scope.
3. The local capstone receives a trusted Principal value directly from its test harness. Constructing that object is not authentication. A real adapter must validate its session or token, issuer, audience, expiry and relevant membership before it invokes the operation.

47.1.6 WORDS — remember these

1. **Principal:** the person or program asking — an authenticated or otherwise explicitly established security identity used in policy evaluation. **Authorisation:** permission for a particular act — evaluation of a principal's rights over a resource in a defined context. **Revocation delay:** time before a withdrawn permission stops working — the exposure window introduced by cached or long-lived authority.

47.2 Least privilege

47.2.1 PLAIN — in simple words

1. Give each person and program the permissions needed for its job, not every permission that would make development easier. A reporting task should not need to rewrite orders merely to read totals.
2. Separate routine work from exceptional maintenance. The program serving customer requests and the operator changing table definitions have different responsibilities and should not automatically share one powerful account.
3. A permission can spread through membership in another group. Removing one direct grant does not necessarily remove the same permission inherited elsewhere.
4. Least privilege is a continuing maintenance task. Old integrations and temporary repair accounts can accumulate privileges after their original purpose has ended.

47.2.2 PLAIN — a picture in your head

1. Mira gives the cleaner a key to the storage room, not authority to alter the shop's agreements. The locksmith needs broader access for a repair, but that access is temporary and supervised.
2. Keeping the locksmith's master key beside the till would erase the benefit of carefully issued smaller keys.
3. **Where the comparison breaks:** software privileges can be inherited through roles, functions and deployment tools. The visible account name may conceal more authority than its direct permissions suggest.

47.2.3 PLAIN — a worked example

1. Define three illustrative database identities: app_runtime, report_reader and schema_operator. app_runtime performs approved application changes; report_reader runs scoped reads; schema_operator performs reviewed schema work outside ordinary request handling.
2. Suppose report_reader belongs to a broad legacy group with UPDATE privileges. Revoking its direct UPDATE grant does not demonstrate that updates are impossible. Inspect and test the effective privilege path, including inherited rights.
3. A migration that creates a new table must also decide its owner and grants. Testing the old tables says nothing about the new table until its effective permissions have been checked.
4. An operator review records the identity, permitted operations, scope, expiry where appropriate and a successful denied-operation test. A screenshot of a role name alone proves little.

47.2.4 PLAIN — what is really happening inside

1. Database privileges govern operations on objects such as schemas, tables, columns and functions. Ownership and role membership can supply additional authority beyond a single visible grant.
2. Applications often connect using a shared database identity. In that arrangement, the database does not automatically know which human is behind each request. The application must preserve that distinction or establish a trustworthy database-level context.
3. Powerful maintenance code can legitimately cross normal boundaries. Its input handling, caller checks and execution identity therefore deserve separate review; calling it a helper does not make its authority harmless.

47.2.5 TECHNICAL — the engineer's version

1. PostgreSQL 17 documents privileges, object ownership and GRANT/REVOKE semantics. Object privileges and row-level policy are complementary: the former controls allowed operations on objects; the latter can restrict rows for applicable commands. [S186]
2. A shared-table design requires an audit of effective runtime authority, including ownership and roles that bypass row policies. Do not assess the request path using only a superuser test session.
3. Capability-style designs can pass narrower handles instead of a global unrestricted connection. That is an application design choice, not a feature automatically supplied by naming a Python variable read_only.

47.2.6 WORDS — remember these

1. **Least privilege:** only the authority needed — minimising effective permissions and their duration for a defined responsibility. **Effective privilege:** what the identity can actually do — the result of direct grants, memberships, ownership and special execution rules. **Runtime identity:** the account serving ordinary work — the database or service identity used by the request-handling process.

47.3 Database and application checks

47.3.1 PLAIN — in simple words

1. An application can check a rule before asking the database to act. A database can also enforce selected rules when the action reaches it. These checks protect different parts of the path.
2. Repeating the same mistaken assumption in two places is not independent protection. Both layers must use trustworthy context and agree about the intended rule.
3. Database checks are especially valuable against forgotten application predicates. They still do not protect an export already copied elsewhere or an unrestricted administrator acting outside the normal path.

47.3.2 PLAIN — a picture in your head

1. The receptionist checks a visitor's assignment, and a room door checks a badge. A mistaken receptionist decision need not open the door if the badge genuinely lacks permission.
2. But if the receptionist can print any badge without restriction, the second check offers less separation than it first appears to.
3. **Where the comparison breaks:** a database policy can inspect a session setting established by the application. A caller able to change that setting may be able to impersonate another scope unless the surrounding boundary prevents it.

47.3.3 PLAIN — a worked example

1. Consider this original PostgreSQL teaching sketch, not an executed server configuration. trusted_tenant() represents a separately reviewed, non-spoofable context mechanism; it is deliberately not defined as "whatever the request says."

```
ALTER TABLE scoped_orders ENABLE ROW LEVEL SECURITY;
ALTER TABLE scoped_orders FORCE ROW LEVEL SECURITY;
CREATE POLICY scoped_order_policy ON scoped_orders
  USING (tenant_id = trusted_tenant())
  WITH CHECK (tenant_id = trusted_tenant());
```

2. USING governs which existing rows pass the policy for applicable operations. WITH CHECK constrains proposed row values. A write must not be allowed to move a row into another tenant merely because the original row was visible.
3. Test through the real intended runtime role and connection pool. Also test omitted context, reused connections and an attempted tenant change. Running the sketch as an unrestricted superuser does not establish tenant isolation.
4. This policy sketch is incomplete as a deployable configuration. Roles, table privileges, context construction, function execution rights and all other policies must be reviewed together.

47.3.4 PLAIN — what is really happening inside

1. A policy predicate participates in query processing. It may filter rows or reject a proposed modification, depending on the command and policy definition.
2. Application checks add business meaning: whether an order is editable, whether a refund needs another approval, or whether the chosen export purpose is permitted. Those meanings are not inferred from a tenant column alone.
3. Connection pooling creates a lifecycle problem. Any per-request context must be established and cleared in a scope that prevents one request from inheriting another's identity, including error and rollback paths.

47.3.5 TECHNICAL — the engineer's version

1. In PostgreSQL 17, enabled row security without an applicable policy defaults to denying row access. Superusers and BYPASSRLS roles bypass it; owners normally bypass it unless FORCE is used. Whole-table operations and integrity checks have additional documented exceptions. [S187]
2. Permissive policies combine with OR and restrictive policies with AND, within the documented applicability rules. An accidentally broad permissive policy can therefore defeat a narrow policy's intended result. [S187]
3. The SQLite capstone uses explicit application predicates instead of PostgreSQL row policies. Its tests establish behaviour through that helper, not safety against someone holding an unrestricted database connection.

47.3.6 WORDS — remember these

1. **Row-level security:** a filter or rule on individual rows — database policy enforcement beyond ordinary object privileges. **Connection context:** identity-related state attached to database work — information whose establishment, reuse and clearing form part of the security boundary. **Policy bypass:** an execution path exempt from a rule — special authority that must be accounted for when testing enforcement.

47.4 Tenant separation

47.4.1 PLAIN — in simple words

1. A tenant is a separately scoped organisation or customer group in a shared service. The service may share machinery, but one tenant must not automatically inherit another's records or authority.
2. Scope belongs in more places than a SELECT statement. It affects record identities, references, duplicate-request keys, caches, object names, exports and background jobs.
3. Two tenants may both have an order called CAP-001. Treating CAP-001 alone as globally unique can mix their work, even when each tenant's own data is internally consistent.

4. A tenant boundary and a branch boundary answer different questions. A manager allowed across branches inside T-A is not thereby allowed into independent tenant T-B.

47.4.2 PLAIN — a picture in your head

1. Two shops rent separate rooms in one warehouse. Each room has a shelf labelled 17. "Get the item from shelf 17" is incomplete without the room.
2. A delivery note must carry the room as well as the shelf. Otherwise the warehouse worker can follow the note faithfully and still take the wrong item.
3. **Where the comparison breaks:** shared services can duplicate and transform records into caches or reports. The tenant scope must survive those transformations; it is not a physical wall that stays attached automatically.

47.4.3 PLAIN — a worked example

1. The capstone uses keys such as (tenant_id, order_id) and (tenant_id, request_id). A request called R-1 in T-A is not a replay of R-1 in T-B.
2. A child line references its parent's complete scoped key. Otherwise a syntactically valid child could point to a parent in another tenant.
3. A cache key might include an explicit tuple of tenant, resource identity and representation version. For user-specific responses it may need additional permission context, or the design should avoid sharing that cached representation.
4. Searching across all tenants, selecting the top ten matches and then removing forbidden results can also change the answer. Apply authorised eligibility before the result limit when that is the intended query contract, and test empty and narrow scopes.

47.4.4 PLAIN — what is really happening inside

1. Tenant context is established at the trusted boundary and travels with the operation. A job producer records it; a worker validates that the job is allowed; every derived resource is named and queried within that scope.
2. Database constraints can prevent some cross-tenant associations through composite keys. They cannot decide whether the supplied tenant truly belongs to the authenticated principal without an authority mechanism.
3. Separate databases offer a different isolation boundary from shared tables. They also change maintenance and reporting costs. Neither architecture is automatically secure if administrative credentials, exports or backup access are shared carelessly.

47.4.5 TECHNICAL — the engineer's version

1. The original shared-table capstone includes tenant in relevant primary and foreign keys and in application predicates. Its boundary is the trusted helper interface, not arbitrary SQL.
2. OWASP's multi-tenant guidance identifies tenant context and separation across data access and shared infrastructure as concerns beyond authentication alone. Our branch/tenant fixtures are original examples of that distinction. [S188]
3. For each store, document the isolation unit, identity source, access mechanism and privileged bypass path. This inventory is more informative than a blanket claim that the architecture is tenant-safe.

47.4.6 WORDS — remember these

1. **Tenant:** one separately scoped customer organisation — an isolation domain within a service that may share infrastructure. **Scoped key:** an identifier plus its domain — a composite identity such as tenant_id and order_id. **Confused deputy:** a privileged helper misused through supplied instructions — an authority-bearing component acting for the wrong caller or scope.

47.5 Secret handling

47.5.1 PLAIN — in simple words

1. A secret is information whose possession grants a capability or exposes protected data: a database password, signing key or service token, for example. A username or public key need not be secret merely because it looks technical.
2. Keep secrets out of books, sample files, public repositories, logs and screenshots. A placeholder should be unmistakably a placeholder, not a copied working credential.
3. Restrict who can obtain each secret, which systems may use it and how long it remains valid. Replacing a leaked string in the source file does not necessarily invalidate copies already made.
4. Plan rotation and revocation before an incident. A service that cannot replace its credentials without losing all availability has a serious operational dependency.

47.5.2 PLAIN — a picture in your head

1. A safe combination should not be taped to the safe. Moving the note into a drawer helps only if everyone who could see the tape can no longer open the drawer.
2. If the combination was photographed, shredding the original note does not make the photograph stop working. The combination itself needs changing.
3. **Where the comparison breaks:** rotating an encryption key is not always equivalent to changing a password. Existing ciphertext may still require an older key or a carefully managed re-encryption process.

47.5.3 PLAIN — a worked example

1. The production design proposal gives the reporting job a short-lived, read-scoped credential obtained from a protected service identity. The book supplies no such credential and makes no call to a secret manager.
2. A rotation rehearsal records when the replacement became usable, which consumers moved to it and when the old authority was revoked. A successful new connection does not by itself demonstrate that old access is disabled.
3. Suppose a log captured an authorization header. Treat log storage and its readers as part of the exposure, revoke the affected credential under the incident procedure and review why redaction failed. Merely deleting one log line is not a complete response.
4. The lab uses synthetic principal names only. Its absence of passwords is not proof of a secure login implementation; login is outside the exercise.

47.5.4 PLAIN — what is really happening inside

1. A secret has a lifecycle: creation, distribution, use, rotation, revocation and disposal. Each step has permissions, evidence and failure cases.

2. Applications should log operational facts without recording the capability itself. Error handlers, tracing middleware and debug dumps deserve attention because they can expose values the normal path hides.
3. A deployment system can inject secrets without placing them in source, but the runtime still has some means of using them. Restricting process access, crash dumps and administrative inspection remains relevant.

47.5.5 TECHNICAL — the engineer's version

1. OWASP's secrets-management guidance covers centralised handling, lifecycle controls, access restriction and rotation. A specific deployment must still establish how workload identity reaches that mechanism and how a failed rotation behaves. [S189]
2. Distinguish authentication credentials from encryption keys and signing keys. Their consumers, compromise consequences and retirement procedures differ; one generic rotate_secret function does not settle all three.
3. The public companion code contains no production credentials. Its tests do not validate an external secret manager, hardware security module or production key-retirement procedure.

47.5.6 WORDS — remember these

1. **Secret:** protected capability-bearing information — a credential or key requiring controlled disclosure and use. **Rotation:** replacing active secret material — transitioning consumers to new authority under an explicit lifecycle. **Revocation:** withdrawing previously valid authority — disabling future accepted use, distinct from deleting one stored copy.

47.6 Adversarial boundary tests

47.6.1 PLAIN — in simple words

1. A test that an authorised user succeeds is only half the story. Also test that a nearby unauthorised request fails without changing or revealing protected data.
2. Change one important input at a time: tenant, object identifier, action, membership, version or lifecycle state. That reveals which boundary caused the decision.
3. Repeat the tests through every supported path. A secure detail page does not prove a secure export, bulk endpoint or background worker.
4. A rejected response is not enough if the operation already changed data. Inspect the protected state as well as the returned error.

47.6.2 PLAIN — a picture in your head

1. Mira tests a locked room by entering with the correct key, then trying a wrong key, an expired pass and an open side entrance.
2. She also checks whether the attempted entry changed the lock or left the side door open. Hearing "access denied" from a guard does not establish what happened elsewhere.
3. **Where the comparison breaks:** timing, error text and aggregate counts can reveal information without returning a whole record. A test suite has to name the observable channels it actually checks.

47.6.3 PLAIN — a worked example

1. Create T-A CAP-001 and T-B CAP-001 with deliberately different synthetic contents. Read using an authorised T-A principal and assert that only T-A's contents appear.
2. Attempt a T-B read and write with that same principal. Assert denial and compare both tenants' records before and after. A cross-tenant write must not quietly become a successful zero-row business operation.
3. Reuse R-1 independently in both tenants. Assert that each tenant receives its own result. Reuse R-1 inside T-A with a changed payload and assert a conflict without a second order.
4. Supply a principal missing the required action and inspect that no request/outbox record was created. These controls make the capstone's scoped helper behaviour observable, while leaving real identity-provider tests to a deployment-specific adapter.

47.6.4 PLAIN — what is really happening inside

1. The test constructs known protected data, invokes one path under known authority and compares outputs and state with explicit expectations. Its evidence concerns that configuration and path.
2. A bypass test deliberately invokes a lower layer to show the boundary. The SQLite connection can execute unrestricted SQL, so possession of that connection is outside the helper's tenant guarantee.
3. Release evidence should identify the tested revision and runtime. A passing result from last week's code does not automatically describe today's additional export endpoint.

47.6.5 TECHNICAL — the engineer's version

1. Organise tests by principal × action × resource scope × state. Add negative and boundary cases rather than only generating many random inputs without a policy oracle.
2. Record denied-operation effects, error-shape expectations and tested interfaces. Avoid describing a narrow unit suite as penetration testing or an independent security audit.
3. The final companion inventory identifies the executable checks. The production checklist separately calls for authentication integration, pooled-context lifecycle tests, real database role tests, export/worker coverage and review of privileged paths. [S185]

47.6.6 WORDS — remember these

1. **Negative control:** a case that should be refused — a deliberately unauthorised or invalid operation with a checked outcome. **Policy oracle:** the expected permission decision — the independently stated rule used to judge a test result. **Boundary coverage:** which protected paths were exercised — an inventory of tested interfaces, scopes and states, not a universal guarantee.

47.97 Practice and worked answers

1. **Question:** dev-a belongs to T-A. Does a caller-supplied tenant_id of T-B change that membership? **Answer:** No. Scope comes from trusted authority, not from the target supplied in a request.
2. **Question:** Why is (tenant_id, request_id) preferable to request_id alone in this shared-table example? **Answer:** Independent tenants can legitimately choose the same request identifier; their replay histories must not collide.
3. **Question:** What is missing from a successful PostgreSQL test run as superuser? **Answer:** Evidence about the intended restricted role, since superuser bypasses ordinary row-policy enforcement.

4. **Question:** Does hiding the export button prevent an unauthorised export request? **Answer:** No. The server-side export path needs its own enforced permission boundary.
5. **Question:** Does replacing a leaked password in a configuration file revoke existing copies? **Answer:** Not necessarily. The credential's authority must be invalidated and affected consumers updated under the incident procedure.
6. **Question:** What does a zero-row update establish? **Answer:** Only that no row matched the applied operation. It is not automatically a successful business update or a sufficiently private denial.
7. **Question:** Can a branch manager in T-A read all of T-B? **Answer:** Not under the example's policy. Branch scope within one tenant does not grant another tenant's authority.
8. **Question:** What is the strongest claim from the capstone's access tests? **Answer:** The stated scoped helper accepts and rejects the supplied synthetic cases in its tested runtime. It does not implement or certify a complete security perimeter.

47.98 Common wrong ideas

1. **Wrong:** Logged in means allowed. **Right:** Each protected action still needs a permission decision.
2. **Wrong:** A difficult-to-guess identifier is access control. **Right:** Knowledge of an identifier is not permission to use it.
3. **Wrong:** Every row-policy test should use the administrator account. **Right:** Test the actual runtime role and its exceptions.
4. **Wrong:** Tenant filtering belongs only in the main table. **Right:** Derived stores and background paths need scoped identities too.
5. **Wrong:** Revoking a direct grant removes every route to authority. **Right:** Inheritance, ownership and special execution paths may remain.
6. **Wrong:** A secret removed from a repository is no longer exposed. **Right:** Existing copies and remaining authority require separate treatment.
7. **Wrong:** An error proves that nothing changed. **Right:** Inspect effects and transaction outcomes.
8. **Wrong:** Passing unit tests means the application is secure. **Right:** Evidence is bounded by the paths, roles, configuration and threats actually tested.

47.99 Chapter summary in 20 lines

1. Authentication and authorisation answer different questions.
2. Name the principal, action, resource and relevant context.
3. Establish tenant scope at a trusted boundary.
4. Deny operations whose required authority is absent.
5. Separate reading, writing, exporting and administrative actions.
6. Treat revocation freshness as a declared operational property.
7. Minimise effective privileges, not only visible direct grants.
8. Separate runtime and schema-maintenance identities.
9. Application and database checks protect different layers.
10. Test database policies using the intended restricted role.
11. Account for ownership and bypass exceptions.
12. Prevent pooled connections from retaining another request's context.
13. Include tenant scope in relevant keys and references.

14. Preserve that scope through caches, jobs, search and exports.
15. Keep branch restrictions distinct from tenant boundaries.
16. Give secrets a complete controlled lifecycle.
17. Rotation and revocation are not the same as deleting a file.
18. Test refused operations and their effects on stored state.
19. State exactly which interfaces and threats a test covers.
20. A bounded teaching helper is not a complete production security system.

48

CHAPTER

Retention, Deletion and Data Lineage

48.0 What this chapter gives you

1. Information has a useful lifetime. Keeping it forever can create unnecessary exposure; deleting it carelessly can destroy evidence, break references or make recovery impossible. The job is to define and implement a lifecycle rather than choose one slogan.
2. You will separate hiding a record, deleting a logical row, removing derived copies and sanitising storage media. You will also learn why an empty query result is not a universal certificate of erasure.
3. The policies and dates in this chapter are synthetic engineering examples. They are not legal retention periods, advice about a jurisdiction or a statement that a particular real organisation may delete a record. Actual legal, contractual and operational requirements need their own assessment.
4. The continuing shop's agreed order facts are not silently erased or rewritten here. The deletion exercise uses a separately introduced optional contact note, NOTE-7, and a finite inventory of stores. This keeps lifecycle learning separate from the canonical order totals.

48.1 Purpose and record classes

48.1.1 PLAIN — in simple words

1. Start by asking why each kind of information exists. An order line, temporary import file, optional delivery note and access log serve different purposes and need not have the same lifetime.
2. A field can be useful at one stage and unnecessary later. Keeping a temporary instruction inside every downstream report can spread information beyond the work it was collected to support.
3. Decide the unit of lifecycle control. Sometimes an entire record is retired; sometimes an optional annotation is removed while a separately justified transaction record remains.
4. The purpose must describe actual use. Calling everything "for analytics" does not explain which question requires which fields, who may use them or when that use ends.

48.1.2 PLAIN — a picture in your head

1. Mira keeps an agreed order form, a packing checklist and a sticky note saying where to leave a parcel. The order form may support one responsibility, while the sticky note serves only a particular delivery.
2. Copying the sticky note into every monthly sales report does not make the report more correct. It creates another copy that someone must control.

3. **Where the comparison breaks:** digital transformations can preserve identifying information indirectly through keys, small groups or joins. Removing a visible name does not automatically make the remaining data anonymous.

48.1.3 PLAIN — a worked example

1. Introduce three fictional record classes: agreed_order_line, optional_contact_note and failed_import_payload. Give each an owner, stated purpose, access scope and lifecycle rule. No real legal period is assigned by the book.
2. NOTE-7 belongs to tenant T-A and a synthetic contact. Its purpose is one temporary follow-up. It is not required to calculate O-1042 or O-1043, whose agreed amounts remain 17,100 and 11,550 paise.
3. If NOTE-7 is copied into a search index and a support export, those are additional lifecycle responsibilities. The source table alone is no longer the complete inventory.
4. A record-class card can state: class, purpose, owner, collected fields, allowed uses, authorised readers, lifecycle trigger, exceptions, stores and evidence required for closure. A missing owner is itself an unresolved operational question.

48.1.4 PLAIN — what is really happening inside

1. Schemas and pipelines decide where a field travels. A lineage description connects original inputs, transformations and derived outputs so lifecycle work can follow those routes.
2. Separating optional annotations from durable agreement facts can make independent lifecycle actions easier. The split must preserve valid references and document what an order means after an annotation disappears.
3. Minimisation can occur before storage as well as afterward. A transformation that never copies an unnecessary field avoids a future deletion problem in that destination.

48.1.5 TECHNICAL — the engineer's version

1. Treat classification as an application policy, not a property automatically inferred from a database type. TEXT can contain harmless labels or sensitive free-form notes; the engine does not determine their purpose.
2. OpenLineage's object model distinguishes jobs, runs and datasets, providing a vocabulary for recording input/output relationships. It does not itself prove that every copy has been discovered or that a deletion was executed. [S193]
3. The chapter's class cards and deletion inventory are original teaching designs. Their adequacy depends on the declared system boundary and the requirements supplied by its owners.

48.1.6 WORDS — remember these

1. **Record class:** a group with a common responsibility — data categorised by purpose, access and lifecycle requirements rather than file extension alone. **Purpose limitation:** using information for a defined need — an explicit boundary on collection and downstream use. **Data minimisation:** keeping only what the task needs — reducing unnecessary fields, copies or lifetime under a stated requirement.

48.2 Retention policies

48.2.1 PLAIN — in simple words

1. A retention rule needs a start event, a duration or end condition, and an action. "Delete after thirty days" is ambiguous until we know thirty days after what.
2. Creation time, completion time and last legitimate activity may be different dates. Choosing the wrong clock can retire active work or keep abandoned information indefinitely.
3. Exceptions need an explicit basis, owner and review point. A permanent unexplained hold can turn a time-limited policy into "keep forever" without anyone making that decision openly.
4. A scheduled job is an implementation of a policy, not the policy itself. The organisation must still decide what should happen and verify that the job does it.

48.2.2 PLAIN — a picture in your head

1. A library loan lasts fourteen days after checkout, not fourteen days after the book was printed. The event starting the clock changes everything.
2. A renewal changes the due date under a rule. Someone rubbing out the date on the card is not a valid renewal process.
3. **Where the comparison breaks:** a digital object can be referenced by several workflows with different completion conditions. One clock may not describe all legitimate uses of shared information.

48.2.3 PLAIN — a worked example

1. For NOTE-7 only, invent a policy of expiry thirty elapsed days after its follow-up is closed. Let the synthetic closure instant be 1 September 2026 at 12:00 UTC. The resulting expiry is 1 October 2026 at 12:00 UTC.
2. This is elapsed-time arithmetic. A policy based on local calendar dates could produce a different boundary and must specify its time zone and treatment of daylight-saving transitions where relevant.
3. A hold on NOTE-7 suspends the deletion action under the chosen policy, but the hold needs a reason code, owner and review deadline. The automated system should not invent the reason.
4. Test one instant before expiry, exactly at expiry and after expiry. Also test an absent closure time and an active hold. In our example, unknown timing means review required, not an invented old date.

48.2.4 PLAIN — what is really happening inside

1. A policy evaluator maps current record facts and policy version to an eligibility decision. The execution worker then attempts the corresponding action within a bounded batch.
2. Eligibility can change between selection and action. A newly recorded hold, for example, must not be ignored simply because the row appeared in yesterday's export. Recheck conditions at the operation boundary where the design requires it.
3. Record the policy version used. When a policy changes, previously scheduled work may need reevaluation rather than blindly executing an obsolete list.

48.2.5 TECHNICAL — the engineer's version

1. Use explicit aware instants for elapsed-time examples and retain the original event-time semantics. A timestamp's storage format does not decide whether the policy is elapsed-time or calendar-based.

2. Our companion evaluator rejects invalid types and negative durations, treats a missing closure instant as unresolved and makes the equality boundary explicit. Those checks test the arithmetic policy only, not a legal obligation.
3. Batch deletion should be restartable with bounded work and an auditable selection predicate. A large unqualified DELETE is not a lifecycle programme merely because it runs every night.

▪ 48.2.6 WORDS — remember these

1. **Retention trigger:** the event starting a lifecycle rule — a defined transition such as closure, not an assumed timestamp. **Hold:** an explicit suspension of an action — a governed exception with reason, authority and review conditions. **Policy version:** which rule was applied — an identity connecting a lifecycle decision to the exact governing definition.

48.3 Deletion across copies

▪ 48.3.1 PLAIN — in simple words

1. Hiding a row from a screen, marking it inactive and deleting it from a table are different actions. None automatically reaches every copy made earlier.
2. A search index, cache, export, replica, analytics table or backup may retain information after the original screen stops showing it.
3. A lifecycle workflow therefore needs a store inventory and a clear action for each store. Unknown destinations remain unknown; they cannot become completed tasks by omission.
4. The strongest honest completion statement names its boundary: which stores, which record identities, which actions and which checks were completed.

▪ 48.3.2 PLAIN — a picture in your head

1. Mira removes a note from the counter. Dev still has a photocopy, and a courier has a photograph. The empty counter says nothing about those copies.
2. A checklist helps only if it includes the places the note actually travelled. Ticking every item on an incomplete checklist does not prove that no other copy exists.
3. **Where the comparison breaks:** storage engines can retain old physical representations that ordinary queries cannot read. Logical visibility and recoverability from media are separate layers.

▪ 48.3.3 PLAIN — a worked example

1. The synthetic inventory for NOTE-7 contains an active table, search projection, short-lived cache and controlled export. Each task has a scoped key and a status: pending, completed, failed, blocked or unknown.
2. Complete the active-table deletion and search removal. Suppose cache invalidation fails and the export owner has not confirmed its action. The overall status is not complete, despite two successful tasks.
3. Retrying a completed task should be safe under the chosen operation contract. However, “record not found” is meaningful only after the worker has confirmed the correct tenant, store and identifier.
4. The companion inventory model checks these status transitions and refuses an empty inventory as proof of completion. It does not contact a cache, delete a real export or inspect an operator’s computer.

48.3.4 PLAIN — what is really happening inside

1. Database deletion changes logical visibility under the engine's rules. Physical storage reclamation may occur later and can depend on old readers, logging, compaction and maintenance.
2. PostgreSQL describes how deleted or superseded row versions remain until vacuum can reclaim their space. Reclaiming space is not the same claim as sanitising all historical copies or media. [S190]
3. SQLite's secure_delete setting concerns overwriting certain deleted content in database pages, with documented limits including some virtual-table arrangements. It is not a promise about every journal, backup, filesystem snapshot or external copy. [S191]

48.3.5 TECHNICAL — the engineer's version

1. Separate logical deletion, application inaccessibility, engine-level reclamation and media sanitisation in both requirements and evidence. A single Boolean deleted column cannot represent every layer honestly.
2. NIST SP 800-88 Revision 2 describes media sanitisation in terms of making target data access infeasible at a stated effort level. It is a media-management reference, not proof that an application row deletion has fulfilled every lifecycle requirement. [S192]
3. A deletion ledger can itself become sensitive. Keep only the identifiers and evidence necessary for its purpose, restrict access and give that ledger a lifecycle too.

48.3.6 WORDS — remember these

1. **Logical deletion:** removal from ordinary active data — a change to the database's visible record state. **Sanitisation:** rendering target media data infeasible to recover under a stated standard — a different layer from an empty application query. **Store inventory:** the declared destinations to address — an explicit set whose completeness must be assessed, not assumed.

48.4 Derived data and lineage

48.4.1 PLAIN — in simple words

1. Derived information is built from other information: a daily total, search document, recommendation feature or cleaned export. Understanding its origins helps us correct or retire it.
2. Some transformations can be updated by removing one contribution. Others need recomputation or a carefully defined approximation. The transformation determines the work.
3. A derived result can remain identifying or sensitive even after obvious names disappear. Its meaning, granularity and joinability matter.
4. Lineage is a map of relationships, not evidence that the mapped action has already happened.

48.4.2 PLAIN — a picture in your head

1. A recipe card lists ingredients used in a cake. It tells Mira which batches might contain one recalled ingredient.
2. The card does not remove the ingredient from a baked cake. The appropriate action might be withdrawing or replacing the whole batch rather than trying to subtract it.
3. **Where the comparison breaks:** some digital aggregates can be adjusted exactly from sufficient retained contributions, while others cannot. A count, median and trained model have different update mechanics.

48.4.3 PLAIN — a worked example

1. A synthetic report stores sum=300 and count=3 for contributions 50, 100 and 150. Removing the 100 contribution gives sum=200 and count=2, so the mean remains 100. Keeping only the original mean would not generally permit this update.
2. A median stored without its contributing values cannot generally be corrected by subtracting one number. The system needs enough retained state or a recomputation path under the applicable lifecycle policy.
3. A search projection that embeds NOTE-7 must be rebuilt or removed under its own index update rules. A later successful source query does not demonstrate that the searchable projection is gone.
4. A lineage record should identify source versions, transformation version and output identity. "Produced by yesterday's job" is insufficient when the job was rerun with different inputs.

48.4.4 PLAIN — what is really happening inside

1. A transformation may combine many records and create new identities. Lifecycle propagation requires a mapping from the affected source scope to the corresponding outputs or a method of rebuilding all affected outputs.
2. Shared outputs raise policy questions. Retiring one person's optional note need not delete unrelated aggregate facts, but the retained result needs an independently justified meaning and risk assessment.
3. Data written outside instrumented pipelines may be absent from automated lineage. Manual exports and debug copies must not disappear from the inventory simply because a lineage tool did not observe them.

48.4.5 TECHNICAL — the engineer's version

1. OpenLineage run and dataset metadata can document transformation relationships. The record must still be populated correctly, and incomplete instrumentation yields incomplete observations. [S193]
2. Distinguish exact incremental maintenance from approximate correction. An exact sum/count representation supports the stated example; a sketch or model requires its own update and error contract.
3. Do not infer model unlearning, anonymisation or comprehensive deletion from a successful DELETE statement. Those claims need methods and evidence specific to the retained representation.

48.4.6 WORDS — remember these

1. **Lineage:** where a result came from — recorded relationships among inputs, transformations, executions and outputs. **Incremental maintenance:** updating a result from a change — applying a justified delta instead of recomputing everything. **Recomputation:** rebuilding from an authorised source set — creating a new derived result under a recorded transformation version.

48.5 Backups and exceptions

48.5.1 PLAIN — in simple words

1. Backups preserve older states for recovery. That purpose can conflict with an expectation that an active-data deletion instantly reaches every historical snapshot.

2. State the policy for retained backups explicitly: access restrictions, expiry, permitted recovery use and what happens to previously deleted information after restoration.
3. Restoring a backup can reintroduce information removed since the backup was made. Recovery therefore needs a lifecycle reconciliation step, not only an integrity check.
4. An exception is not permission for arbitrary reuse. A restricted retained copy should not quietly become a normal source for analytics or customer support.

48.5.2 PLAIN — a picture in your head

1. Mira seals an old notebook in a recovery box. It contains a note later removed from the working notebook.
2. Opening the box after a flood restores useful history, but also restores that old note. The recovery procedure must know which later lifecycle decisions to reapply.
3. **Where the comparison breaks:** encrypted backups, key escrow, incremental chains and storage snapshots complicate selective removal. A practical policy must match the actual storage architecture rather than assume every copy is independently editable.

48.5.3 PLAIN — a worked example

1. A synthetic backup is taken at checkpoint B-20. NOTE-7 is retired afterward under lifecycle action D-21. Restoring B-20 without considering D-21 can bring the note back into active use.
2. In the exercise design, the recovery process restores into isolation, loads the authorised post-backup lifecycle decision set, reapplies applicable actions and checks the result before serving traffic.
3. The lifecycle decision record must contain enough information to prevent reintroduction while not preserving unnecessary original note content. Its access and retention need separate rules.
4. If the recovery team lacks the required decision set, it records that gap and keeps the affected restored data out of ordinary use until resolved. It does not mark the lifecycle work complete because the database starts successfully.

48.5.4 PLAIN — what is really happening inside

1. Backup policies describe retained states and recovery paths; lifecycle policies describe permitted uses and actions. The two have to be designed together.
2. Deleting one encrypted key can make selected ciphertext unavailable only under strict assumptions about key copies, derivation, plaintext copies and scope. Destroying a shared key can also make unrelated necessary records unrecoverable.
3. Record exceptions with their scope, owner, reason and review conditions. The system should distinguish retained-but-restricted from active-and-permitted, rather than hiding both behind one deleted flag.

48.5.5 TECHNICAL — the engineer's version

1. Cryptographic erasure is a key-management and sanitisation claim with prerequisites, not a generic synonym for deleting a row. NIST's sanitisation guidance provides the relevant media/key context; it does not validate our application example. [S192]
2. The companion's restore test uses a fresh SQLite destination and synthetic data. It demonstrates logical state comparison and a subsequent lifecycle decision; it does not test cloud retention locks, key destruction or physical-media recovery.

3. Evidence should identify restored backup, recovery time boundary, applied lifecycle actions and unresolved retained copies. A broad "everything deleted" certificate would exceed those observations.

48.5.6 WORDS — remember these

1. **Reintroduction:** retired information returning through recovery — restoration of an older state without later lifecycle decisions. **Restricted retention:** kept for a bounded exception — retained data whose access and use differ from ordinary active records. **Cryptographic erasure:** making protected ciphertext unusable by sanitising necessary key material — a claim dependent on key scope and remaining copies.

48.6 Evidence of completion and remaining limits

48.6.1 PLAIN — in simple words

1. Completion evidence should describe what was attempted, what succeeded and what remains unresolved. A reassuring message without those details is not a reliable record.
2. Each action needs a target identity, policy basis, execution outcome and suitable check. A failed worker and a completed worker are not interchangeable because both stopped running.
3. Keep the certificate narrower than the evidence. "Removed from these three active stores" can be supported while "no copy exists anywhere" cannot.
4. When new destinations are discovered, the inventory and outstanding work change. A prior completion result may remain historically true for its old scope without being sufficient for the expanded scope.

48.6.2 PLAIN — a picture in your head

1. Mira signs a checklist saying she inspected rooms A, B and C. The statement should not say she inspected the entire building if she never entered room D.
2. If room D is discovered later, she adds a new inspection task rather than erasing the history of what she actually checked.
3. **Where the comparison breaks:** absence is often harder to observe in distributed stores than presence. Search results may be stale, access-limited or scoped incorrectly, so the verification method matters.

48.6.3 PLAIN — a worked example

1. A synthetic completion record for NOTE-7 states tenant T-A, policy version RP-3, inventory version INV-2 and three completed active-store tasks. It separately lists one backup retained under a restricted exception until its configured expiry.
2. That record supports "active-store actions completed under INV-2; restricted backup remains." It does not support "all copies permanently erased."
3. A task marked unknown prevents the inventory model from reporting complete. A task marked blocked requires its exception to remain visible. Empty task sets are not treated as successful comprehensive action.
4. The companion tests those status rules and checks their deterministic outputs. It cannot verify an unobserved third-party store; its evidence is deliberately limited to the model and local exercise.

48.6.4 PLAIN — what is really happening inside

1. Lifecycle work is a state machine with evidence attached to transitions. Repeated delivery and retries should preserve action identity without manufacturing additional authority.
2. An append-only audit trail can retain action outcomes while the original content disappears. The trail's identifiers, actor information and reasons still need access and retention controls.
3. Operational review compares expected destinations with observed executions, reconciles gaps and assigns follow-up responsibilities. The useful result is a bounded, inspectable claim rather than an absolute adjective.

48.6.5 TECHNICAL — the engineer's version

1. A completion predicate can require a nonempty declared task set and successful verification for every required task, while reporting exceptions separately. The task set's completeness remains an external premise.
2. Store the evidence method, not merely a Boolean: query scope, target version, execution result and timestamp, with sensitive values minimised. Confirm that the check itself ran under authority capable of seeing the intended target.
3. Publication of a deletion receipt does not create proof of independently controlled copies. Separate controllable scope, verified scope, restricted retention and unknown scope in the final statement.

48.6.6 WORDS — remember these

1. **Completion predicate:** the exact rule for saying an action is done — a condition over declared tasks, evidence and exceptions. **Verification scope:** what a check could actually observe — the stores, identities, versions and authority included in the observation. **Residual copy:** information remaining after selected actions — a retained, failed, blocked or unknown destination requiring explicit treatment.

48.97 Practice and worked answers

1. **Question:** Does hiding NOTE-7 from a screen remove its search projection? **Answer:** Not unless the projection's own update or deletion path is completed and checked.
2. **Question:** Is the example's thirty-day period a legal recommendation? **Answer:** No. It is invented elapsed-time arithmetic for teaching policy implementation.
3. **Question:** Why specify the retention trigger? **Answer:** Creation, closure and last activity can produce different expiry dates and different permitted actions.
4. **Question:** Can an unchanged aggregate mean prove that no contribution was removed? **Answer:** No. In the sum/count example, removing 100 from 50,100,150 leaves the mean at 100 while changing the population.
5. **Question:** Does an empty deletion-task list establish complete erasure? **Answer:** No. The inventory may be missing; the model rejects that vacuous conclusion.
6. **Question:** What risk follows restoring B-20 after action D-21? **Answer:** An older copy can reintroduce retired information unless the recovery workflow reapplies relevant later lifecycle decisions.
7. **Question:** Does vacuum establish media sanitisation? **Answer:** No. Space reclamation and physical recoverability are different claims.
8. **Question:** What should a completion record say when a restricted backup remains? **Answer:** Name the completed active scope and disclose the retained backup exception and its governing conditions.

48.98 Common wrong ideas

1. **Wrong:** Retention is one period for the whole database. **Right:** Different record classes can have different purposes and rules.
2. **Wrong:** Missing timing means the record is old enough to delete. **Right:** Unknown timing requires a declared unresolved-data policy.
3. **Wrong:** Deleting a row reaches every copy. **Right:** Derived and historical stores require explicit treatment.
4. **Wrong:** A removed name makes every remaining record anonymous. **Right:** Joinability, granularity and context can preserve identifying meaning.
5. **Wrong:** A lineage map executes deletion. **Right:** It records relationships; execution and verification are separate.
6. **Wrong:** A backup restore is complete when the engine starts. **Right:** Later lifecycle and access decisions may need reconciliation.
7. **Wrong:** A deletion ledger can safely keep everything forever. **Right:** The ledger also has purpose, access and retention requirements.
8. **Wrong:** An absolute erasure claim follows from a few successful queries. **Right:** State the stores and observation boundary actually verified.

48.99 Chapter summary in 20 lines

1. Give each record class a purpose and an owner.
2. Separate optional annotations from independently required facts.
3. Minimise unnecessary fields and downstream copies.
4. Define the event that starts a retention clock.
5. Distinguish elapsed durations from calendar rules.
6. Govern exceptions with reasons and review conditions.
7. Recheck changing eligibility at the action boundary.
8. Hiding, logical deletion and sanitisation are different actions.
9. Inventory caches, indexes, exports, analytics and backups.
10. Unknown destinations do not count as completed work.
11. Record the transformation history of derived outputs.
12. Preserve sufficient state for justified incremental corrections.
13. Recompute when a simple subtraction is not valid.
14. Do not infer anonymity or model unlearning from a removed field.
15. Make backup retention and permitted use explicit.
16. Prevent restored historical data from bypassing later lifecycle decisions.
17. Treat key destruction as a scoped, prerequisite-dependent claim.
18. Give lifecycle evidence its own access and retention policy.
19. Report completed, restricted and unresolved scope separately.
20. A completion statement must not exceed its actual evidence.

49

CHAPTER

Schema Changes Without Surprises

49.0 What this chapter gives you

1. A schema change alters the agreement between stored records and the programs using them. The challenge is not merely to make a new table definition valid; it is to keep old data and overlapping application versions meaningful while the change takes place.
2. You will plan compatibility windows, separate expansion from removal, design bounded backfills and define evidence for cutover. You will also distinguish undoing a deployment from reconstructing information already lost.
3. The worked change adds an explicit currency field to a separate copy of the shop's INR-only teaching data. It does not change the agreed amounts or pretend that a real multi-currency checkout has been implemented.
4. All executable changes in the companion use newly created, disposable databases. No instruction in this chapter authorises running a migration against an existing user or production database.

49.1 Compatibility windows

49.1.1 PLAIN — in simple words

1. During an update, not every program changes at the same instant. An old web process, new worker, reporting job and delayed mobile request may all use the database during the transition.
2. A safe plan asks which combinations must work. A new program with the old schema and an old program with the new schema can fail in different ways.
3. Compatibility includes meaning, not only syntax. A field still called amount can become dangerous if one version reads rupees while another writes paise.
4. List readers and writers before changing the contract. A forgotten export script is still a consumer even if it has no place on the architecture diagram.

49.1.2 PLAIN — a picture in your head

1. Mira changes an order form while some printed pads with the old design remain at the tills. The office has to interpret both forms for a while.
2. Removing a box from the new form does not remove it from old pads. Changing the meaning of an existing box without a label can be worse than adding a new one.
3. **Where the comparison breaks:** software can keep running old code for hours, replay old messages later and cache schema assumptions. The overlap is not limited to what is visibly on the counter today.

49.1.3 PLAIN — a worked example

1. The original agreed prices are integer paise and the teaching contract says INR. A proposed schema adds currency_code so future readers no longer need that implicit assumption.
2. Version V1 knows only amount_paise. Version V2 knows amount_paise and currency_code. During the overlap, a V1 insert may omit the new field; a V2 reader must not falsely claim that every missing value in every dataset means INR.
3. For this specific copied fixture, documented provenance establishes that all old records are INR. A compatibility adapter may use that provenance while the backfill runs. Imported records without the same provenance remain unresolved.
4. The compatibility matrix therefore includes V1 read/write before and after expansion, V2 read/write during partial backfill and V2-only work after cutover. It also records when V1 writers are no longer allowed.

49.1.4 PLAIN — what is really happening inside

1. Readers depend on field presence, type and meaning. Writers depend on accepted inputs, defaults and constraints. Schema and application releases change those dependencies at different times.
2. A message or API contract can outlive the program that originally created it. Replaying old payloads must follow a declared version adapter or rejection policy, not accidental interpretation by current code.
3. Database locks and resource use are a second compatibility dimension. A logically compatible DDL statement can still block ordinary work for longer than the service permits.

49.1.5 TECHNICAL — the engineer's version

1. Model compatibility as a relation among application version, schema version, payload version and data state. "Backward compatible" is incomplete unless the direction and supported combinations are stated.
2. PostgreSQL's ALTER TABLE operations have operation-specific lock and rewrite behaviours. A change plan must consult the exact operation and version rather than assume every ADD COLUMN is operationally identical. [S194]
3. The capstone migration experiment tests a small local subset. It is not an online production migration or evidence of acceptable lock duration under concurrent load.

49.1.6 WORDS — remember these

1. **Compatibility window:** the overlap period for contracts — the time during which specified old and new consumers must coexist. **Consumer:** anything relying on the data contract — a reader, writer, job, export, API adapter or replay process. **Semantic compatibility:** preserving meaning across versions — agreement about units and interpretation, not only accepted syntax.

49.2 Expand and contract

49.2.1 PLAIN — in simple words

1. Expansion adds the new representation while preserving the old path. Contraction removes the old path only after the remaining dependencies have been checked.
2. Separating these steps gives programs time to move. It also creates temporary complexity, which needs an owner and a removal plan.

3. Two writable copies of the same fact can disagree. Decide which one is authoritative during each phase and how disagreement is detected.
4. Do not make the destructive final step the first step merely because it produces a cleaner schema diagram.

49.2.2 PLAIN — a picture in your head

1. To replace a bridge while traffic continues, first build the new crossing, direct traffic onto it and only then remove the old one.
2. Building the new bridge is not proof that every road sign now points to it. The switch requires observation and coordination.
3. **Where the comparison breaks:** two schema representations can both accept writes. Unlike passive bridges, they can create conflicting versions of the same fact unless the transition protocol controls them.

49.2.3 PLAIN — a worked example

1. Phase E1 adds nullable currency_code to the copied exercise table. Existing reads remain possible, and no historical value is guessed during the DDL step itself.
2. Phase E2 deploys a reviewed writer that supplies INR for this INR-only workflow and a reader that handles the recorded transition state. Old-writer behaviour is explicitly tracked.
3. Phase E3 backfills eligible historical rows using their known provenance. Phase E4 validates completeness and allowed values, confirms that old writers are gone and strengthens the constraint.
4. Only a later contraction removes compatibility fallback logic. That removal is a distinct decision with evidence, not an automatic side effect of the backfill completing once.

49.2.4 PLAIN — what is really happening inside

1. An additive schema permits both representations for a limited period. Application releases then change which representation is read or written.
2. Dual writes should normally be kept within one appropriate atomic boundary when they describe one fact. Separate best-effort writes can leave one representation stale after a failure.
3. Monitoring must distinguish expected transitional gaps from defects. An alert cannot interpret missing currency_code correctly without knowing the current phase and its deadline.

49.2.5 TECHNICAL — the engineer's version

1. The expand/backfill/validate/contract sequence here is an original deployment plan. Its safety depends on the stated writer controls and provenance assumptions; it is not a universal recipe guaranteeing zero downtime.
2. PostgreSQL 17 supports adding certain CHECK and foreign-key constraints as NOT VALID and validating them later. The feature does not mean "ignore all checks": new changes remain subject to the documented enforcement rules. [S194]
3. SQLite's supported ALTER TABLE operations differ from PostgreSQL's, and more involved changes may require the documented table-rebuild procedure. Do not copy PostgreSQL syntax into the SQLite lab and infer equivalence. [S196]

49.2.6 WORDS — remember these

1. **Expand:** add a compatible new path — introducing representation or capability before removing the old one. **Contract:** retire the old path — removing obsolete fields, adapters or access after dependencies are checked. **Authoritative representation:** the selected source of meaning — the version whose value governs while multiple copies coexist.

49.3 Backfills

49.3.1 PLAIN — in simple words

1. A backfill transforms existing records into a new representation. It should have explicit eligibility, ordering, batch limits and rules for partial progress.
2. A stopped backfill should be resumable without silently duplicating work or overwriting a more recent legitimate change.
3. A default is not evidence. Filling every missing currency with INR is justified only where the record's origin or governing contract establishes INR.
4. Progress must represent completed work. Advancing a cursor before the data change is committed can skip records after a failure.

49.3.2 PLAIN — a picture in your head

1. Dev adds a currency label to old order cards. He knows one labelled box contains only INR orders, but a mixed box from an external source has no such guarantee.
2. He completes and records one small batch before moving his bookmark. If interrupted, the bookmark tells him which completed batch to resume after.
3. **Where the comparison breaks:** records can change while a backfill is running. A database bookmark and a person's static pile of cards do not have identical concurrency behaviour.

49.3.3 PLAIN — a worked example

1. Copy five synthetic rows with ordered identifiers 1 through 5 and a provenance flag. Rows 1–4 have the documented INR-only origin. Row 5 has unknown origin.
2. Process at most two eligible rows per transaction. The first committed batch fills rows 1 and 2. The next fills 3 and 4. Row 5 remains unresolved rather than receiving an invented label.
3. Retry the second batch. Its conditional predicate requires currency_code IS NULL and the known origin; already-filled rows are unchanged. Idempotence here means repeating the permitted transformation does not invent additional changes.
4. Inject an exception before the batch commit. Both data changes and progress metadata owned by that transaction must roll back. A progress record outside the transaction would need its own recovery protocol.

49.3.4 PLAIN — what is really happening inside

1. A worker selects a bounded set under a stable ordering and applies a conditional update. Keyset progress can avoid some problems of changing offset-based lists, but its correct use still depends on the key and eligibility rules.
2. A competing writer can make a row ineligible or change its version. Recheck relevant conditions at update time; do not treat an earlier selection as permanent permission to overwrite.

3. Resource budgets matter. A backfill uses I/O, logs, locks and cache capacity that ordinary traffic also needs. Throttle or pause based on defined service signals rather than forcing maximum speed throughout.

49.3.5 TECHNICAL — the engineer's version

1. Specify a transformation precondition and postcondition. For the exercise: source provenance is INR_ONLY, destination is NULL, and the only permitted new value is INR. Unknown provenance is not a successful transformation.
2. Keep progress and transformed data atomic where possible, or design replay-safe reconciliation where they cross a boundary. Preserve counts for selected, changed, already-complete, conflicting and unresolved rows.
3. The companion backfill is sequential and in memory. Its rollback and replay tests do not establish correctness under every production writer schedule or acceptable WAL growth.

49.3.6 WORDS — remember these

1. **Backfill:** transforming already-stored records — population of a new representation under explicit source assumptions. **Keyset progress:** resuming from a stable ordered key — a traversal technique whose eligibility and concurrent-change boundaries still need design. **Transformation precondition:** what must be true before changing a row — the provenance, version or state justifying the operation.

49.4 Validation and cutover

49.4.1 PLAIN — in simple words

1. Validation asks whether the new representation satisfies the intended contract. A backfill process exiting successfully does not answer every validation question.
2. Compare counts, keys, values, exclusions and meaning. Equal totals can hide two offsetting mistakes.
3. Cutover changes which path the service trusts or serves. It should occur after the relevant evidence is available and while a clearly defined recovery option still exists.
4. The validation itself needs a time boundary. A clean check followed by an uncontrolled old writer can make the new state incomplete again.

49.4.2 PLAIN — a picture in your head

1. Mira compares old and new order forms before telling every till to use the new version. She checks actual identifiers and line values, not only the number of pages.
2. She also removes the old blank pads. Otherwise the next order can recreate the ambiguity she just resolved.
3. **Where the comparison breaks:** software versions can remain in worker pools, clients or retry queues after a deployment dashboard says the main service is updated.

49.4.3 PLAIN — a worked example

1. Validate the copied canonical fixture: four lines, six units, two order identities and 28,650 paise. Compare each line's scoped key, product, quantity, agreed price and currency.
2. Confirm O-1042 remains 17,100 paise and O-1043 remains 11,550. A currency addition is not permission to replace their agreed prices with current catalogue prices.

3. Count unresolved origins separately. If the five-row backfill example still contains row 5 with unknown currency, it cannot support the statement "all rows satisfy the new required currency contract."
4. Before cutover, stop or adapt every old writer, rerun the applicable validation and establish the new constraint. The order matters because uncontrolled writes between checking and enforcing can reopen the gap.

49.4.4 PLAIN — what is really happening inside

1. Validation queries observe a snapshot or sequence of snapshots according to the database and transaction settings. Their scope should be recorded so discrepancies can be interpreted.
2. A database constraint can maintain selected properties after successful validation. It does not prove that the chosen currency value reflects the real transaction unless the input provenance is sound.
3. Index creation and schema locking may require special operational treatment. Even a carefully chosen concurrent index build has failure states that a deployment script must inspect.

49.4.5 TECHNICAL — the engineer's version

1. PostgreSQL documents CREATE INDEX CONCURRENTLY restrictions and the possibility of an invalid index after failure. "The command was started" is not evidence of a usable index or successful schema cutover. [S195]
2. Bind validation evidence to source revision, schema state, record scope and runtime. A row count from one database and a constraint report from another do not compose into one verified result.
3. The companion validates a deliberately small copied fixture. Production cutover needs representative traffic, real role/constraint tests and a deployment-specific concurrency and recovery plan.

49.4.6 WORDS — remember these

1. **Cutover:** switching the authoritative path — a controlled transition from old to new representation or service behaviour. **Reconciliation:** comparing meaningfully corresponding records — checking identities, values and declared exclusions rather than one headline total. **Validation boundary:** the exact state a check describes — its database, snapshot or time, schema and input scope.

49.5 Rollback versus roll-forward

49.5.1 PLAIN — in simple words

1. Rolling back an application release can restore old code. It does not automatically restore deleted information or reverse every data change made by the new code.
2. Some changes preserve enough information to undo them. Others lose distinctions: merging fields, rounding values or deleting columns can make exact reconstruction impossible without another source.
3. A roll-forward repair applies a new corrective change instead of pretending the earlier state can simply be restored. The choice depends on evidence, compatibility and the remaining recovery paths.
4. A backup is useful only if it can restore the needed state under the allowed time and data-loss constraints. Restoring an old backup can also discard legitimate work performed afterward.

49.5.2 PLAIN — a picture in your head

1. Putting the old form back on the counter does not recover notes shredded during the update. The paper design and the recorded information are different things.
2. Sometimes the correct response is to retain the new form and repair its unclear label, rather than force every completed order back into a structure that no longer fits.
3. **Where the comparison breaks:** databases can support transactional rollback for some DDL and data changes, but behaviour depends on the engine, operation and commit boundary. Do not generalise from one product's example.

49.5.3 PLAIN — a worked example

1. Adding nullable currency_code and filling justified INR values preserves the old amount_paise field. Reverting V2 to V1 during the compatible phase may be possible under the declared reader/writer rules.
2. By contrast, replacing all exact paise with rounded rupee amounts loses information. Both 7,550 and 7,549 paise could round to 75 under one whole-rupee rounding policy. The old exact value cannot be inferred from that rounded value alone.
3. If new legitimate orders were created after a backup, restoring that backup to undo the migration may remove those orders too. The recovery plan must address the intervening work, not call it irrelevant because rollback is urgent.
4. A repair record should state which values were reconstructed from retained evidence, which were corrected by new authorised input and which remain unresolved.

49.5.4 PLAIN — what is really happening inside

1. A rollback boundary is a point before or after durable commitment. Within a transaction, supported changes can be discarded. After commitment, recovery requires another operation with its own risks.
2. Reversibility is a property of the transformation and retained evidence. An invertible encoding can be reversed; an aggregation or destructive overwrite may not be.
3. Deployment teams should rehearse the intended recovery procedure on representative disposable copies. Discovering that the old application cannot read newly accepted values during an incident is too late.

49.5.5 TECHNICAL — the engineer's version

1. Record a point of no return for each destructive step and the evidence required before crossing it. Keep application rollback, schema rollback and data restoration as separate entries.
2. SQLite's documented generalised ALTER TABLE procedure includes explicit copying, rebuilding and checks. A home-made variant that changes ordering or skips dependencies is not validated merely because the resulting CREATE TABLE looks right. [S196]
3. The companion demonstrates rollback inside its owned transaction and explicitly labels transformations that lose information. It does not claim an automated production rollback facility.

49.5.6 WORDS — remember these

1. **Rollback:** returning within a defined reversible boundary — undoing a transaction or deployment, with scope that must be stated. **Roll-forward:** correcting through a subsequent change — a new reviewed transformation rather than automatic restoration of the past. **Irreversible trans-**

formation: one that loses needed distinctions — a mapping whose original inputs cannot be recovered from the retained output alone.

49.6 Staged evidence

49.6.1 PLAIN — in simple words

1. A good migration record separates proposed work, reviewed work, executed work and verified outcomes. Those stages are related but are not the same event.
2. Preparing a script proves that a script exists. Running it proves that execution occurred. A successful exit does not prove every intended business property.
3. Preserve failure evidence. Replacing a failed report with a later green result without recording the change hides what was learned and which state was actually tested.
4. The final decision belongs to the authorised owner. A technical check supplies evidence; it does not grant itself permission to change a live system.

49.6.2 PLAIN — a picture in your head

1. A bridge drawing, an approved construction plan, a completed bridge and an inspected bridge are four different things.
2. A stamp saying "plan approved" cannot be moved onto a photograph and treated as a structural inspection of the finished work.
3. **Where the comparison breaks:** software artefacts can change with one byte, and the difference may be invisible in a screenshot. Evidence needs a reliable identity for the actual candidate tested.

49.6.3 PLAIN — a worked example

1. The proposed currency migration record lists the schema precondition, eligible provenance, maximum batch size, transaction owner, stop conditions, validation queries and recovery plan.
2. A disposable rehearsal records the exact script digest and input fixture identity, then reports changed rows, unresolved rows and rollback-control results. Its scope is the rehearsal environment.
3. A later live execution would require separate authority and live preconditions. The rehearsal's four correct rows do not establish that a larger external dataset has only four eligible rows or no exceptions.
4. After execution, verify the intended schema, row-level reconciliation and the behaviours of the actual readers and writers. Retain the before/after record and unresolved findings.

49.6.4 PLAIN — what is really happening inside

1. Evidence connects an artefact identity to an environment, an action and an observation. Losing any of those links weakens the conclusion.
2. A migration runner can automate ordering and record applied identifiers. It cannot infer a business requirement omitted from the migration design.
3. A staged process limits the size of an uncertain step. It also makes a failure easier to diagnose because the observed change is bounded and the previous state is recorded.

49.6.5 TECHNICAL — the engineer's version

1. Use a review template containing preimages, change identity, schema version, engine/driver versions, permissions, transaction boundaries, reconciliation outputs and recovery conditions. The template is an original engineering aid, not certification.
2. Distinguish idempotent transformation logic from a migration registry that simply refuses a repeated migration ID. They address different retry situations.
3. The capstone's final checks establish only the listed disposable-example outcomes. They neither access the user's database nor grant a release decision for an unrelated application.

49.6.6 WORDS — remember these

1. **Precondition:** what must be true before a change — the checked state and authority on which safe execution depends. **Artefact identity:** which exact candidate was used — a stable reference, often a digest, linking execution to reviewed bytes. **Rehearsal:** a bounded execution before live work — an experiment whose representativeness and limitations must be stated.

49.97 Practice and worked answers

1. **Question:** Why test an old writer after adding a required concept? **Answer:** It may omit the new field or supply values with old semantics during the compatibility window.
2. **Question:** Can every NULL currency become INR? **Answer:** Only where authoritative provenance establishes that meaning; unknown origins remain unresolved.
3. **Question:** What must be atomic in the backfill example? **Answer:** The owned batch changes and its committed progress record, or an alternative replay-safe protocol must handle their separation.
4. **Question:** Does 28,650 paise after migration prove every line is unchanged? **Answer:** No. Compare keys and individual values too; offsetting errors can preserve a total.
5. **Question:** Why remove old writers before final validation/enforcement? **Answer:** Otherwise they can recreate incompatible data after an earlier clean check.
6. **Question:** Does restoring old code recover rounded-away paise? **Answer:** No. Lost information needs another trustworthy source or remains unknown.
7. **Question:** What does a successful rehearsal establish? **Answer:** The recorded candidate behaved as observed on the stated disposable input and environment, not every production condition.
8. **Question:** Does a green technical report authorise a live migration? **Answer:** No. Authority and technical evidence are separate requirements.

49.98 Common wrong ideas

1. **Wrong:** A schema is used only by the newest application. **Right:** Old workers, exports and replayed payloads can remain active.
2. **Wrong:** Adding a default supplies missing historical evidence. **Right:** Defaults are rules, not recovered facts.
3. **Wrong:** Dual writing guarantees agreement. **Right:** It needs an atomic or reconciled transition protocol.
4. **Wrong:** A backfill should always run as fast as possible. **Right:** Ordinary service capacity and failure budgets still matter.
5. **Wrong:** Equal totals prove a correct migration. **Right:** Keys, values, exclusions and semantics also need comparison.

6. **Wrong:** Reverting code undoes every data change. **Right:** Committed and lossy transformations may need separate recovery.
7. **Wrong:** A migration ID proves safe repeated execution. **Right:** Registry identity and transformation idempotence are different properties.
8. **Wrong:** A prepared script is an executed and verified result. **Right:** Preparation, execution, verification and authority are separate stages.

49.99 Chapter summary in 20 lines

1. A schema is a contract with multiple consumers.
2. List old and new readers and writers.
3. Preserve units and meaning across the compatibility window.
4. Inspect operation-specific lock and rewrite behaviour.
5. Expand before removing the old path.
6. State which representation is authoritative during transition.
7. Give temporary compatibility code an owner and retirement condition.
8. Backfill only where provenance justifies the new value.
9. Use bounded batches and explicit ordering.
10. Keep committed progress aligned with committed changes.
11. Recheck update conditions when competing writers can act.
12. Preserve unresolved rows instead of guessing.
13. Validate identities and values, not only totals.
14. Prevent old writers from reopening a validated gap.
15. Treat cutover as a distinct controlled event.
16. Separate application rollback from data restoration.
17. Identify transformations that lose information.
18. Rehearse the actual recovery procedure on disposable data.
19. Bind evidence to exact artefacts and environments.
20. Technical success does not manufacture live-change authority.

50

CHAPTER

Observability, Capacity and Cost

50.0 What this chapter gives you

1. A running system produces signals about its behaviour. Useful observability connects those signals to questions: Are users completing their work? Where is time being spent? Which resource is approaching a limit? What changed before the failure?
2. You will distinguish logs, metrics and traces; calculate a capacity forecast with explicit assumptions; and separate workload measurements from guesses based on server labels.
3. Every rate, storage size and cost in the worked examples is invented for arithmetic. None is a measurement of KedByte's live products, a current hosting price or a capacity promise for a particular machine.
4. This chapter connects Chapter 20's measurement discipline with Chapter 46's metric definitions. A precise dashboard can still answer the wrong question when it misses failed attempts or mixes incompatible populations.

50.1 Signals tied to user outcomes

50.1.1 PLAIN — in simple words

1. Start with the work the service is supposed to complete. A healthy-looking processor graph does not prove that an order was recorded correctly or that a customer can retrieve it.
2. Define success, failure and unknown outcomes. A request that timed out may have committed or may not have reached the database; the timeout alone does not settle the stored result.
3. Measure the experience at a useful boundary. Server processing time excludes some network and client delays, while a browser measurement may include them.
4. An objective is a target, not an observation. The system must collect evidence to determine whether the target was met.

50.1.2 PLAIN — a picture in your head

1. Mira's delivery van has a working engine and a full fuel tank. Those facts are useful, but customers care whether the correct parcels arrive on time.
2. Engine temperature helps explain a delay; delivery completion tells her whether the service actually fulfilled its purpose.
3. **Where the comparison breaks:** digital work may complete invisibly after a timeout. Outcomes need stable identities and reconciliation, not just a single caller's immediate impression.

50.1.3 PLAIN — a worked example

1. Define a synthetic request objective: at least 99.5% of 20,000 eligible read requests return an authorised, valid response within a stated threshold during one observation window.
2. Under that definition, the allowance for nonconforming requests is 0.5% × 20,000 = 100. If 60 are nonconforming, the measured success proportion is 19,940 / 20,000 = 99.7%, with 40 requests of the illustrative allowance remaining.
3. These calculations depend on the eligibility and observation rules. Excluding failed requests because they lack a successful response record would change the denominator and could make the result misleading.
4. Record unknown outcomes separately. A claim of 99.7% cannot silently treat unobserved attempts as successful merely because no error reached the dashboard.

50.1.4 PLAIN — what is really happening inside

1. A service-level indicator operationalises a user-facing property through measured events or values. A service-level objective selects a target for that indicator over a defined window.
2. Collection occurs at a particular boundary. Load balancers, application handlers and database drivers may each see different populations and durations.
3. Reconcile those populations when interpreting a failure. An attempt can disappear before application logging, and a database commit can occur before its acknowledgement is lost.

50.1.5 TECHNICAL — the engineer's version

1. Google's SRE text distinguishes indicators, objectives and agreements and emphasises choosing indicators that represent useful service behaviour. The book's 20,000-request example is our own arithmetic, not Google's reliability target. [S197]
2. Document event eligibility, correctness checks, measurement boundary, threshold, window, missingness and aggregation. Include the specification version in the metric definition.
3. Error-budget arithmetic does not permit violating a data-integrity invariant. A latency objective and a rule against cross-tenant disclosure are different kinds of requirement, not interchangeable percentages.

50.1.6 WORDS — remember these

1. **Service-level indicator:** a measured service property — a defined observation of availability, latency, correctness or another relevant outcome. **Service-level objective:** the target for an indicator — a specified level over a stated population and window. **Error budget:** the allowed nonconformance under an objective — an arithmetic planning quantity whose definition does not override other requirements.

50.2 Logs, metrics and traces

50.2.1 PLAIN — in simple words

1. A log records an event with details. A metric summarises quantities over time or categories. A trace connects steps belonging to a particular operation across components.
2. Each helps answer different questions. A count shows how often something happened; a trace can show where one request waited; a log can explain a particular rejection.

3. Collect enough context to connect evidence without copying unnecessary private data. Request identifiers are often more useful than complete request bodies.
4. Missing signals need explanation. An absent log can mean nothing happened, collection failed, the event was sampled out or the retention window expired.

50.2.2 PLAIN — a picture in your head

1. Mira has a daily counter of orders, a diary of unusual incidents and a route sheet following one parcel through packing and delivery.
2. The counter cannot explain one damaged parcel. The route sheet cannot by itself show the whole month's failure rate. The tools complement each other.
3. **Where the comparison breaks:** digital signals can be sampled, buffered, delayed and dropped. Their apparent timing and completeness depend on the collection system.

50.2.3 PLAIN — a worked example

1. A synthetic request takes 240 milliseconds end to end. Its trace records 20 milliseconds before the application, 40 milliseconds in a queue, 130 milliseconds in database work and 50 milliseconds in other application/response work. The stated non-overlapping intervals sum to 240.
2. Real spans can overlap, so blindly adding every displayed span duration can double-count time. The example's disjoint intervals are an explicit teaching assumption.
3. The associated metric counts one eligible request and its outcome. A rejection log stores the scoped request identifier and reason category, not the secret credential or full customer note.
4. If only one in ten traces is retained, the retained traces cannot automatically supply an exact count of all requests. Use a suitable unsampled counter or a documented estimator instead.

50.2.4 PLAIN — what is really happening inside

1. Instrumentation creates events and measurements in running code. Collectors transport and process them; storage and query systems retain and expose them.
2. Correlation identifiers let investigators connect related signals. Their propagation needs care at trust boundaries, because external callers can supply misleading identifiers unless the system distinguishes trusted and untrusted context.
3. Labels determine how metrics are divided. Adding every order ID as a metric label can create a huge number of distinct time series; detailed per-order evidence may belong in a different store with appropriate access and retention.

50.2.5 TECHNICAL — the engineer's version

1. OpenTelemetry describes signals including traces, metrics and logs and their associated data models. Adopting its vocabulary or library does not automatically establish complete instrumentation or causal diagnosis. [S198]
2. Record sampling, buffering, export failure and clock assumptions. A timestamp difference across unsynchronised hosts is not necessarily a precise duration; use suitable local duration measurements and trace semantics.
3. Google's monitoring guidance identifies latency, traffic, errors and saturation as useful general signals. Our observation design must still connect those signals to the shop's actual operations and declared failure conditions. [S103]

50.2.6 WORDS — remember these

1. **Log:** a recorded event — timestamped or otherwise ordered details describing a particular occurrence. **Metric:** a measured quantity — an aggregated or sampled numeric signal with units and dimensions. **Trace:** connected steps of an operation — spans and context describing work across a request path. **Cardinality:** how many distinct labelled combinations exist — a property that can drive metric storage and query cost.

50.3 Storage and growth

50.3.1 PLAIN — in simple words

1. Storage demand depends on how much arrives, how large each retained representation is and how long it remains. Indexes, logs, copies and temporary work add to the primary records.
2. A count of students, shops or users does not determine storage by itself. One user can create thousands of records, large attachments or almost no activity.
3. Distinguish logical payload size from physical occupied storage. Compression, row overhead, fragmentation and retention can change the latter substantially.
4. A forecast should expose its assumptions and define when measurements will replace them. A neat multiplication is not a measured capacity result.

50.3.2 PLAIN — a picture in your head

1. Mira estimates shelf space from boxes arriving per day, the space per box and the days each box remains. Counting suppliers alone is not enough.
2. She also needs room for labels, aisles, returns and temporary packing. Those are not new sales, but they still occupy the building.
3. **Where the comparison breaks:** database compression and indexes do not scale exactly like cardboard boxes. Updates and maintenance can create temporary or retained representations whose size depends on engine behaviour.

50.3.3 PLAIN — a worked example

1. Invent a workload of 12,000 records per day, each with 600 bytes of logical payload, retained for 365 days. The payload-only estimate is 12,000 × 600 × 365 = 2,628,000,000 bytes, or 2.628 decimal GB.
2. Under a deliberately hypothetical 1.8× combined table/index overhead multiplier, the active representation estimate becomes 4.7304 GB. Three active copies would total 14.1912 GB under the same simplifying assumptions.
3. This does not include backups, logs, exports, attachments, temporary sort space or free capacity for maintenance. Do not add them through an unexplained universal percentage; estimate their mechanisms separately.
4. If a host has 100 GB usable and 70 GB is the chosen intervention threshold, starting at 20 GB with 2 GB/day net growth reaches that threshold in (70–20)/2 = 25 days. The 70 GB threshold is an example policy, not a universal safe limit.

50.3.4 PLAIN — what is really happening inside

1. Physical storage grows through inserts, updates, index maintenance and retained history. Deletion may make space reusable without immediately shrinking the file.

2. Copies have different lifetimes. A follower, nightly backup and exported report cannot be modelled accurately as three identical continuously updated files.
3. Measure representative data and maintenance cycles. A fresh empty database can understate steady-state overhead; an unusual bulk load can overstate typical daily growth.

50.3.5 TECHNICAL — the engineer's version

1. Keep byte units explicit: decimal GB is 10^9 bytes and GiB is 2^30 bytes. The forecast above uses decimal units throughout.
2. PostgreSQL's statistics views expose specific counters and activity information under documented visibility and timing rules. They are observations with semantics, not a direct prediction of next year's data size. [S200]
3. A useful forecast records measured baseline, growth distribution, retention action, replication factor, backup policy and uncertainty. Update it when the workload mix changes rather than treating the first spreadsheet as permanent truth.

50.3.6 WORDS — remember these

1. **Logical payload:** the data's application representation — the meaningful values before engine and operational overhead. **Net growth:** increase after relevant additions and removals — a rate whose measurement window and storage boundary must be stated. **Intervention threshold:** the point to act before exhaustion — a chosen policy leaving time and resources for a controlled response.

50.4 Saturation and headroom

50.4.1 PLAIN — in simple words

1. A system is saturated when demand pushes a relevant resource toward the point where more work mostly creates waiting, rejection or failure.
2. The bottleneck might be CPU, storage latency, memory, a connection pool, a lock or a slow external dependency. Buying more of a different resource may not help.
3. Headroom is spare usable capacity under the chosen workload and failure assumptions. It is not simply the percentage of CPU currently idle.
4. Repeated retries can increase demand while the system is already struggling. A recovery policy needs limits, not an instruction to retry everything immediately forever.

50.4.2 PLAIN — a picture in your head

1. A counter can serve customers at a finite rate. Once arrivals exceed that rate, the queue grows even though the assistant is working continuously.
2. Asking waiting customers to rejoin the queue every few seconds makes the crowd larger without producing more completed orders.
3. **Where the comparison breaks:** software can parallelise some work and has multiple coupled queues. A single service-rate number is a model, not a universal description of the whole architecture.

50.4.3 PLAIN — a worked example

1. A synthetic worker completes 100 jobs per second, while 120 arrive per second for 30 seconds. Ignoring variability and starting empty, the queue grows by (120–100) × 30 = 600 jobs.

2. After arrivals stop, draining those 600 jobs at 100 per second takes 6 seconds under the same assumptions. If arrivals instead continue at 90 per second, net drain is only 10 per second and the backlog takes 60 seconds.
3. For a stable observation window, Little's-law arithmetic relates average work in the system L, throughput λ and average time W as $L=\lambda W$, under the relevant steady-state conditions. At 80 jobs/second and 0.25 seconds average residence, L=20 jobs. This is not a p99 latency estimate.
4. If retries add 50 attempts per second to the original 120 while capacity stays 100, the backlog grows faster. Backoff, retry budgets and admission control address this feedback, but each needs an explicit business failure policy.

50.4.4 PLAIN — what is really happening inside

1. Requests wait when a needed resource is unavailable. Queueing changes latency distributions and can consume additional memory or connection capacity.
2. Rejecting or shedding some work can protect the rest, but the service must decide which operations may be rejected and how callers learn the outcome. An order with unknown commit status cannot be retried as a brand-new order without considering identity.
3. Capacity during one-node failure can be lower than normal capacity. Headroom calculations should include the failure state the service claims to tolerate.

50.4.5 TECHNICAL — the engineer's version

1. Google's overload discussion covers load shedding and mechanisms for avoiding overload amplification. The queue arithmetic here is an original deterministic model, not an experiment on Google's systems. [S199]
2. Throughput and latency should be measured together under a specified arrival process. Closed-loop clients that wait for responses can reduce their offered load as the server slows, hiding a production arrival-rate problem.
3. Report accepted, completed, rejected, timed-out and unknown attempts separately. A benchmark that silently discards failed requests may make overload appear faster rather than less reliable.

50.4.6 WORDS — remember these

1. **Saturation:** demand meeting a limiting resource — a condition producing increased waiting, rejection or reduced useful throughput. **Headroom:** usable spare capacity — margin under stated workload and failure assumptions, not a generic machine percentage. **Admission control:** deciding which work to accept — protection of bounded resources through explicit entry policies. **Retry amplification:** repeated attempts adding load during trouble — feedback that can worsen the condition callers are trying to recover from.

50.5 Cost attribution

50.5.1 PLAIN — in simple words

1. Cost belongs to resources and work over time. A monthly server price alone does not describe the full cost of a data service.
2. Include storage, backups, transfer, monitoring, support and operational effort where they matter. State what is excluded rather than hiding it in an apparently precise cost per user.
3. Shared infrastructure needs an allocation rule. Equal division, usage-based division and capacity-reservation division answer different questions.

4. A cheaper architecture is not necessarily cheaper for the required outcome. A design that saves storage but makes restoration too slow may fail its purpose.

50.5.2 PLAIN — a picture in your head

1. Mira shares a delivery van with another shop. Dividing the bill equally is simple, but one shop may use most of the route time.
2. Charging by parcel count can still be misleading when one shop's parcels are much larger. The allocation method needs to match the decision being made.
3. **Where the comparison breaks:** computing has fixed commitments, burst usage, tiered prices and shared bottlenecks. A single per-request charge may not represent marginal cost or reserved capacity.

50.5.3 PLAIN — a worked example

1. Invent monthly amounts of 6,000 currency units for compute, 1,000 for storage, 800 for backups and 1,200 for observability. The sum is 9,000 before any omitted transfer, tax, support or labour costs.
2. With 300,000 eligible completed operations, the allocated average is 9,000 / 300,000 = 0.03 units per completed operation under this chosen boundary.
3. If only 200,000 operations complete while the fixed bill remains 9,000, the same allocated measure becomes 0.045. That does not mean each additional operation physically costs 0.045; average and marginal cost differ.
4. A report should state the currency, accounting window, included charges, denominator and allocation method. These invented values are not quotes from a cloud provider or estimates for the user's website.

50.5.4 PLAIN — what is really happening inside

1. Resource meters measure selected consumption. Billing rules convert it into charges, and an attribution model allocates those charges to tenants, workflows or products.
2. A shared fixed cost cannot be uniquely divided by arithmetic alone; an allocation policy is required. The policy should be recorded so comparisons remain consistent.
3. Optimisation should preserve required outcomes. Removing all backups lowers one bill line while removing the recovery property the system was supposed to provide.

50.5.5 TECHNICAL — the engineer's version

1. Separate invoice reconciliation, allocated unit cost, marginal cost and opportunity cost. They answer different management questions and can legitimately produce different numbers.
2. Evaluate changes using a fixed workload definition and correctness constraints. A query that becomes cheap by excluding required records is not an optimisation of the original question.
3. The companion cost function checks nonnegative finite amounts and an explicit nonzero denominator. It does not connect to billing accounts or supply current market prices.

50.5.6 WORDS — remember these

1. **Allocated cost:** a share assigned under a rule — a reporting quantity based on an explicit apportionment policy. **Marginal cost:** the change caused by an additional unit of work — distinct from

dividing the total bill by current volume. **Cost boundary:** which charges are counted — the stated inclusions and exclusions behind a reported amount.

50.6 Operational reviews

50.6.1 PLAIN — in simple words

1. Review a data service as an ongoing responsibility, not a machine purchased once. Workload, dependencies, retention and permissions change.
2. A useful review compares actual outcomes with objectives, checks forecasts against measurements and assigns concrete actions for gaps.
3. Alerts need an owner and a response. A dashboard containing hundreds of red boxes but no decision path is not a complete operating method.
4. Keep a record of what changed and what followed. An incident is a chance to improve the system's evidence and controls, not an excuse to replace uncertainty with a confident story.

50.6.2 PLAIN — a picture in your head

1. Mira checks the van's service record, delivery outcomes and next month's expected workload. She does not wait for the engine to fail before discovering that no replacement driver is available.
2. When a delivery goes wrong, she records the observed sequence and tests plausible explanations rather than blaming the nearest person automatically.
3. **Where the comparison breaks:** distributed incidents can have several interacting causes. One graph changing first does not automatically establish it as the root cause.

50.6.3 PLAIN — a worked example

1. A synthetic weekly review contains: request success under its metric definition, p50/p95/p99 latency with sample counts, oldest queued job age, restore rehearsal age, growth versus forecast and unresolved lifecycle tasks.
2. A storage forecast says the intervention threshold is 25 days away. Procurement or migration takes 14 days and the team reserves 7 days for uncertainty. That leaves only 4 days before the planned action should begin under these assumptions.
3. A backup exists, but the last verified restore is old and uses a different application schema. The review records a new rehearsal task; it does not treat backup presence as current recovery proof.
4. An alert for growing outbox age includes a runbook: inspect worker health and failures, check destination availability, preserve message identities and avoid blind replay with new keys.

50.6.4 PLAIN — what is really happening inside

1. A review closes the loop between measurements and decisions. Forecast errors change the next forecast; recurring incidents change tests, capacity or architecture.
2. Operational evidence should retain its context: revision, environment, workload and observation time. Comparing two dashboards without those details can confuse a workload change with an implementation improvement.
3. A runbook proposes bounded observations and safe actions. Destructive changes and broad privilege escalation need their own authority, not an implicit permission hidden in an alert.

50.6.5 TECHNICAL — the engineer's version

1. Tie each alert to an actionable condition and owner. Separate symptoms affecting users from diagnostic signals that help explain them; both are valuable but serve different roles. [S103]
2. Maintain explicit unknowns: untested failover, incomplete telemetry, obsolete restore evidence or unidentified growth sources. Assigning an owner makes an unknown manageable; renaming it green does not.
3. The capstone's operational worksheet uses original synthetic arithmetic. Its production extension requires measurements from the actual service rather than estimates inferred from user count alone.

50.6.6 WORDS — remember these

1. **Runbook:** a documented response path — observations, decisions, bounded actions and escalation conditions for an operational situation. **Forecast error:** the difference between prediction and observation — evidence used to update assumptions rather than hide uncertainty. **Operational owner:** the accountable responder — the person or team responsible for a signal, policy or unresolved task.

50.97 Practice and worked answers

1. **Question:** What is the nonconformance allowance at 99.5% over 20,000 eligible requests? **Answer:** 100 under the example's exact definition.
2. **Question:** Can all trace span durations simply be added? **Answer:** Not if spans overlap; the worked 240 ms decomposition explicitly uses disjoint intervals.
3. **Question:** How much payload do 12,000 daily records of 600 bytes create over 365 days? **Answer:** 2,628,000,000 bytes before physical and operational overhead.
4. **Question:** At 20 GB used, 2 GB/day growth and a 70 GB intervention threshold, how much time remains? **Answer:** 25 days under constant net growth.
5. **Question:** How quickly does a 600-job backlog drain with capacity 100/s and ongoing arrivals 90/s? **Answer:** In 60 seconds under the deterministic assumptions, not 6 seconds.
6. **Question:** Does $L=\lambda W$ estimate p99 latency? **Answer:** No. It relates suitable averages under steady-state conditions.
7. **Question:** What is the allocated average of 9,000 units over 300,000 completed operations? **Answer:** 0.03 units per operation within the stated cost boundary.
8. **Question:** Does an existing backup establish current recoverability? **Answer:** No. A relevant restore procedure must be tested and its outcome checked.

50.98 Common wrong ideas

1. **Wrong:** Low CPU proves healthy service. **Right:** User outcomes and other bottlenecks may disagree.
2. **Wrong:** Missing errors mean all requests succeeded. **Right:** Observation coverage and unknown outcomes must be checked.
3. **Wrong:** Logs, metrics and traces are interchangeable. **Right:** They preserve different kinds of evidence.
4. **Wrong:** Number of users determines capacity. **Right:** Workload, data size, concurrency and objectives determine the relevant demand.

5. **Wrong:** More retries always improve reliability. **Right:** Retries can amplify overload and duplicate effects.
6. **Wrong:** A cost per operation is its marginal cost. **Right:** Allocated averages and incremental resource costs differ.
7. **Wrong:** A forecast is a specification. **Right:** It is an assumption-dependent prediction that needs updating.
8. **Wrong:** A dashboard replaces an operating team. **Right:** Signals need ownership, interpretation and bounded response procedures.

50.99 Chapter summary in 20 lines

1. Start observability with the user's intended outcome.
2. Define success, failure and unknown states.
3. Keep indicator, objective and agreement distinct.
4. Preserve the eligibility denominator and observation boundary.
5. Use logs, metrics and traces for their different strengths.
6. Correlate operations without copying unnecessary secrets or personal data.
7. Account for sampling, delayed export and missing signals.
8. Avoid unbounded metric-label cardinality.
9. Forecast storage from rates, representation sizes and lifetimes.
10. Separate payload from indexes, logs, copies and temporary work.
11. Keep GB and GiB explicit.
12. Set intervention thresholds early enough to act.
13. Identify the actual bottleneck before buying resources.
14. Measure throughput and latency under a declared arrival process.
15. Bound retries and protect the service from overload amplification.
16. State the failure assumptions behind headroom.
17. Make cost inclusions and allocation rules explicit.
18. Distinguish average cost from marginal cost.
19. Turn forecast errors and incidents into revised actions and tests.
20. Give every important unresolved operational condition an owner.

51

CHAPTER

Capstone: From One Shop to a Multi-Branch Service

51.0 What this chapter gives you

1. We began with a person recording a sale. We can now name the contracts, representations, queries, transactions, copies and operational responsibilities that make a larger service understandable.
2. This capstone assembles a small executable teaching system rather than adding a new layer of unexplained technology. Its purpose is to make selected correctness claims observable, including refused operations and recovery after injected failures.
3. The original Mira's Corner fixture remains unchanged: O-1042 is 17,100 paise, O-1043 is 11,550, and four agreed lines represent six units and 28,650 paise. Those are agreed line amounts, not proof of payment settlement.
4. The capstone introduces separate synthetic identifiers: tenants T-A/T-B, branch B-1, products NOTE/PEN, requests R-1/R-2 and orders CAP-001/CAP-002. These are new test fixtures, not explanations of the earlier unresolved stock discrepancy.
5. The implementation is an isolated Python/SQLite learning lab. It does not include a production login, payment processor, carrier, web server, replicated database, physical-failure test or legal compliance determination.

51.1 Requirements and boundaries

51.1.1 PLAIN — in simple words

1. Before choosing machinery, define what the small service must do. It should accept a permitted order, record the agreed prices, reduce the selected stock and remember the result of a repeated request.
2. Related changes must succeed together or leave no accepted order. A partially written order with fully deducted stock is not the intended outcome.
3. Each tenant's work must remain separate through the trusted helper interface. The same request or order identifier may exist independently in two tenants.
4. The service also records an intention to create a local follow-up task. That intention must not disappear because an acknowledgement is lost, and repeated delivery must not create repeated local tasks.
5. Not every real shop requirement belongs in the miniature model. Exclusions are stated so readers do not mistake a useful teaching implementation for a finished commercial checkout.

51.1.2 PLAIN — a picture in your head

1. Mira writes an order, marks the shelf count and leaves a task for packing. The three records describe different responsibilities but belong to one accepted operation.
2. Dev repeating the same request after the answer was lost should recover the earlier result, not pretend another customer bought the same goods.
3. **Where the comparison breaks:** the lab's packing task is another database row. Creating that row does not physically pack a parcel, and it cannot make an uncontrolled external carrier perform an action exactly once.

51.1.3 PLAIN — a worked example

1. Initialise T-A/B-1 with ten notebooks and fifty pens. The catalogue prices are 7,550 and 2,000 paise respectively. The separate T-B fixture has its own records with the same local product codes.
2. Submit R-1 for T-A order CAP-001 with two NOTE and one PEN. Its accepted total is 2×7,550 + 1×2,000 = 17,100 paise. Remaining stock is eight notebooks and forty-nine pens in that branch.
3. Submit the identical R-1 again. The result remains CAP-001 at 17,100; stock remains eight and forty-nine; the outbox still contains one intention for that request.
4. Submit R-1 with three notebooks instead. That is a different command under the same request identity and is rejected as a conflict. The earlier result is not overwritten.
5. Submit a T-B target using a T-A-only principal. The helper rejects it before returning or changing protected T-B records. This does not test whether a real token was authenticated correctly; the principal is supplied by the trusted test harness.

51.1.4 PLAIN — what is really happening inside

1. Input validation establishes the command's shape. Authorisation establishes its permitted scope. A database transaction then binds the accepted result, stock change, request record and outbox intention.
2. A request identity connects retries to the same intended operation. The stored command representation permits comparison with later uses of that identity.
3. Database constraints reject selected impossible states. Application code handles workflow decisions the schema does not express. Tests inspect both the successful path and the places where protection is intentionally absent.

51.1.5 TECHNICAL — the engineer's version

1. The contract accepts bounded nonempty identifiers, one branch, a nonempty cart, unique product codes and strictly positive integer quantities. Python booleans are rejected as quantities even though bool is a subclass of int.
2. Prices are copied from the lab's catalogue inside the owned transaction. Request replay returns the persisted accepted result instead of repricing against a later catalogue. The example does not implement quote expiry, discounts or a separate price-approval workflow.
3. The helper requires an idle connection and owns BEGIN IMMEDIATE, COMMIT and ROLLBACK. SQLite transaction and driver behaviour are explained by their documentation; the selected workflow and limits are original. [S25] [S86]

51.1.6 WORDS — remember these

1. **Accepted operation:** a committed result under a declared contract — the business outcome represented by the transaction, not merely a received HTTP request. **Scope boundary:** where the guarantee applies — the trusted helper, tenant, database and supported operations included in the claim. **Exclusion:** a deliberately unimplemented responsibility — an explicit limit such as payment settlement or physical dispatch.

51.2 Schema and workflows

51.2.1 PLAIN — in simple words

1. Store each fact where its meaning is clear. Catalogue price describes what is offered now; an order-line price describes what was accepted for that order.
2. Stock belongs to a tenant, branch and product. An order belongs to a tenant and has lines referring to products in that same tenant.
3. Request records describe replay identity. Outbox records describe intended follow-up. Inbox records describe locally accepted deliveries. These are not three copies of an order total with identical purposes.
4. A separate schema for these responsibilities makes inconsistencies easier to state and test. It does not remove the need to decide which changes must share a transaction.

51.2.2 PLAIN — a picture in your head

1. Mira keeps a price board, an order ledger, a shelf register and a packing-task tray. The price board changes without rewriting yesterday's signed order.
2. A note in the tray refers to an order rather than asking the packer to reconstruct the order from today's prices.
3. **Where the comparison breaks:** database references can be enforced mechanically, while paper references depend on people. Neither a foreign key nor a paper number proves that the real goods moved.

51.2.3 PLAIN — a worked example

1. The lab schema uses these logical grains. A scoped key includes tenant wherever two tenants can legitimately reuse the local identity.

Relation	One row means	Important key or boundary
tenants	One synthetic organisation	tenant_id
catalogue	Current offer for one product	tenant_id, sku
stock	One branch's available quantity	tenant_id, branch_id, sku
orders	One accepted order	tenant_id, order_id
order_lines	One accepted product line	tenant_id, order_id, line_no
requests	One successful request result	tenant_id, request_id
outbox	One intended task notification	tenant_id, event_id
inbox	One locally processed delivery identity	tenant_id, event_id
tasks	One local follow-up task	tenant_id, event_id

2. CAP-001 has two order lines. Its total is derived from the accepted line quantities and prices, not recalculated from the current catalogue on every read.
3. Reprice NOTE in a separate test after acceptance. Reading CAP-001 still yields 17,100 paise. A genuinely new order can use the new catalogue price under the lab's contract.
4. The schema rejects a child whose scoped parent does not exist. Test that the child cannot borrow an order identity from another tenant by omitting part of the reference.

51.2.4 PLAIN — what is really happening inside

1. The accept_order helper validates and authorises, starts its transaction, checks request identity, resolves product prices, conditionally deducts stock and inserts the related records.
2. The stock UPDATE includes a sufficient-quantity predicate. A zero-row result is treated as a failure of the intended reservation, not as an accepted order with missing stock work.
3. The accepted prices are protected from routine helper edits because no supported helper operation rewrites order lines. That is not the same as database-level immutability against an unrestricted connection.
4. All writes use bound values. Parameterisation prevents values from being interpreted as SQL syntax; it does not establish whether the caller is allowed to make the request.

51.2.5 TECHNICAL — the engineer's version

1. SQLite STRICT tables, explicit NOT NULL requirements, CHECK predicates and composite foreign keys form the declarative part of the model. Foreign-key enforcement is enabled and verified on helper-created connections. [S60] [S59]
2. A transaction serialises this lab's write operation under SQLite's locking model. The result is not a proof that a corresponding PostgreSQL implementation behaves identically at every isolation level.
3. The implementation accepts one owned transaction at a time per connection and fails rather than silently committing a caller's unrelated pending work. This transaction-ownership boundary is part of its API contract.

51.2.6 WORDS — remember these

1. **Row grain:** what one row asserts — the identity and meaning that determine valid joins and constraints. **Agreed price snapshot:** the price stored at acceptance — an order fact distinct from the mutable catalogue. **Transaction ownership:** responsibility for completion — the component entitled to commit or roll back the particular unit of work.

51.3 Correctness under retries

51.3.1 PLAIN — in simple words

1. A retry is ambiguous unless it identifies the operation being repeated. The service needs to tell another delivery of the same command from a new command that happens to look similar.
2. Store the request identity with its successful result in the same transaction as the effect. Otherwise a failure can leave either an effect with no replay record or a replay record with no effect.
3. An outbox records follow-up intent atomically with the accepted order. Delivery can still be repeated, so the receiver needs its own duplicate policy.

4. Exactly one local database task is a narrower claim than exactly one email, payment or physical shipment. Keep that distinction when crossing external boundaries.

51.3.2 PLAIN — a picture in your head

1. Dev sends a numbered instruction and does not hear back. He sends the same number again. Mira checks the completed-instruction register and returns the stored answer.
2. Mira's packing notification can also be delivered twice. The packer checks that notification's number before creating a second task.
3. **Where the comparison breaks:** the two registers may live in different databases and commit independently. A message acknowledgement cannot magically make two independent commits one indivisible event.

51.3.3 PLAIN — a worked example

1. Run accept_order for R-1 and deliberately ignore the returned result. The committed state still contains the order, stock reduction, request result and outbox intention.
2. Retry R-1 with the same canonical command. The helper returns the stored result without a second deduction. Change its cart under the same key and the comparison rejects the conflict.
3. Deliver the outbox event to the local consumer. The consumer commits an inbox identity and one task together. Now simulate a lost acknowledgement before the producer records delivery completion.
4. Redeliver that event. The consumer sees the already-recorded identical event and does not create another task. The producer can then record a successful acknowledgement. The example observes two deliveries but one local task.
5. Inject an exception after task insertion but before its consumer transaction commits. Both task and inbox insert roll back. A later delivery can therefore create the task once instead of being falsely suppressed by an inbox-only residue.

51.3.4 PLAIN — what is really happening inside

1. Idempotency requires an identity scope, a payload comparison, an effect boundary and a retention rule. A unique request key alone does not explain what happens when the payload changes or the old record expires.
2. The lab's command representation has sorted unique product lines and explicit branch/order identifiers. It compares canonical text, not only a hash, so digest equality is not treated as proof that arbitrary payloads are equal.
3. Producer completion and consumer effect are distinct observations. A lost acknowledgement leaves the producer uncertain even though the consumer may already have committed.
4. Retiring replay/inbox records would change the protection window. The exercise does not silently garbage-collect them and continue claiming permanent duplicate suppression.

51.3.5 TECHNICAL — the engineer's version

1. Uniqueness and related effect writes share an owned transaction. Duplicate detection is a database-backed protocol, not a process-local set that disappears when the application restarts.
2. The consumer stores the event payload with its identity and checks conflicting redelivery. An identity reused for a different task is not accepted as a harmless duplicate.

3. The demonstration uses local SQLite transactions and synthetic delivery calls. It does not exercise a broker, network partition or external provider's idempotency contract. Chapters 26 and 39 explain why those boundaries need their own design.

■ 51.3.6 WORDS — remember these

1. **Idempotency scope:** where a replay identity is meaningful — the tenant, operation and retention boundary governing duplicate handling. **Outbox:** atomically recorded delivery intent — a local record that survives with the business change but does not guarantee external completion. **Inbox:** the consumer's delivery record — an identity committed with the local effect to support repeat handling. **Acknowledgement loss:** the result exists but the sender did not learn it — a source of uncertainty rather than proof that the action failed.

51.4 Reporting and reconciliation

■ 51.4.1 PLAIN — in simple words

1. A useful report states what its rows and totals mean. The same data can answer questions about orders, lines, units or agreed money, but those are not interchangeable measures.
2. Keep the original teaching fixture separate from newly accepted capstone orders. Combining them without labels changes the population and makes earlier totals appear wrong.
3. Reconcile at the level where mistakes can occur: identities, quantities, prices, stock changes and task counts. One matching grand total cannot detect every error.
4. Report generation should preserve authorisation scope. A correct sum over the wrong tenant is still the wrong response.

■ 51.4.2 PLAIN — a picture in your head

1. Mira compares the order ledger, shelf changes and packing tasks at closing time. Each record answers a different part of what happened.
2. Two wrong prices can cancel in the total. Checking the individual lines reveals the defect the headline number hides.
3. **Where the comparison breaks:** a database reconciliation compares stored claims, not the physical shelf itself. A physical count remains an independent observation that can disagree for several reasons.

■ 51.4.3 PLAIN — a worked example

1. Recreate the earlier canonical fixture using its unchanged lab. Assert two orders, four lines, six units and 28,650 paise. The expected-stock-10/observed-stock-9 example remains unresolved.
2. In a separate capstone database, accept CAP-001 only. Assert its two lines, three units, 17,100 paise and one outbox event. Do not add this deliberately similar example to the canonical total.
3. After two deliveries of the same capstone event, assert one inbox row and one task for that scoped identity. Count delivery attempts separately from completed local effects.
4. If an independent T-B order has the same local order ID, a T-A report must still include only T-A's authorised records. Test the filter using distinguishable values, not two identical fixtures that would hide a mix-up.

51.4.4 PLAIN — what is really happening inside

1. A query produces a new relation whose grain follows its joins and grouping. Joining one order to multiple lines and multiple task records can multiply contributions unless the query controls each relationship.
2. Stored totals can be checked against line sums; stock changes can be compared with accepted reservations; task identities can be compared with recorded outbox events. Each reconciliation has a time and scope boundary.
3. External outcomes remain separate. An order total is not a payment settlement, and a task row is not a dispatched parcel. The schema would need additional evidence-bearing records for those claims.

51.4.5 TECHNICAL — the engineer's version

1. The companion tests use explicit ORDER BY where result ordering matters and compare complete keys and values. SQL output order is not inferred from insertion order.
2. Keep aggregates at their intended grain, using preaggregation or EXISTS where appropriate. Chapter 13's fan-out counterexample remains a reminder that syntactically valid SQL can answer the wrong question. [S58] [S80]
3. A reconciliation report records its definition and exclusions. It does not quietly turn a sum of agreed line amounts into revenue recognition, tax calculation or cash accounting.

51.4.6 WORDS — remember these

1. **Reconciliation population:** the records intended for comparison — a defined set that prevents mixing separate fixtures or tenants. **Effect count:** how many accepted changes occurred — distinct from how many attempts or deliveries were observed. **External outcome:** an event beyond the local transaction — payment, delivery or another result needing separate evidence.

51.5 Recovery and access tests

51.5.1 PLAIN — in simple words

1. Test interrupted work, not only successful work. A failure halfway through a multi-step change should leave a state the contract can explain.
2. Test recovery into a fresh destination and compare meaningful records. The presence of a backup file is not a successful restore result.
3. Test forbidden scope before and after errors. A rollback should not leave reused request context pointing at another tenant.
4. Name the failure being simulated. A raised Python exception is not a power cut, and an in-memory restore is not an off-site disaster-recovery exercise.

51.5.2 PLAIN — a picture in your head

1. Mira rehearses an interrupted order by stopping after writing one page and checking that the temporary work is discarded consistently.
2. She rehearses recovery by opening a copied ledger at a different desk and reading actual entries. Looking at the closed recovery box is not enough.

3. **Where the comparison breaks:** software failures occur at several layers. Exception handling exercises the application path; physical durability depends on storage, operating-system and device promises not exercised by that same test.

51.5.3 PLAIN — a worked example

1. Inject an exception after stock deduction and before accepted-order completion. Assert that stock, orders, lines, request record and outbox return to the pre-operation state under the owned transaction.
2. Retry after removing the injected failure. Assert a single accepted result and the expected remaining stock. A failed attempt did not consume the successful request key permanently in this contract.
3. Use SQLite's backup API to copy the synthetic database into a fresh destination. Re-enable connection-level foreign-key enforcement as required and compare scoped orders, line values, stock and replay records.
4. Run integrity and foreign-key checks for their distinct purposes. An integrity result is not a substitute for checking reference violations or business totals.
5. Attempt cross-tenant reads/writes and a forbidden action. Assert that outputs and stored state respect the trusted-helper contract. Then explicitly note that unrestricted SQL access lies outside that contract.

51.5.4 PLAIN — what is really happening inside

1. Rollback restores the transaction's database changes. It cannot undo an external side effect already performed, which is why the lab records intent instead of dispatching an email inside the transaction.
2. A backup API coordinates copying according to the engine's rules. The copied database still needs validation and appropriate application configuration before it is used.
3. The test harness supplies known state and forced interruption points. It checks the resulting state against an expected contract rather than inferring success from an exception name alone.

51.5.5 TECHNICAL — the engineer's version

1. Python's sqlite3 API exposes backup and explicit transaction management; SQLite documents the distinct integrity and foreign-key checks. The final test record identifies the actual Python/SQLite runtime used. [S55] [S59] [S86]
2. The earlier two-connection temporary-file locking exercise demonstrates one bounded real SQLite writer-contention scenario. The capstone's deterministic models and in-memory tests do not expand that into a multi-host concurrency claim.
3. Production evidence would additionally need genuine authentication/role integration, representative concurrent workloads, restore objectives, failure-domain tests, secret handling and lifecycle verification across actual copies. Those are a deployment programme, not an unreported result of this lab.

51.5.6 WORDS — remember these

1. **Fault injection:** deliberately interrupting a known path — a test technique whose simulated failure must be named precisely. **Restore rehearsal:** recovering and checking a separate destination — evidence about a specific recovery procedure and input. **Integrity check:** inspection of de-

clared structural properties — one layer of validation, distinct from complete business or security correctness.

51.6 Review and release evidence

51.6.1 PLAIN — in simple words

1. A finished teaching package should let another reader see which examples were implemented, which tests ran and which discussions remain conceptual.
2. Keep manuscript, code and evidence aligned. A paragraph claiming a particular lab exists is misleading if the download contains no such exercise.
3. Test counts measure cases executed, not overall truth. A test can correctly demonstrate missing protection, and a large suite can still omit an important requirement.
4. A public learning book is not a production-service certificate. Readers should be able to learn from the evidence without inheriting an unsupported guarantee.

51.6.2 PLAIN — a picture in your head

1. Mira hands Dev a recipe, the ingredients actually used and the notes from a trial batch. He can distinguish what was cooked from what was only proposed.
2. If the trial used one small oven, the notes should not claim that a factory production line was tested.
3. **Where the comparison breaks:** reproducibility also depends on software versions and exact input bytes. A familiar filename alone may not identify the same candidate.

51.6.3 PLAIN — a worked example

1. Extract the companion code into a fresh folder and read its README before execution. Run the supplied unittest discovery command with the supported Python version. The output identifies successes, failures and any explicit skips.
2. Read the lab inventory. The foundations modules cover Chapters 1–12; advanced_lab contains bounded models and query/representation exercises for later topics; capstone_lab assembles the scoped local order workflow and Part H calculations.
3. Compare the included QA report with the package manifest. The report identifies the actual test run and generated artefacts; the manifest supports checking whether a later copy has changed.
4. A skipped optional capability is not a passed test. An illustrative PostgreSQL statement remains documented rather than executed unless the evidence explicitly identifies a real PostgreSQL run.
5. The package's website is the book reader, not a deployed checkout application. Publishing the reader makes educational text and downloads available; it does not install the capstone as a commercial service.

51.6.4 PLAIN — what is really happening inside

1. A test suite supplies executable assertions over known inputs. A build pipeline converts one manuscript into PDF, volume and browser representations. Structural checks then compare chapter coverage, identifiers and resource targets.
2. Visual inspection catches a different class of problem: a clipped formula, unreadable table or misplaced heading may survive a text-only check.

3. The release record should distinguish automatic checks, visual review, source research and any independent review. Do not invent a reviewer or declare that one authoring pass is independent of itself.

51.6.5 TECHNICAL — the engineer's version

1. Python unittest's result model distinguishes failures, errors and skips. The supplied runner records the runtime and actual counts rather than hardcoding a flattering number in the manuscript. [S201]
2. Stable chapter and section identifiers support cross-edition reference even when a standalone volume has different page numbers. Each volume's contents refer to that volume; its chapter numbers match the complete edition.
3. A package manifest is an integrity aid, not authentication of its publisher by itself. A trustworthy distribution channel and an independently obtained expected digest are needed for stronger origin claims.
4. The practical conclusion is not "never trust a system." It is to define the required claim, inspect the mechanism, run a suitable observation and keep the conclusion within the evidence.

51.6.6 WORDS — remember these

1. **Executable assertion:** a machine-checkable expectation — a predicate comparing an observed result with a declared outcome. **Release manifest:** an inventory of delivered bytes — filenames, sizes and digests that support integrity checks. **Evidence boundary:** the limit of a justified conclusion — the tested scope, assumptions and unresolved properties attached to a result.

51.97 Practice and worked answers

1. **Question:** What stock remains after CAP-001 and its identical retry? **Answer:** Eight notebooks and forty-nine pens in T-A/B-1; the retry returns the accepted result without a second deduction.
2. **Question:** What happens if R-1 returns with a different quantity? **Answer:** A conflicting command under the same scoped identity is refused; the old result is not overwritten.
3. **Question:** Does repricing NOTE change an accepted CAP-001? **Answer:** No. Its line prices remain the accepted snapshot.
4. **Question:** What does two deliveries and one task establish? **Answer:** The tested local inbox/task protocol suppresses an identical repeated delivery. It does not establish exactly-once external dispatch.
5. **Question:** Why compare both tenants' state after a denied write? **Answer:** A denied response alone does not prove that no protected state changed.
6. **Question:** Does this capstone explain the original stock mismatch? **Answer:** No. Its new fixtures are separate; the original expected-ten/observed-nine discrepancy still has no asserted cause.
7. **Question:** What does the backup test not establish? **Answer:** Off-site recovery, device power-loss durability, PostgreSQL recovery and production objectives remain outside its scope.
8. **Question:** Is the website reader a production checkout deployment? **Answer:** No. It publishes educational content and downloads, not a live order-processing service.

51.98 Common wrong ideas

1. **Wrong:** A capstone must implement every commercial requirement. **Right:** A useful educational system has explicit included claims and exclusions.

2. **Wrong:** A request key alone creates idempotency. **Right:** Scope, payload comparison, atomic effects and retention also matter.
3. **Wrong:** The current catalogue is the historical agreement. **Right:** Accepted line prices are separate facts.
4. **Wrong:** An outbox guarantees exactly one external action. **Right:** It records intent; receiver and external-effect boundaries remain.
5. **Wrong:** A cross-tenant SELECT test proves all access paths are safe. **Right:** Exports, workers, mutations and privileged interfaces need their own coverage.
6. **Wrong:** Matching totals reconcile everything. **Right:** Compare scoped identities and values as well.
7. **Wrong:** Exception injection tests physical durability. **Right:** It exercises only the injected software path and transaction outcome.
8. **Wrong:** Many passing tests make the publication infallible. **Right:** Test coverage and factual/editorial review remain distinct forms of evidence.

51.99 Chapter summary in 20 lines

1. Begin the capstone with explicit requirements and exclusions.
2. Keep new fixtures separate from the original canonical records.
3. Establish principal and tenant scope before protected operations.
4. Validate bounded command shapes and integer quantities.
5. Capture agreed prices separately from mutable catalogue prices.
6. Give every table a clear row grain and scoped identity.
7. Own one transaction for each accepted local operation.
8. Keep stock, order, request result and outbox intent atomic.
9. Compare repeated payloads under the same request identity.
10. Recover a lost response without creating another order.
11. Commit consumer inbox and local task together.
12. Distinguish redelivery from repeated local effect.
13. Do not extend local deduplication to uncontrolled external actions.
14. Reconcile keys, quantities, prices and scoped totals.
15. Test refused operations and their stored effects.
16. Inject bounded failures and inspect rollback.
17. Restore into a fresh destination and check useful records.
18. Identify the actual runtime, assertions and skips.
19. Keep text, code, downloads and release records consistent.
20. Carry the evidence discipline from one shop to every larger system.

52

CHAPTER

The Reference Desk: Glossary and Diagnostic Guides

52.0 What this chapter gives you

1. This chapter is a place to return when a term, query or failure needs a precise next step. It is not a substitute for the explanations earlier in the book; it connects their vocabulary and methods to reusable reference material.
2. The complete A–Z glossary follows the chapter in the full edition and is available in the browser reader. Each entry points back to its teaching section. The sources register records document titles, versions, URLs and consultation dates.
3. Stable references such as 26.4 and 47.3 mean the same section across the complete PDF, volume editions and website. Printed page numbers may differ between editions, so chapter/section identifiers are the reliable cross-edition route.
4. The diagnostic paths below begin with observation and scope. They do not give blanket permission to run destructive commands, broaden access or modify an unfamiliar live database.

52.1 Plain-to-technical glossary

52.1.1 PLAIN — in simple words

1. A useful definition connects an everyday idea to a precise technical meaning. It also marks where that meaning changes with context or implementation.
2. Similar words do not always mean the same thing. A database, database engine, database server and database file refer to different parts of the system.
3. Begin with the plain meaning, then use the linked section to inspect assumptions and mechanisms. Memorising a term without its boundary can make a mistaken explanation sound more convincing.
4. The A–Z register collects the vocabulary blocks from the manuscript. Where a term appears in several contexts, the relevant definitions and locations remain visible rather than being silently merged into one misleading sentence.

52.1.2 PLAIN — a picture in your head

1. A city map tells you where a street is; it does not describe every room in every building. The glossary locates an idea, and the chapter supplies the detailed route.
2. Two streets can share a familiar name in different towns. In the same way, “consistency” can refer to several distinct properties unless the surrounding discussion names the model.

3. **Where the comparison breaks:** technical vocabulary is not governed by one universal naming authority. Standards, product documentation and informal usage can use related words differently.

52.1.3 PLAIN — a worked example

1. Look up transaction. The everyday meaning is a group of related changes treated as one controlled unit. The technical discussion adds commit, rollback, isolation, durability and the boundary around external effects.
2. Look up idempotency. The plain idea is that repeating the same intended operation need not produce another effect. Chapter 26 and the capstone add scoped identity, payload comparison, transaction placement and retention.
3. Look up consistency before using it in an argument. A business invariant, a replica freshness requirement and linearizability are different claims. A system can satisfy one and fail another.
4. A glossary answer becomes useful when it leads to the relevant question: Which changes? Which identity? Which observation order? Which failure? Which version of the product?

52.1.4 PLAIN — what is really happening inside

1. The glossary builder extracts named terms and their definitions from the WORDS blocks and associates them with chapter/section identifiers.
2. The website groups entries alphabetically and offers text filtering. Chapter search uses titles, section terms and chapter text rather than pretending a keyword match establishes technical correctness.
3. A repeated term can have several entries or locations. That is preferable to discarding context simply to make a smaller count look tidier.

52.1.5 TECHNICAL — the engineer's version

1. Use the source register when a definition is standard- or product-specific. The glossary's role is navigation; it does not supersede the cited specification.
2. Treat identifiers in a reference system as stable keys. A title can be edited while a chapter number and section identity preserve links across formats.
3. Generated vocabulary is checked for resolvable source locations. Mechanical extraction does not independently verify every definition; factual accuracy still depends on the manuscript and its cited evidence.

52.1.6 WORDS — remember these

1. **Glossary:** a vocabulary reference — definitions connected to their contexts and teaching locations. **Cross-reference:** a route to related material — a stable chapter, section or source identifier rather than a guessed page number. **Implementation boundary:** where a claim stops generalising — the product, version and configuration for which a detail is established.

52.2 SQL and data-type reference

52.2.1 PLAIN — in simple words

1. Read a query as a precise question about a chosen population. First understand what one input row means, then what joins, filters and grouping do to that population.

2. Choose a type by the value's meaning. Counts, money, names, dates, instants and missing values should not become interchangeable strings merely because strings are easy to store.
3. Use explicit ordering when order matters and bound values when inputs come from outside the SQL text.
4. A correct query still needs appropriate authority, transaction context and an honest interpretation of its result.

52.2.2 PLAIN — a picture in your head

1. Mira asks Dev to take particular folders, match related pages, discard ineligible entries, group the remaining facts and sort the answer.
2. Each instruction changes the question. Accidentally matching each order page to every packing slip can multiply the answer even when Dev adds the numbers correctly.
3. **Where the comparison breaks:** a database optimiser can execute a logically equivalent plan in a different physical order. The readable conceptual sequence is not a claim about the engine's exact execution schedule.

52.2.3 PLAIN — a worked example

1. The canonical agreed-line calculation is quantity multiplied by agreed unit price in paise, summed over the intended four lines. Its result is 28,650 paise. It is not a sum of today's catalogue prices.
2. This small reference table connects common intentions to important cautions. It is not a complete SQL grammar.

Intention	Common SQL form	Check before trusting the result
Choose values	SELECT expressions FROM relation	What does one input row mean?
Select a population	WHERE predicate	How are NULL and excluded rows treated?
Connect facts	JOIN … ON complete_key	Can either side multiply the other?
Preserve unmatched rows	LEFT JOIN	Does a later WHERE remove them again?
Summarise groups	GROUP BY, SUM, COUNT	What is the output grain and denominator?
Retain detail with a calculation	OVER (…)	What partition, ordering and frame apply?
Specify order	ORDER BY, explicit tie-breaker	Are ties and missing values defined?
Change selected rows	UPDATE … WHERE …	Is the complete key and expected version included?
Bind input values	Driver parameters	Values are not SQL identifiers or permissions
Complete a unit of work	COMMIT or ROLLBACK	Which component owns the transaction?

3. `COUNT(*)` and `COUNT(nullable_column)` can differ because they count different things. Preserve the denominator required by the question rather than choosing whichever expression gives the more attractive number.
4. `SUM(DISTINCT amount)` does not generally repair a multiplied join. Two legitimate lines can have equal amounts and both should count. Fix the grain and relationship, not the coincidence of values.

52.2.4 PLAIN — what is really happening inside

1. Parsing establishes a valid statement; name/type resolution connects it to objects and operations; planning chooses access paths; execution produces the result under the database's rules.
2. Representation decisions affect correctness. An integer amount needs a unit and range; a decimal needs scale and rounding policy; a timestamp needs a meaning and time basis.
3. A constraint can reject selected invalid values. It cannot determine whether a supplied measurement, currency or identity matches the outside world without a trusted observation or policy.

52.2.5 TECHNICAL — the engineer's version

1. PostgreSQL and SQLite are not interchangeable implementations of every type, function or transaction behaviour. Use the product/version source attached to the example, especially around NULL, typing, locking and schema changes. [S58] [S80] [S86]
2. A compact representation checklist follows. It describes design questions rather than claiming one universally best physical type.

Value meaning	Representation decision	Required surrounding contract
Count	Integer with a suitable range	Unit, allowed sign, overflow behaviour
Money	Exact decimal or integer minor units	Currency, scale, rounding and agreement time
Identifier	Text, integer or structured key	Scope, stability, generation and reuse policy
Name	Unicode text	Comparison, normalisation and display rules
Instant	Aware time representation	Time scale, zone/offset interpretation and precision
Local date	Calendar date	Calendar and policy context; not an instant by itself
Unknown value	NULL or explicit status model	Unknown versus not applicable versus invalid
Measured quantity	Value plus unit and context	Method, precision, uncertainty and observation time

3. Keep SQL syntax examples and executable files aligned. A code block labelled as a sketch should not be treated as a complete application merely because it resembles runnable SQL.

52.2.6 WORDS — remember these

1. **Query grain:** what one result row means — the level created by joins, projection and grouping. **Tie-breaker:** an additional ordering key — a rule making otherwise tied results deterministic where required. **Representation contract:** meaning attached to stored values — units, range, missingness and interpretation beyond the physical type.

52.3 Diagnostic decision guides

52.3.1 PLAIN — in simple words

1. A symptom is the starting observation, not the conclusion. "The total is wrong" and "the database is broken" are different statements.

2. Choose a small next observation that can distinguish plausible explanations. Broad changes made before preserving evidence can destroy the information needed to understand the failure.
3. Keep read-only inspection separate from repair. A diagnostic query does not authorise deleting the rows it happens to find.
4. Record the database, tenant, time, revision and exact query or operation involved. A result from the wrong environment can be perfectly accurate and irrelevant.

52.3.2 PLAIN — a picture in your head

1. A shop light does not turn on. Mira first checks whether one bulb, one room or the whole building is affected. She does not immediately replace every wire.
2. The scope of the symptom determines the next useful check. A systematic route reduces guessing and avoids damaging evidence.
3. **Where the comparison breaks:** data-system failures can be partial and concurrent. The same query can legitimately observe different states under different snapshots or replicas.

52.3.3 PLAIN — a worked example

1. Use this wrong-total route on a disposable copy or through authorised read-only inspection:

```
Unexpected total
  -> confirm metric definition, unit and population
  -> confirm tenant, time basis and source version
  -> compare contributing primary keys
  -> inspect join multiplicity before aggregation
  -> inspect NULL, filters and unmatched rows
  -> compare agreed values with mutable catalogue values
  -> reconcile raw contributions and declared exclusions
  -> preserve evidence; propose a bounded correction
```

2. Use this slow-query route without assuming an index is the answer:

```
Slow request
  -> locate where the time is spent
  -> separate queueing, locks, database work and network
  -> capture query, parameters, plan and observation scope
  -> compare estimates with observed work where safe
  -> check workload, selectivity, memory and storage waits
  -> test one justified change on representative data
  -> compare equal results and full latency distributions
```

3. Use this timeout route before retrying a write:

```
Caller did not receive a result
  -> preserve the original request identity
  -> ask whether the operation's status can be queried
  -> replay only under its documented same-command policy
  -> distinguish committed, rejected and still-unknown outcomes
  -> do not create a new operation merely to make uncertainty disappear
```

4. Each route chooses an observation. None says that a symptom alone proves data corruption, failed commit or insufficient hardware.

52.3.4 PLAIN — what is really happening inside

1. Diagnosis compares hypotheses with observations. A good next check eliminates or narrows explanations while changing as little relevant state as possible.

2. Evidence can be time-sensitive. Query plans, lock waits and replica lag can change; record their observation window rather than presenting them as permanent properties.
3. A repair should have its own preconditions, scope, validation and recovery plan. The urgency of a symptom does not remove those responsibilities.

52.3.5 TECHNICAL — the engineer's version

1. For stale reads, check the requested consistency contract, routing, snapshot and observed replication position. A follower being behind is not automatically corruption.
2. For missing records, check authorisation filters, tenant scope, transaction outcome, lifecycle action and environment before attempting restoration. A correctly hidden row and a lost row require different responses.
3. For suspected corruption, preserve the original evidence and use engine-specific checks on an appropriate copy. Structural checks, checksums and business reconciliation establish different properties; consult Chapters 31–32 before repair.

52.3.6 WORDS — remember these

1. **Symptom:** what was observed — a result needing explanation, not an already-proven cause. **Hypothesis:** a candidate explanation — a claim that should be distinguished from alternatives through suitable evidence. **Bounded observation:** a deliberately limited check — a query or measurement with named scope and minimal unintended change.

52.4 Design-review questions

52.4.1 PLAIN — in simple words

1. A design review should make hidden assumptions visible. Ask what each record means, who may act and what happens when requests overlap or answers are lost.
2. Review the whole path, not only the database diagram. Inputs, workers, caches, exports and recovery procedures can invalidate an otherwise careful schema.
3. Require a counterexample to each important guarantee. Asking "How could this fail?" helps turn an attractive description into a testable contract.
4. A reviewer can identify evidence gaps without claiming the design is impossible. The next step is a targeted test or clarification, not an invented green verdict.

52.4.2 PLAIN — a picture in your head

1. Before building a new storeroom, Mira asks what it must hold, who gets a key, how deliveries enter and what happens if the only door is blocked.
2. A beautiful floor plan is incomplete until those operating questions have answers.
3. **Where the comparison breaks:** software can change very quickly. A review of one version should not automatically cover a later edit to a constraint, permission or retry policy.

52.4.3 PLAIN — a worked example

1. Apply these questions to the capstone before extending it:

Area	Review question	Example evidence
Meaning	What does one row assert?	Grain, units, provenance and exclusions
Identity	Where is each key unique?	Scoped key and reuse counterexamples
Authority	Who may perform each action?	Allow/deny matrix and effective-role checks
Change	Which facts must commit together?	Interrupted-operation state comparisons
Concurrency	Which overlapping schedule would break the rule?	A failing schedule and a tested protocol
Replay	What is the same intended request?	Identity, payload comparison and retention
Copies	Which reads may be stale?	Routing and declared consistency model
Recovery	What can actually be restored?	Isolated restore with checked records
Lifecycle	Which copies remain after deletion?	Inventory and explicit residual scope
Operations	What objective and capacity assumptions apply?	Workload, measurements and response owners

2. A new payment integration changes the external-effect boundary. The local task's once-only test does not become proof of once-only charging.
3. A new tenant export changes access coverage. The original detail-query test is not enough; the export's scope and content need separate tests.
4. A new retention policy changes replay guarantees if it removes request identities. The design must state what happens after that protection window ends.

52.4.4 PLAIN — what is really happening inside

1. Reviews compare the proposed mechanisms with requirements and counterexamples. They expose which promises rely on unverified assumptions.
2. A small set of well-chosen adversarial cases can be more informative than a large set of near-identical successes. The goal is evidence coverage, not a decorative test count.
3. Review findings should retain severity, scope, proposed response and verification status. Author approval of a plan does not by itself close an implementation defect.

52.4.5 TECHNICAL — the engineer's version

1. Separate safety properties, progress properties and operational objectives. "Never create two accepted orders for one scoped request" differs from "eventually deliver every valid notification" and from "respond within 200 ms."
2. Identify failure assumptions: process crash, device loss, network partition, byzantine behaviour or accidental operator changes. A mechanism designed for one fault model should not inherit another model's guarantee silently.
3. The review worksheet is an original reference aid. It records questions, not an automatic approval algorithm for every possible data system.

52.4.6 WORDS — remember these

1. **Safety property:** a prohibited outcome never occurs — a correctness claim such as no duplicate accepted effect under the stated model. **Progress property:** required work can eventually advance — a liveness claim dependent on assumptions about failures and resources. **Fault model:** the failures considered — the boundary within which a mechanism's reasoning and tests are intended to apply.

52.5 Test and recovery templates

52.5.1 PLAIN — in simple words

1. A test record should let another person repeat the important steps and understand the result. "It worked" omits the input, environment, operation and expected outcome.
2. A recovery record needs a specific source, destination and time boundary. Restoring the wrong backup correctly is still the wrong recovery.
3. Failed and skipped steps should remain visible. They are not automatically covered by the successes around them.
4. Templates organise evidence; filling the boxes does not manufacture observations that were never made.

52.5.2 PLAIN — a picture in your head

1. A recipe notebook records ingredients, oven setting and actual result. Leaving out the temperature makes a successful trial hard to repeat.
2. A recovery checklist records which box was opened and which pages were compared. Checking only that a box exists cannot fill the result column.
3. **Where the comparison breaks:** execution environments can differ invisibly through library versions, configuration and concurrent work. Reproducibility requires more than copying visible commands.

52.5.3 PLAIN — a worked example

1. Use this original bounded-test template for a new exercise:

```
Claim:
Candidate identity:
Environment and versions:
Input identities and scope:
Assumptions and excluded failures:
Operation and authority:
Expected output and state:
Actual output and state:
Negative or interruption control:
Result: pass / fail / error / skipped / unresolved
Evidence location:
What this result does not establish:
```

2. Use this recovery template before treating a backup as useful evidence:

```
Recovery purpose and authorised scope:
Backup identity, time and dependencies:
Required keys/configuration, without embedding secrets:
Fresh isolated destination:
Target recovery point and declared objectives:
Procedure and stop conditions:
```

```
Structural checks:
Key/value and application reconciliation:
Post-backup lifecycle decisions to reapply:
Access-boundary checks:
Actual duration and unresolved loss interval:
Approval required before serving restored data:
```

3. For a replay test, record both attempts with the same scoped identity and compare effects after each. A response that merely repeats the same text is not enough if stock changed twice.
4. For a denied-operation test, inspect state and returned information. The error message is only one observable result.

52.5.4 PLAIN — what is really happening inside

1. A test harness constructs controlled conditions, invokes the candidate and checks assertions. A recovery rehearsal applies a procedure to a selected historical state and validates the resulting destination.
2. Both need a reproducible identity for the tested material. Digests support byte comparison, while version and environment records explain how those bytes were used.
3. A result can be narrow and still valuable. The discipline is to keep the useful finding without converting it into a wider unsupported guarantee.

52.5.5 TECHNICAL — the engineer's version

1. Use machine-readable reports alongside human interpretation. Record failures, errors and skips distinctly rather than treating every completed process as a successful test. [S201]
2. Restore evidence should compare business-level identities and values in addition to engine checks. A structurally valid database can still contain the wrong dataset.
3. Keep credentials out of the evidence bundle. Record how authority was established without copying the secret material needed to exercise it.

52.5.6 WORDS — remember these

1. **Test fixture:** the controlled input state — synthetic or appropriately authorised data used for a specified check. **Reproducibility record:** enough context to repeat a result — candidate, inputs, environment, operations and observations. **Skipped check:** a check not executed under current conditions — neither a pass nor proof of failure of the underlying feature.

52.6 Source and version register

52.6.1 PLAIN — in simple words

1. Technical behaviour changes across products and versions. The reference beside a claim tells you where its documented meaning came from and which scope was consulted.
2. A standard, a vendor implementation and a teaching model are different sources of authority. An original shop example can demonstrate arithmetic without becoming an industry standard.
3. Read sources when applying a mechanism to real work. The book provides a path to them, not a promise that every future release preserves every detail.
4. The final source register combines the earlier source identifiers with the added references. Old consultation dates remain historical; they are not relabelled as newly rechecked merely because a new PDF was built.

52.6.2 PLAIN — a picture in your head

1. Mira keeps the instruction manual for the exact machine in the shop. A manual for another model may look similar but describe different buttons and limits.
2. A handwritten experiment note is useful too, provided it says what was tried and does not pretend to be the manufacturer's specification.
3. **Where the comparison breaks:** online documentation can change at an unchanged address. A URL alone is weaker evidence than a versioned document plus a consultation date and recorded scope.

52.6.3 PLAIN — a worked example

1. A paragraph marked [S86] refers to Python's version-3.13 sqlite3 documentation. It does not establish which exact Python or SQLite build a reader's computer uses; the executable report records the actual authoring runtime separately.
2. A paragraph about PostgreSQL 17 row policies marked [S187] describes that documented feature and its exceptions. The local SQLite lab does not become a PostgreSQL test because the two appear in one chapter.
3. An invented queue with 120 arrivals/s and 100 completions/s is an original arithmetic model. Its result follows from its assumptions, not from a measurement of a hosting plan.
4. For an applied design, record the selected source version, relevant configuration, local experiment and remaining uncertainty together. No single reference establishes every layer of the final system.

52.6.4 PLAIN — what is really happening inside

1. The publication builder resolves source identifiers to entries containing author or organisation, title, URL, consultation date and a short description of supported scope.
2. Manuscript references are checked for missing identifiers. That mechanical check prevents dangling citations; it does not prove that every citation is sufficient or that every interpretation is correct.
3. Readers should recheck version-specific details before deployment. When a source changes, preserve the earlier statement's historical context rather than pretending it always described the new version.

52.6.5 TECHNICAL — the engineer's version

1. The final package includes canonical Markdown, structured chapter metadata, glossary/source registers, executable exercises and build/QA records. Derived formats use the same chapter numbering and headings.
2. The cited materials include official database and language documentation, standards, original research and author-hosted technical texts. The synthetic shop, diagrams, chosen protocols and worked arithmetic are original teaching material unless explicitly attributed otherwise.
3. No part of the source register claims independent technical certification of the book. Source support, local testing, visual review and publication approval remain distinct kinds of evidence.
4. The final lesson is the method used throughout: define meaning, name the boundary, inspect the mechanism, test the relevant case and state only what follows. That method is transferable beyond the particular tools in these pages.

52.6.6 WORDS — remember these

1. **Primary source:** the originating documentation or research — evidence closer to the specified behaviour than a second-hand summary. **Consultation date:** when a source was checked — a historical marker that does not turn a moving page into an immutable specification. **Version register:** the declared implementation/document context — a record distinguishing documented versions from the runtime actually tested.

52.97 Practice and worked answers

1. **Question:** Which reference survives a change in volume pagination? **Answer:** The stable chapter and section identifier, such as 47.3.
2. **Question:** What is the first check for an unexpected total? **Answer:** Confirm the metric definition, units, population and authorised scope before proposing a repair.
3. **Question:** Why is SUM(DISTINCT amount) not a general join repair? **Answer:** Equal legitimate values can be distinct contributions; the relationship grain must be corrected.
4. **Question:** What identity should be preserved after a write timeout? **Answer:** The original scoped request identity under its documented replay contract.
5. **Question:** What distinguishes a safety property from a latency target? **Answer:** Safety forbids a defined outcome; latency targets quantify service performance over a stated population and window.
6. **Question:** Does a skipped test count as passed? **Answer:** No. It records a check not executed under the current conditions.
7. **Question:** Does a valid citation ID establish independent accuracy review? **Answer:** No. It establishes a resolvable reference, not universal correctness or independence.
8. **Question:** What is the book's most reusable skill? **Answer:** Connecting a precisely defined claim to a mechanism and suitable bounded evidence, while preserving what remains unknown.

52.98 Common wrong ideas

1. **Wrong:** Knowing a term means knowing its guarantees. **Right:** Read the definition's context, assumptions and implementation boundary.
2. **Wrong:** Page numbers are stable across every edition. **Right:** Chapter and section identities are the stable cross-edition route.
3. **Wrong:** A symptom is already a diagnosis. **Right:** Use observations to distinguish competing explanations.
4. **Wrong:** A faster query is necessarily the same question. **Right:** Compare results, population and semantics as well as time.
5. **Wrong:** A complete template supplies missing evidence. **Right:** Fields must contain actual observations or explicit unresolved status.
6. **Wrong:** A cited product's example tests another product. **Right:** Documentation and runtime scope must remain distinct.
7. **Wrong:** A file hash proves its publisher's identity by itself. **Right:** Integrity comparison needs a trusted expected value or distribution context.
8. **Wrong:** A reference book can remove every uncertainty. **Right:** It can teach readers to identify, bound and investigate uncertainty accurately.

52.99 Chapter summary in 20 lines

1. Use the reference desk to locate precise ideas and next steps.
2. Connect plain meanings to technical definitions and their limits.
3. Prefer stable chapter and section identifiers across editions.
4. Preserve context when one term has several meanings.
5. Read SQL as a question about a declared population.
6. Check grain before joins and aggregation.
7. Choose representations by meaning, units and range.
8. Treat NULL and missing status deliberately.
9. Specify ordering and tie-breakers where needed.
10. Preserve request identity after an uncertain write outcome.
11. Diagnose from symptoms rather than assuming a cause.
12. Choose bounded observations before repair.
13. Separate safety, progress and performance requirements.
14. Ask for counterexamples to important guarantees.
15. Record exact candidates, inputs and environments.
16. Distinguish pass, fail, error, skip and unresolved states.
17. Restore into isolation and compare useful records.
18. Keep primary sources and version context accessible.
19. Do not turn local checks into unearned universal certification.
20. Meaning, mechanisms and evidence belong together in every data system.

Companion Labs

The supplied practice package uses Python's standard library and fresh synthetic data. It does not connect to your website, payment provider, production database or cloud account. Read this guide before running code.

Start in an isolated folder

1. Extract `practice-code.zip` into a new folder. Keep its Python files together because the later suites import the earlier teaching modules.
2. Use Python 3.11 or newer with SQLite 3.37.0 or newer. This minimum covers the syntax and STRICT tables used here; it is not a recommendation to operate an old runtime in production. The accompanying `test-results.json` identifies the actual authoring environment.
3. Open a terminal in that extracted folder and run the test command below. It discovers the six test modules. Some cases deliberately confirm a missing safeguard or a failing design; their success means the stated counterexample was observed.
4. The examples include in-memory databases and one earlier bounded temporary-file locking demonstration. Temporary resources are cleaned up by their tests. No personal database or user records are required.

```
python3 -m unittest discover -s . -p 'test_*.py' -v
python3 advanced_lab.py
python3 capstone_lab.py
```

The last two commands print a small local performance observation and a capstone transaction/replay/restore trace. Timing results vary by machine and workload. They do not promise that a particular index always wins.

What is implemented

Module	Chapters and exercises	Boundary
`foundations_lab.py`	1–3: canonical orders, number/text/time representations and NULL behavior	Pure functions and a fresh SQLite fixture.
`history_lab.py`	4–6: measurements, scoped identity, duplicates and reconstructed history	Small explicit state machines; not a production event store.
`data_bridge_lab.py`	7–9 and 13: CSV/JSON import, the shop tables, relationships, totals, backup and a lock example	Bounded input profile, synthetic fixtures and local SQLite.
`schema_sql_lab.py`	10–12: schema rules, SQL, parameters, rollback and deliberately missing protections	Product-specific SQLite observations; PostgreSQL is documented, not executed.

Module	Chapters and exercises	Boundary
`advanced_lab.py`	14–46: query measurements, a leaf split, hashes, wait graphs, revisions, LRU, recovery prefixes, tombstones, replicas, fencing, majorities, protocol states, caches, windows, manifests, encodings, search, vectors and uncertainty	Local models and isolated observations, not full storage engines or network protocols.
`capstone_lab.py`	47–52: permission fixtures, tenant-scoped orders, outbox/inbox tasks, retries, local restore, retention status, resumable backfill and capacity/cost arithmetic	Trusted Principal objects, in-memory SQLite and bounded functions; not authentication, a public API or a deployed service.

The advanced models

The B-tree exercise implements ordered separator selection and a single overflowing-leaf split. It does not implement a complete balanced persistent B-tree. The log example replays committed toy transactions from a supplied durable prefix; it does not parse an engine's actual WAL or simulate torn storage sectors. The compaction example can discard tombstones only under its stated complete-history, latest-only assumptions.

The replica model tracks contiguous applied positions. The fencing model has the resource check a monotonically increasing authority token. The consensus exercises enumerate majorities, compare last-term/last-index pairs and test the current-term direct-commit condition. These are selected rules, not a full Raft implementation or proof of progress under a real network.

The transaction participant and compensation classes show explicit state transitions and repeated-decision handling. The cache and manifest classes model generation/version checks in one process; they do not implement shared-memory atomicity or a distributed metadata service.

The stream examples distinguish fixed windows, sliding windows, session overlap and late corrections. The columnar exercise packs signed 64-bit integers and uses zlib to compare representations. It is not a Parquet writer or a disk-I/O benchmark. Text search uses a tiny tokenized corpus, with an additional SQLite FTS5 check when that feature is present. Vector examples calculate exact distances and filter by tenant before selecting results; they do not train an embedding model or implement a production approximate-nearest-neighbor index.

The Wilson interval, missing-status bounds and nearest-rank percentiles check arithmetic under explicit assumptions. They do not establish representative sampling, independence or causal identification in real business records.

The capstone boundary

The capstone accepts a canonicalized cart, reads current catalogue prices during acceptance, conditionally reduces stock and stores the agreed order, request identity and outbox event in one owned transaction. An identical scoped replay returns the committed result without repricing or reducing stock again. A conflicting payload using the same scoped request identity is rejected.

The consumer stores one local task and its inbox identity atomically. An injected lost acknowledgement occurs after that local consumer commit; redelivery does not create another task. This is not proof that an external courier, email provider or payment processor performs its work exactly once.

The local restore uses SQLite's backup API, enables foreign-key checking on the target, checks database integrity and references, compares logical records and continues a synthetic operation. It does not test power loss, cloud loss, independent failure domains or an off-site recovery objective.

The Principal objects are trusted fixtures constructed by the test. They are not login tokens. Holding the raw SQLite connection bypasses the helper authorization boundary; a test explicitly demonstrates this limit. The model does not enforce immutable agreements against an administrator or implement a complete checkout lifecycle.

Reading the test record

The release test record gives the exact test count, failures, errors, skips, runtime versions and hashes of the executable files. A skipped feature check is not a pass for that feature. Test names and assertions are the evidence; the number of tests is only a count.

Exercises elsewhere in the book can ask you to draw a schedule, inspect a plan, choose a workload or design a real restore procedure. Not every such task is a ready-made automated implementation. PostgreSQL deployments, actual network faults, device flush behavior, complete tenant security and live schema migrations remain tasks for an appropriately isolated environment with explicit authorization.

A–Z Glossary

Definitions are retained with their teaching context. Some familiar terms have several related meanings. Chapter and section identifiers are stable across editions.

A

Accepted operation

a committed result under a declared contract — the business outcome represented by the transaction, not merely a received HTTP request.

Read in context: 51.1.6

Acknowledgement loss

the result exists but the sender did not learn it — a source of uncertainty rather than proof that the action failed.

Read in context: 51.3.6

Admission control

deciding which work to accept — protection of bounded resources through explicit entry policies.

Read in context: 50.4.6

Agreed price snapshot

the price stored at acceptance — an order fact distinct from the mutable catalogue.

Read in context: 51.2.6

Allocated cost

a share assigned under a rule — a reporting quantity based on an explicit apportionment policy.

Read in context: 50.5.6

Artefact identity

which exact candidate was used — a stable reference, often a digest, linking execution to reviewed bytes.

Read in context: 49.6.6

Authorisation

permission to perform an action — an access decision separate from whether the action is a valid domain transition.

permission for a particular act — evaluation of a principal's rights over a resource in a defined context.

Read in context: 47.1.6

Authoritative representation

the record a decision is defined to rely on — a source-of-truth role assigned by the architecture.

the selected source of meaning — the version whose value governs while multiple copies coexist.

Read in context: 49.2.6

B

Backfill

transforming already-stored records — population of a new representation under explicit source assumptions.

Read in context: 49.3.6

Boundary coverage

which protected paths were exercised — an inventory of tested interfaces, scopes and states, not a universal guarantee.

Read in context: 47.6.6

Bounded observation

a deliberately limited check — a query or measurement with named scope and minimal unintended change.

Read in context: 52.3.6

C

Cardinality

how many; here, the number of rows in a dataset or intermediate result.

how many related rows are permitted or required — the minimum and maximum counts on each side of a relationship.

how many records or matches there are — a count or relationship multiplicity whose context must be stated.

how many distinct labelled combinations exist — a property that can drive metric storage and query cost.

Read in context: 50.2.6

Compatibility window

the overlap period for contracts — the time during which specified old and new consumers must coexist.

Read in context: 49.1.6

Completion predicate

the exact rule for saying an action is done — a condition over declared tasks, evidence and exceptions.

Read in context: 48.6.6

Confused deputy

a privileged helper misused through supplied instructions — an authority-bearing component acting for the wrong caller or scope.

Read in context: 47.4.6

Connection context

identity-related state attached to database work — information whose establishment, reuse and clearing form part of the security boundary.

Read in context: 47.3.6

Consultation date

when a source was checked — a historical marker that does not turn a moving page into an immutable specification.

Read in context: 52.6.6

Consumer

anything relying on the data contract — a reader, writer, job, export, API adapter or replay process.

Read in context: 49.1.6

Contract

what participants can rely on; technically, specified inputs, outputs, invariants, errors and relevant guarantees.

retire the old path — removing obsolete fields, adapters or access after dependencies are checked.

Read in context: 49.2.6

Cost boundary

which charges are counted — the stated inclusions and exclusions behind a reported amount.

Read in context: 50.5.6

Cross-reference

a route to related material — a stable chapter, section or source identifier rather than a guessed page number.

Read in context: 52.1.6

Cryptographic erasure

making protected ciphertext unusable by sanitising necessary key material — a claim dependent on key scope and remaining copies.

Read in context: 48.5.6

Cutover

the point a new path becomes authoritative — a transition with explicit prerequisites and postconditions.

switching the authoritative path — a controlled transition from old to new representation or service behaviour.

Read in context: 49.4.6

D

Data minimisation

avoiding unnecessary stored detail — collecting and retaining only the information justified for the defined purpose.

keeping only what the task needs — reducing unnecessary fields, copies or lifetime under a stated requirement.

Read in context: 48.1.6

E

Effect count

how many accepted changes occurred — distinct from how many attempts or deliveries were observed.

Read in context: 51.4.6

Effective privilege

what the identity can actually do — the result of direct grants, memberships, ownership and special execution rules.

Read in context: 47.2.6

Error budget

the allowed nonconformance under an objective — an arithmetic planning quantity whose definition does not override other requirements.

Read in context: 50.1.6

Evidence boundary

the limit of a justified conclusion — the tested scope, assumptions and unresolved properties attached to a result.

Read in context: 51.6.6

Exclusion

a deliberately unimplemented responsibility — an explicit limit such as payment settlement or physical dispatch.

Read in context: 51.1.6

Executable assertion

a machine-checkable expectation — a predicate comparing an observed result with a declared outcome.

Read in context: 51.6.6

Expand

add a compatible new path — introducing representation or capability before removing the old one.

Read in context: 49.2.6

External outcome

an event beyond the local transaction — payment, delivery or another result needing separate evidence.

Read in context: 51.4.6

F

Fault injection

deliberately interrupting a known path — a test technique whose simulated failure must be named precisely.

Read in context: 51.5.6

Fault model

the failures an experiment assumes or injects — the specific process, operating-system, device or communication conditions being tested.

the failures considered — the boundary within which a mechanism's reasoning and tests are intended to apply.

Read in context: 52.4.6

Forecast error

the difference between prediction and observation — evidence used to update assumptions rather than hide uncertainty.

Read in context: 50.6.6

G

Glossary

a vocabulary reference — definitions connected to their contexts and teaching locations.

Read in context: 52.1.6

H

Headroom

usable spare capacity — margin under stated workload and failure assumptions, not a generic machine percentage.

Read in context: 50.4.6

Hold

an explicit suspension of an action — a governed exception with reason, authority and review conditions.

Read in context: 48.2.6

Hypothesis

a candidate explanation — a claim that should be distinguished from alternatives through suitable evidence.

Read in context: 52.3.6

I

Idempotency scope

where a replay identity is meaningful — the tenant, operation and retention boundary governing duplicate handling.

Read in context: 51.3.6

Implementation boundary

where a claim stops generalising — the product, version and configuration for which a detail is established.

Read in context: 52.1.6

Inbox

the consumer's delivery record — an identity committed with the local effect to support repeat handling.

Read in context: 51.3.6

Incremental maintenance

updating a result from a change — applying a justified delta instead of recomputing everything.

Read in context: 48.4.6

Integrity check

a check that retained content agrees with a reference — evidence of consistency, not automatic evidence of real-world truth.

inspection of declared structural properties — one layer of validation, distinct from complete business or security correctness.

Read in context: 51.5.6

Intervention threshold

the point to act before exhaustion — a chosen policy leaving time and resources for a controlled response.

Read in context: 50.3.6

Irreversible transformation

one that loses needed distinctions — a mapping whose original inputs cannot be recovered from the retained output alone.

Read in context: 49.5.6

K

Keyset progress

resuming from a stable ordered key — a traversal technique whose eligibility and concurrent-change boundaries still need design.

Read in context: 49.3.6

L

Least privilege

only the authority needed — minimising effective permissions and their duration for a defined responsibility.

Read in context: 47.2.6

Lineage

how one result came from other data; technically, the dependency path between inputs, transformations and outputs.

where a result came from — recorded relationships among inputs, transformations, executions and outputs.

Read in context: 48.4.6

Log

a recorded event — timestamped or otherwise ordered details describing a particular occurrence.

Read in context: 50.2.6

Logical deletion

removal from ordinary active data — a change to the database's visible record state.

Read in context: 48.3.6

Logical payload

the data's application representation — the meaningful values before engine and operational overhead.

Read in context: 50.3.6

M

Marginal cost

the change caused by an additional unit of work — distinct from dividing the total bill by current volume.

Read in context: 50.5.6

Metric

a measured quantity — an aggregated or sampled numeric signal with units and dimensions.

Read in context: 50.2.6

N

Negative control

a case that should be refused — a deliberately unauthorised or invalid operation with a checked outcome.

Read in context: 47.6.6

Net growth

increase after relevant additions and removals — a rate whose measurement window and storage boundary must be stated.

Read in context: 50.3.6

O

Operational owner

the person or team responsible for running and recovering the system — a named responsibility beyond writing its initial code.

the accountable responder — the person or team responsible for a signal, policy or unresolved task.

Read in context: 50.6.6

Outbox

committed work carries a delivery instruction — a transactional record of an event or message to be delivered by a separate process.

a durable list of external work implied by accepted changes — message-intent records committed atomically with local business state.

atomically recorded delivery intent — a local record that survives with the business change but does not guarantee external completion.

Read in context: 51.3.6

P

Policy bypass

an execution path exempt from a rule — special authority that must be accounted for when testing enforcement.

Read in context: 47.3.6

Policy oracle

the expected permission decision — the independently stated rule used to judge a test result.

Read in context: 47.6.6

Policy version

which rule was applied — an identity connecting a lifecycle decision to the exact governing definition.

Read in context: 48.2.6

Precondition

something that must hold before an action — a predicate required at an operation's entry boundary.

what must be true before a change — the checked state and authority on which safe execution depends.

Read in context: 49.6.6

Primary source

the originating documentation or research — evidence closer to the specified behaviour than a second-hand summary.

Read in context: 52.6.6

Principal

the person or program asking — an authenticated or otherwise explicitly established security identity used in policy evaluation.

Read in context: 47.1.6

Progress property

required work can eventually advance — a liveness claim dependent on assumptions about failures and resources.

Read in context: 52.4.6

Purpose limitation

using information for a defined need — an explicit boundary on collection and downstream use.

Read in context: 48.1.6

Q

Query grain

what one result row means — the level created by joins, projection and grouping.

Read in context: 52.2.6

R

Recomputation

rebuilding from an authorised source set — creating a new derived result under a recorded transformation version.

Read in context: 48.4.6

Reconciliation

comparing related accounts of the same work; technically, identifying and resolving or explicitly retaining discrepancies under a defined procedure.

checking whether related records agree as expected — a comparison of identities, relationships and totals rather than a single balancing sum.

explaining how the input relates to the output — accounting for candidates, repeats, rejections and file-level limits without silently losing material.

compare defined representations of the same work — an evidence-based check of populations, keys, amounts and explained differences.

compare related records to resolve disagreement — a procedure that checks quantities, identities and outcomes against stated invariants and evidence.

comparing expected and observed information — a set of coverage, identity, value and invariant checks across representations.

comparing meaningfully corresponding records — checking identities, values and declared exclusions rather than one headline total.

Read in context: 49.4.6

Reconciliation population

the records intended for comparison — a defined set that prevents mixing separate fixtures or tenants.

Read in context: 51.4.6

Record class

a group with a common responsibility — data categorised by purpose, access and lifecycle requirements rather than file extension alone.

Read in context: 48.1.6

Rehearsal

a bounded execution before live work — an experiment whose representativeness and limitations must be stated.

Read in context: 49.6.6

Reintroduction

retired information returning through recovery — restoration of an older state without later lifecycle decisions.

Read in context: 48.5.6

Release manifest

an inventory of delivered bytes — filenames, sizes and digests that support integrity checks.

Read in context: 51.6.6

Representation contract

meaning attached to stored values — units, range, missingness and interpretation beyond the physical type.

Read in context: 52.2.6

Reproducibility record

enough context to repeat a result — candidate, inputs, environment, operations and observations.

Read in context: 52.5.6

Residual copy

information remaining after selected actions — a retained, failed, blocked or unknown destination requiring explicit treatment.

Read in context: 48.6.6

Restore rehearsal

actually exercise the recovery procedure — a controlled test measuring whether the required state and service can be recovered.

recovering and checking a separate destination — evidence about a specific recovery procedure and input.

Read in context: 51.5.6

Restricted retention

kept for a bounded exception — retained data whose access and use differ from ordinary active records.

Read in context: 48.5.6

Retention trigger

the event starting a lifecycle rule — a defined transition such as closure, not an assumed timestamp.

Read in context: 48.2.6

Retry amplification

repeated attempts adding load during trouble — feedback that can worsen the condition callers are trying to recover from.

Read in context: 50.4.6

Revocation

withdrawing previously valid authority — disabling future accepted use, distinct from deleting one stored copy.

Read in context: 47.5.6

Revocation delay

time before a withdrawn permission stops working — the exposure window introduced by cached or long-lived authority.

Read in context: 47.1.6

Roll-forward

correcting through a subsequent change — a new reviewed transformation rather than automatic restoration of the past.

Read in context: 49.5.6

Rollback

cancelling uncommitted work; technically, discarding the transaction changes within the requested rollback scope.

abandoning uncommitted database work — restoration of the transaction's prior database state, not reversal of unrelated external actions.

abandon uncommitted transactional work — restoration of the transaction's database effects under its supported semantics.

returning within a defined reversible boundary — undoing a transaction or deployment, with scope that must be stated.

Read in context: 49.5.6

Rotation

replacing active secret material — transitioning consumers to new authority under an explicit lifecycle.

Read in context: 47.5.6

Row grain

what one row stands for — the unit of representation against which counts and aggregates must be interpreted.

what one row asserts — the identity and meaning that determine valid joins and constraints.

Read in context: 51.2.6

Row-level security

a filter or rule on individual rows — database policy enforcement beyond ordinary object privileges.

Read in context: 47.3.6

Runbook

a documented response path — observations, decisions, bounded actions and escalation conditions for an operational situation.

Read in context: 50.6.6

Runtime identity

the account serving ordinary work — the database or service identity used by the request-handling process.

Read in context: 47.2.6

S

Safety property

a prohibited outcome never occurs — a correctness claim such as no duplicate accepted effect under the stated model.

Read in context: 52.4.6

Sanitisation

rendering target media data infeasible to recover under a stated standard — a different layer from an empty application query.

Read in context: 48.3.6

Saturation

a limiting resource is at or near effective capacity — a condition in which additional demand tends to create waiting, rejection or deterioration rather than proportional useful work.

demand meeting a limiting resource — a condition producing increased waiting, rejection or reduced useful throughput.

Read in context: 50.4.6

Scope boundary

where the guarantee applies — the trusted helper, tenant, database and supported operations included in the claim.

Read in context: 51.1.6

Scoped key

an identifier plus its domain — a composite identity such as tenant_id and order_id.

Read in context: 47.4.6

Secret

protected capability-bearing information — a credential or key requiring controlled disclosure and use.

Read in context: 47.5.6

Semantic compatibility

preserving meaning across versions — agreement about units and interpretation, not only accepted syntax.

Read in context: 49.1.6

Service-level indicator

a measured service outcome — a defined quantity such as the fraction of eligible requests meeting a correctness and latency condition.

a measured service property — a defined observation of availability, latency, correctness or another relevant outcome.

Read in context: 50.1.6

Service-level objective

the target for an indicator — a specified level over a stated population and window.

Read in context: 50.1.6

Skipped check

a check not executed under current conditions — neither a pass nor proof of failure of the underlying feature.

Read in context: 52.5.6

Store inventory

the declared destinations to address — an explicit set whose completeness must be assessed, not assumed.

Read in context: 48.3.6

Symptom

what was observed — a result needing explanation, not an already-proven cause.

Read in context: 52.3.6

T

Tenant

one separately scoped customer organisation — an isolation domain within a service that may share infrastructure.

Read in context: 47.4.6

Test fixture

the controlled input state — synthetic or appropriately authorised data used for a specified check.

Read in context: 52.5.6

Tie-breaker

an additional rule for equal primary sort values — a subsequent sort expression used to distinguish otherwise tied rows.

an extra rule for equal first choices — additional ordering attributes that distinguish otherwise tied results.

an additional ordering key — a rule making otherwise tied results deterministic where required.

Read in context: 52.2.6

Trace

connected steps of an operation — spans and context describing work across a request path.

Read in context: 50.2.6

Transaction ownership

responsibility for starting and ending a transaction — a contract defining which component may commit, roll back or leave the connection active.

responsibility for completion — the component entitled to commit or roll back the particular unit of work.

Read in context: 51.2.6

Transformation precondition

what must be true before changing a row — the provenance, version or state justifying the operation.

Read in context: 49.3.6

V

Validation boundary

the exact state a check describes — its database, snapshot or time, schema and input scope.

Read in context: 49.4.6

Verification scope

what a check could actually observe — the stores, identities, versions and authority included in the observation.

Read in context: 48.6.6

Version register

the declared implementation/document context — a record distinguishing documented versions from the runtime actually tested.

Read in context: 52.6.6

Sources and Version Register

These primary sources support adjacent technical claims. The synthetic shop, arithmetic fixtures and teaching models are original examples. Versioned references are not automatically descriptions of a newer release.

[S01] PostgreSQL 17 documentation: Transactions

PostgreSQL Global Development Group

Atomic changes, commit and rollback; product-specific tutorial.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/tutorial-transactions.html

[S02] PostgreSQL 17 documentation: Constraints

PostgreSQL Global Development Group

CHECK, NOT NULL, primary-key and foreign-key behaviour.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/ddl-constraints.html

[S03] PostgreSQL 17 documentation: Sorting Rows (ORDER BY)

PostgreSQL Global Development Group

No guaranteed result order without explicit ordering.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/queries-order.html

[S04] PostgreSQL 17 documentation: Asynchronous Commit

PostgreSQL Global Development Group

Acknowledgement and durability depend on configuration.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/wal-async-commit.html

[S05] PROV-DM: The PROV Data Model (2013)

W3C; editors Luc Moreau and Paolo Missier

Entities, activities, agents and derivation in provenance.

Consulted: 23 September 2026

https://www.w3.org/TR/prov-dm/

[S06] Data on the Web Best Practices (2017)

W3C

Metadata, provenance, quality, versioning and persistent identifiers.

Consulted: 23 September 2026

https://www.w3.org/TR/dwbp/

[S07] RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format (2017)

T. Bray, editor; IETF

JSON value types, interoperable numbers, names and encoding.

Consulted: 23 September 2026

https://www.rfc-editor.org/info/rfc8259/

[S08] RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files (2005)

Y. Shafranovich; IETF

Informational RFC describing common CSV conventions; not a universal spreadsheet schema.

Consulted: 23 September 2026

https://www.rfc-editor.org/info/rfc4180/

[S09] RFC 3339: Date and Time on the Internet: Timestamps (2002)

G. Klyne and C. Newman; IETF

Timestamp syntax and numeric UTC offsets.

Consulted: 23 September 2026

https://www.rfc-editor.org/info/rfc3339/

[S10] PostgreSQL 17 documentation: Numeric Types

PostgreSQL Global Development Group

Integer ranges, exact numeric types and floating-point types.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/datatype-numeric.html

[S11] PostgreSQL 17 documentation: Date/Time Types

PostgreSQL Global Development Group

Date and timestamp semantics; time-zone handling.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/datatype-datetime.html

[S12] Python 3.12 tutorial: Floating-Point Arithmetic, Issues and Limitations

Python Software Foundation

Binary fractions and displayed floating-point results.

Consulted: 23 September 2026

https://docs.python.org/3.12/tutorial/floatingpoint.html

[S13] Python 3.12 library reference: decimal

Python Software Foundation

Decimal construction, precision, quantisation and rounding contexts.

Consulted: 23 September 2026

https://docs.python.org/3.12/library/decimal.html

[S14] FAQ: UTF-8, UTF-16, UTF-32 and BOM

Unicode Consortium

Unicode encoding forms and UTF-8 byte lengths.

Consulted: 23 September 2026

https://www.unicode.org/faq/utf_bom.html

[S15] Unicode Standard Annex #15: Unicode Normalization Forms

Unicode Consortium

Canonical equivalence, NFC and the limits of normalisation.

Consulted: 23 September 2026

https://www.unicode.org/reports/tr15/

[S16] Unicode Standard Annex #29: Unicode Text Segmentation

Unicode Consortium

Grapheme clusters and user-perceived character boundaries.

Consulted: 23 September 2026

https://www.unicode.org/reports/tr29/

[S17] PostgreSQL 17 documentation: Comparison Functions and Operators

PostgreSQL Global Development Group

NULL comparisons and IS NULL.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/functions-comparison.html

[S18] PostgreSQL 17 documentation: Logical Operators

PostgreSQL Global Development Group

Three-valued Boolean logic.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/functions-logical.html

[S19] PostgreSQL 17 documentation: Aggregate Functions

PostgreSQL Global Development Group

COUNT and AVG treatment of non-null inputs.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/functions-aggregate.html

[S20] Python 3.12 library reference: sqlite3

Python Software Foundation

Embedded SQLite connections and bound parameters.

Consulted: 23 September 2026

https://docs.python.org/3.12/library/sqlite3.html

[S21] Datatypes in SQLite

SQLite project

Dynamic typing, storage classes and differences from other engines.

Consulted: 23 September 2026

https://www.sqlite.org/datatype3.html

[S22] Python 3.12 library reference: Built-in Types

Python Software Foundation

Integer precision and the bool subclass relationship.

Consulted: 23 September 2026

https://docs.python.org/3.12/library/stdtypes.html

[S23] Python 3.12 library reference: datetime

Python Software Foundation

Aware datetimes, offsets and conversion to UTC.

Consulted: 23 September 2026

https://docs.python.org/3.12/library/datetime.html

[S24] SQL Language Expressions

SQLite project

NULL and logical operations in the supplied SQLite lab.

Consulted: 23 September 2026

https://www.sqlite.org/lang_expr.html

[S25] Transaction

SQLite project

Explicit BEGIN, COMMIT and ROLLBACK in the supplied lab.

Consulted: 23 September 2026

https://www.sqlite.org/lang_transaction.html

[S26] VIM3, 2.3: Measurand

JCGM / BIPM

Defining the quantity intended to be measured.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/2.3.html

[S27] SI prefixes

BIPM

Decimal factors and prefix symbols; conversions in the text are original calculations.

Consulted: 23 September 2026

https://www.bipm.org/en/measurement-units/si-prefixes

[S28] VIM3, 2.13: Measurement accuracy

JCGM / BIPM

Accuracy is distinguished from precision.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/2.13.html

[S29] VIM3, 2.15: Measurement precision

JCGM / BIPM

Agreement among repeated indications under specified conditions.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/2.15.html

[S30] VIM3, 2.39: Calibration

JCGM / BIPM

Calibration is not the same operation as adjustment or verification.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/2.39.html

[S31] Combining uncertainty components

NIST

Propagation of uncertainty, sensitivities and covariances.

Consulted: 23 September 2026

https://physics.nist.gov/cuu/Uncertainty/combination.html

[S32] Type A evaluation of standard uncertainty

NIST

Standard uncertainty of a mean for independent observations.

Consulted: 23 September 2026

https://physics.nist.gov/cuu/Uncertainty/typea.html

[S33] VIM3, 4.14: Resolution

JCGM / BIPM

A perceptible response to a change in the measured quantity.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/4.14.html

[S34] Type B evaluation of standard uncertainty

NIST

Uncertainty evaluated from other information and assumed distributions.

Consulted: 23 September 2026

https://physics.nist.gov/cuu/Uncertainty/typeb.html

[S35] VIM3, 2.26: Measurement uncertainty

JCGM / BIPM

Dispersion of values attributed to a measurand using available information.

Consulted: 23 September 2026

https://jcgm.bipm.org/vim/en/2.26.html

[S36] Expanded uncertainty and coverage factors

NIST

U = k times combined standard uncertainty; coverage depends on assumptions.

Consulted: 23 September 2026

https://physics.nist.gov/cuu/Uncertainty/coverage.html

[S37] PostgreSQL 17 documentation: Identity Columns

PostgreSQL Global Development Group

Generated values do not by themselves enforce uniqueness.

Consulted: 23 September 2026

https://www.postgresql.org/docs/17/ddl-identity-columns.html

[S38] RFC 9562: Universally Unique IDentifiers (UUIDs), 2024

IETF; K. Davis, B. Peabody and P. Leach

UUID version 4 has 122 random bits; collision estimate is an original conditional calculation.

Consulted: 23 September 2026

https://www.rfc-editor.org/rfc/rfc9562.html

[S39] CloudEvents specification, version 1.0.2

Cloud Native Computing Foundation; CloudEvents project

Event source and id uniqueness; the wire specversion value is 1.0.

Consulted: 23 September 2026

https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/spec.md

[S40] Event Sourcing pattern

Microsoft Azure Architecture Center

Architecture context: event history, projections, snapshots, replay and trade-offs. The shop state machines are original teaching designs, not a Microsoft reference implementation.

Consulted: 23 September 2026

https://learn.microsoft.com/en-us/azure/architecture/patterns/event-sourcing

[S41] Time, Clocks, and the Ordering of Events in a Distributed System (1978)

Leslie Lamport

Communications of the ACM 21(7), 558–565; happened-before and logical clocks. Counter exercises are original illustrations.

Consulted: 23 September 2026

https://lamport.azurewebsites.net/pubs/time-clocks.pdf

[S42] Python 3.13 library reference: pathlib

Python Software Foundation

Pure paths versus filesystem operations; path components and lexical interpretation.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/pathlib.html

[S43] Python 3.13 library reference: io

Python Software Foundation

Text/byte streams, encoding, newline handling and input boundaries.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/io.html

[S44] Python 3.13 library reference: csv

Python Software Foundation

CSV dialects, reader and writer behaviour; strict mode is not domain validation.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/csv.html

[S45] Python 3.13 library reference: json

Python Software Foundation

Object pairs hooks, numeric constants, supported types and parser resource cautions.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/json.html

[S46] Python 3.13 library reference: struct

Python Software Foundation

Explicit byte order, widths and binary packing; the KDBF envelope is original teaching material.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/struct.html

[S47] Apache Parquet documentation: Overview

Apache Software Foundation

Column-oriented file-format purpose; no Parquet performance measurements are claimed.

Consulted: 24 September 2026

https://parquet.apache.org/docs/overview/

[S48] Apache Avro 1.12.0 specification

Apache Software Foundation

Schema-based representation and writer/reader schema resolution; not implemented by the toy envelope.

Consulted: 24 September 2026

https://avro.apache.org/docs/1.12.0/specification/

[S49] Model for Tabular Data and Metadata on the Web

W3C

Representations, semantic values, cells and annotations in tabular data.

Consulted: 24 September 2026

https://www.w3.org/TR/tabular-data-model/

[S50] SQLite Is Serverless

SQLite project

Embedded, in-process engine versus a separately running database server.

Consulted: 24 September 2026

https://www.sqlite.org/serverless.html

[S51] Appropriate Uses For SQLite

SQLite project

Embedded-use scope, local access and situations favouring client-server databases.

Consulted: 24 September 2026

https://www.sqlite.org/whentouse.html

[S52] PostgreSQL 17 documentation: Architectural Fundamentals

PostgreSQL Global Development Group

Database client/server architecture and separation from application code.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/tutorial-arch.html

[S53] Atomic Commit In SQLite

SQLite project

Journaling and filesystem/device assumptions. This lab does not inject power failures.

Consulted: 24 September 2026

https://www.sqlite.org/atomiccommit.html

[S54] Isolation In SQLite

SQLite project

Writer serialization and transaction visibility; local lock demonstration is separately bounded.

Consulted: 24 September 2026

https://www.sqlite.org/isolation.html

[S55] SQLite Online Backup API

SQLite project

Engine-supported copying into a destination database; no production recovery-time claim.

Consulted: 24 September 2026

https://www.sqlite.org/backup.html

[S56] PostgreSQL 17 documentation: Table Basics

PostgreSQL Global Development Group

Tables, columns and declared data types.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/ddl-basics.html

[S57] PostgreSQL 17 documentation: Select Lists

PostgreSQL Global Development Group

Projection, output columns and duplicate elimination.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/queries-select-lists.html

[S58] PostgreSQL 17 documentation: Table Expressions

PostgreSQL Global Development Group

Joins, outer joins, qualification and the effect of later filtering.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/queries-table-expressions.html

[S59] SQLite Foreign Key Support

SQLite project

Per-connection enforcement, parent/child references and nullable child keys.

Consulted: 24 September 2026

https://www.sqlite.org/foreignkeys.html

[S60] STRICT Tables

SQLite project

SQLite 3.37.0 minimum, strict storage types and permitted lossless coercion.

Consulted: 24 September 2026

https://www.sqlite.org/stricttables.html

[S61] CREATE TABLE

SQLite project

CHECK, NOT NULL, uniqueness and primary-key implementation details.

Consulted: 24 September 2026

https://www.sqlite.org/lang_createtable.html

[S62] Python 3.13 library reference: pickle

Python Software Foundation

Do not unpickle untrusted data; serialization is not automatically safe execution.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/pickle.html

[S63] CSV Injection

OWASP Foundation community

Spreadsheet formula interpretation is separate from CSV quotation; no universal sanitizer is claimed.

Consulted: 24 September 2026

https://community.owasp.org/attacks/CSV_Injection

[S64] Python 3.13 library reference: os

Python Software Foundation

Filesystem operations, replacement semantics and system-dependent boundaries.

Consulted: 24 September 2026

https://docs.python.org/3.13/library/os.html

[S65] PostgreSQL 17 documentation: Using EXPLAIN

PostgreSQL Global Development Group

Plan costs and actual execution with EXPLAIN ANALYZE.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/using-explain.html

[S66] PostgreSQL 17 documentation: Transaction Isolation

PostgreSQL Global Development Group

Engine-specific isolation guarantees and transaction retry requirements.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/transaction-iso.html

[S67] PostgreSQL 17 documentation: SQL Dump

PostgreSQL Global Development Group

Logical backup and restoration; cited context, not executed in the SQLite lab.

Consulted: 24 September 2026

https://www.postgresql.org/docs/17/backup-dump.html

[S68] PostgreSQL 17 documentation: CREATE DOMAIN

PostgreSQL Global Development Group

Reusable constrained base types and domain-nullability caveats; the SQL illustration is not executed in the SQLite lab.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-createdomain.html

[S69] PostgreSQL 17 documentation: Schemas

PostgreSQL Global Development Group

Schema namespaces, qualified names and name-resolution context; distinct from the general modelling meaning of schema.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-schemas.html

[S70] PostgreSQL 17 documentation: COMMENT

PostgreSQL Global Development Group

Object comments as descriptive metadata, not enforced business rules or a place for secrets.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-comment.html

[S71] PostgreSQL 17 documentation: The Information Schema

PostgreSQL Global Development Group

Standard-oriented metadata inspection and its distinction from business meaning.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/information-schema.html

[S72] PostgreSQL 17 documentation: Lexical Structure

PostgreSQL Global Development Group

Identifiers, text literals and quoting; naming conventions in the book are editorial choices.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-syntax-lexical.html

[S73] PostgreSQL 17 documentation: SET CONSTRAINTS

PostgreSQL Global Development Group

Eligible constraint classes and checking-mode changes; PostgreSQL is documented, not executed.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-set-constraints.html

[S74] PostgreSQL 17 documentation: CREATE TABLE

PostgreSQL Global Development Group

Keys, references, match semantics and deferrability; product-specific rather than a cross-engine promise.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-createtable.html

[S75] PRAGMA Statements

SQLite project

Metadata interfaces and separate scopes of foreign_keys, integrity_check and foreign_key_check.

Consulted: 25 September 2026

https://www.sqlite.org/pragma.html

[S76] The ON CONFLICT Clause

SQLite project

Default ABORT statement rollback and the distinction between replacement and an ordinary update.

Consulted: 25 September 2026

https://www.sqlite.org/lang_conflict.html

[S77] Result and Error Codes

SQLite project

Machine-readable error categories; selected extended constraint codes observed by the lab.

Consulted: 25 September 2026

https://www.sqlite.org/rescode.html

[S78] PostgreSQL 17 documentation: Modifying Tables

PostgreSQL Global Development Group

Constraint-change and existing-data considerations; no production or PostgreSQL migration is run.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-alter.html

[S79] PostgreSQL 17 documentation: Querying a Table

PostgreSQL Global Development Group

SELECT fundamentals, source columns, expressions and result questions.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/tutorial-select.html

[S80] SELECT

SQLite project

Logical result semantics, output expressions, filtering, duplicate elimination and ordering.

Consulted: 25 September 2026

https://www.sqlite.org/lang_select.html

[S81] PostgreSQL 17 documentation: LIMIT and OFFSET

PostgreSQL Global Development Group

Output limits and the need for predictable ordering; not a performance bound.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/queries-limit.html

[S82] PostgreSQL 17 documentation: Conditional Expressions

PostgreSQL Global Development Group

CASE and COALESCE as explicit conditional choices; the narrative fixtures are original.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/functions-conditional.html

[S83] INSERT

SQLite project

Explicit target columns and value insertion in isolated draft examples.

Consulted: 25 September 2026

https://www.sqlite.org/lang_insert.html

[S84] UPDATE

SQLite project

Assignment and predicate scope; unqualified update demonstrated only in a disposable transaction.

Consulted: 25 September 2026

https://www.sqlite.org/lang_update.html

[S85] DELETE

SQLite project

Selected-row deletion semantics; not a claim of erasure across backups or external copies.

Consulted: 25 September 2026

https://www.sqlite.org/lang_delete.html

[S86] Python 3.13 library reference: sqlite3

Python Software Foundation

Driver binding, cursors, result counts, connection closing and explicit transaction configuration.

Consulted: 25 September 2026

https://docs.python.org/3.13/library/sqlite3.html

[S87] Built-In Scalar SQL Functions

SQLite project

COALESCE behaviour, including the difference between NULL and empty text.

Consulted: 25 September 2026

https://www.sqlite.org/lang_corefunc.html

[S88] Database System Concepts, seventh edition: Chapter 7 author slides, Normalization

Abraham Silberschatz, Henry F. Korth and S. Sudarshan

Functional dependencies, lossless decomposition, dependency preservation, BCNF and third normal form. The shop exercises are original.

Consulted: 25 September 2026

https://www.db-book.com/slides-dir/PDF-dir/ch7.pdf

[S89] Query Planning

SQLite project

Scans, ordered lookup, compound and covering indexes, and sorting choices.

Consulted: 25 September 2026

https://www.sqlite.org/queryplanner.html

[S90] EXPLAIN QUERY PLAN

SQLite project

Human-readable SCAN, SEARCH and temporary-tree descriptions; output format is not a stable application interface.

Consulted: 25 September 2026

https://www.sqlite.org/eqp.html

[S91] PostgreSQL 17: Multicolumn Indexes

PostgreSQL Global Development Group

Column order and constraints on efficient access in the explicitly cited version.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/indexes-multicolumn.html

[S92] PostgreSQL 17: Index-Only Scans and Covering Indexes

PostgreSQL Global Development Group

INCLUDE payloads and visibility checks; covering does not guarantee zero heap visits.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/indexes-index-only-scans.html

[S93] Partial Indexes

SQLite project

Predicate-restricted indexes and eligibility based on implication.

Consulted: 25 September 2026

https://www.sqlite.org/partialindex.html

[S94] PostgreSQL 17: Statistics Used by the Planner

PostgreSQL Global Development Group

Sampling, distribution estimates and extended statistics.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/planner-stats.html

[S95] PostgreSQL 17: Resource Consumption

PostgreSQL Global Development Group

Operation-level memory allowances and spill; not a benchmark of this manuscript.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/runtime-config-resource.html

[S96] PostgreSQL 17 tutorial: Window Functions

PostgreSQL Global Development Group

Window partitions, ordering and preservation of detail rows.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/tutorial-window.html

[S97] Window Functions

SQLite project

Window frames, peers and aggregate-window behaviour in the educational SQL exercises.

Consulted: 25 September 2026

https://www.sqlite.org/windowfunctions.html

[S98] The SQLite Query Optimizer Overview

SQLite project

Access-path transformations, joins and implementation-specific planning decisions.

Consulted: 25 September 2026

https://www.sqlite.org/optoverview.html

[S99] Python 3.13: hashlib

Python Software Foundation

Digest interfaces; the collision and comparison exercises are original.

Consulted: 25 September 2026

https://docs.python.org/3.13/library/hashlib.html

[S100] FIPS 180-4: Secure Hash Standard, August 2015

NIST

Standardized SHA-2 digests, distinguished from authentication and encryption.

Consulted: 25 September 2026

https://csrc.nist.gov/pubs/fips/180-4/upd1/final

[S101] PostgreSQL 17: Hash Indexes

PostgreSQL Global Development Group

Equality-oriented, lossy hash access and collision rechecking.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/hash-index.html

[S102] Python 3.13: time

Python Software Foundation

Monotonic performance timers and their units, not an assertion of nanosecond accuracy.

Consulted: 25 September 2026

https://docs.python.org/3.13/library/time.html

[S103] Monitoring Distributed Systems

Rob Ewaschuk; Google SRE

Latency, traffic, errors and saturation as monitoring context; workload examples are invented.

Consulted: 25 September 2026

https://sre.google/sre-book/monitoring-distributed-systems/

[S104] PostgreSQL 17: Index Types

PostgreSQL Global Development Group

Index methods support different operators; a method name is not a universal performance promise.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/indexes-types.html

[S105] PostgreSQL 17: B-Tree Indexes

PostgreSQL Global Development Group

B-tree structure and implementation notes. The hand-worked B+ tree is deliberately not a PostgreSQL implementation.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/btree.html

[S106] PostgreSQL 17: Indexes and ORDER BY

PostgreSQL Global Development Group

Ordered access and explicit query ordering.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/indexes-ordering.html

[S107] PostgreSQL 17: ANALYZE

PostgreSQL Global Development Group

Updating sampled planner statistics and operational scope.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-analyze.html

[S108] Python 3.13: Data Model

Python Software Foundation

Hash/equality contracts and process-randomized string and byte hashes.

Consulted: 25 September 2026

https://docs.python.org/3.13/reference/datamodel.html

[S109] Python 3.13: bisect

Python Software Foundation

Ordered-list insertion positions; insertion cost and concurrent-mutation limitations.

Consulted: 25 September 2026

https://docs.python.org/3.13/library/bisect.html

[S110] PostgreSQL 17: Window Functions

PostgreSQL Global Development Group

Ranking, offsets, frames and frame-sensitive last_value behaviour.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/functions-window.html

[S111] PostgreSQL 17: The Path of a Query

PostgreSQL Global Development Group

Parsing, rewriting, planning and execution stages.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/query-path.html

[S112] PostgreSQL 17: Planner/Optimizer

PostgreSQL Global Development Group

Plan search, nested-loop, merge and hash joins; optimality is bounded by search and estimation.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/planner-optimizer.html

[S113] PostgreSQL 17: EXPLAIN

PostgreSQL Global Development Group

Diagnostic options, actual execution with ANALYZE, instrumentation boundaries and non-text formats.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-explain.html

[S114] PostgreSQL 17: Query Planning

PostgreSQL Global Development Group

Planner cost and method settings; experimentation is distinct from production tuning.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/runtime-config-query.html

[S115] PostgreSQL 17: Row Estimation Examples

PostgreSQL Global Development Group

Selectivity estimation and its assumptions; the shop arithmetic is original.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/row-estimation-examples.html

[S116] PostgreSQL 17: WITH Queries (Common Table Expressions)

PostgreSQL Global Development Group

Named query stages and materialisation/folding behaviour in the explicitly cited version.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/queries-with.html

[S117] PostgreSQL 17 tutorial: Transactions

PostgreSQL Global Development Group

Transaction blocks, commit, rollback and the scope of database changes.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/tutorial-transactions.html

[S118] PostgreSQL 17: Explicit Locking

PostgreSQL Global Development Group

Lock modes, row/table distinctions, deadlocks and advisory-lock lifetimes.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/explicit-locking.html

[S119] PostgreSQL 17: Concurrency Control Introduction

PostgreSQL Global Development Group

Multiversion visibility and distinction from application history.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/mvcc-intro.html

[S120] PostgreSQL 17: Serialization Failure Handling

PostgreSQL Global Development Group

Whole-transaction retries, SQLSTATE 40001 and careful classification of other failures.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/mvcc-serialization-failure-handling.html

[S121] PostgreSQL 17: SAVEPOINT

PostgreSQL Global Development Group

Savepoint scope within a transaction.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-savepoint.html

[S122] PostgreSQL 17: Sequence Manipulation Functions

PostgreSQL Global Development Group

Sequence allocation and non-rollback behaviour; gaps are not missing-business-event proof.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/functions-sequence.html

[S123] PostgreSQL 17: Routine Vacuuming

PostgreSQL Global Development Group

Version cleanup, visibility maps, space reuse and transaction-identifier maintenance.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/routine-vacuuming.html

[S124] PostgreSQL 17: System Columns

PostgreSQL Global Development Group

Version-related metadata and the limits of physical tuple identifiers.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-system-columns.html

[S125] Transactional outbox pattern

Amazon Web Services

Atomic recording of business changes and delivery intent; duplicate delivery still requires handling.

Consulted: 25 September 2026

https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/transactional-outbox.html

[S126] Making retries safe with idempotent APIs

Amazon Web Services Builders Library

Caller request identity, repeated intent and semantic-equivalence considerations.

Consulted: 25 September 2026

https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/

[S127] Savepoints

SQLite project

ROLLBACK TO and RELEASE, including the difference between inner release and outer commit.

Consulted: 25 September 2026

https://www.sqlite.org/lang_savepoint.html

[S128] PostgreSQL 17 documentation: Database Page Layout

PostgreSQL Global Development Group

Page headers, item identifiers, tuple layout and implementation boundaries.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/storage-page-layout.html

[S129] PostgreSQL 17 documentation: Write-Ahead Logging

PostgreSQL Global Development Group

WAL-before-data ordering, redo and grouped synchronization.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/wal-intro.html

[S130] PostgreSQL 17 documentation: Reliability

PostgreSQL Global Development Group

Storage-cache assumptions, partial page writes and reliability limits.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/wal-reliability.html

[S131] PostgreSQL 17 documentation: Continuous Archiving and Point-in-Time Recovery

PostgreSQL Global Development Group

Base backups, continuous WAL coverage, recovery targets and timelines.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/continuous-archiving.html

[S132] Write-Ahead Logging

SQLite project

WAL frames, checkpointing, concurrency limits and the disclosed 2026 WAL-reset bug; not an endorsement of the historical lab runtime.

Consulted: 25 September 2026

https://www.sqlite.org/wal.html

[S133] Database File Format

SQLite project

Page sizes, header fields, overflow and journal/WAL representation.

Consulted: 25 September 2026

https://www.sqlite.org/fileformat.html

[S134] Compaction

RocksDB project

Sorted runs, leveled/tiered strategies and read/write/space trade-offs.

Consulted: 25 September 2026

https://github.com/facebook/rocksdb/wiki/Compaction

[S135] PostgreSQL 17 documentation: WAL Configuration

PostgreSQL Global Development Group

Checkpoints, redo positions and resource trade-offs.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/wal-configuration.html

[S136] PostgreSQL 17 documentation: Asynchronous Commit

PostgreSQL Global Development Group

Acknowledgement before WAL flush and the distinction from disabling fsync.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/wal-async-commit.html

[S137] PostgreSQL 17 documentation: Data Checksums

PostgreSQL Global Development Group

Detection scope and limitations of page checksums.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/checksums.html

[S138] PostgreSQL 17 documentation: pg_verifybackup

PostgreSQL Global Development Group

Manifest verification and the explicit need for test restores.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/app-pgverifybackup.html

[S139] How To Corrupt An SQLite Database File

SQLite project

Operational corruption hazards, copying and journal-handling cautions.

Consulted: 25 September 2026

https://www.sqlite.org/howtocorrupt.html

[S140] RocksDB Overview

RocksDB project

Memtables, log files, sorted tables and the library boundary.

Consulted: 25 September 2026

https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

[S141] fsync(2)

Linux man-pages project

File synchronization, directory metadata and error reporting; Linux-specific.

Consulted: 25 September 2026

https://man7.org/linux/man-pages/man2/fsync.2.html

[S142] write(2)

Linux man-pages project

Partial writes and the difference between write success and persistence.

Consulted: 25 September 2026

https://man7.org/linux/man-pages/man2/write.2.html

[S143] PostgreSQL 17 documentation: TOAST

PostgreSQL Global Development Group

Large-value compression/out-of-line storage and physical row boundaries.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/storage-toast.html

[S144] PostgreSQL 17 documentation: The Cumulative Statistics System

PostgreSQL Global Development Group

I/O statistics, cache boundaries, update timing and monitoring scope.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/monitoring-stats.html

[S145] VACUUM

SQLite project

File rebuilding, space reclamation and operational constraints.

Consulted: 25 September 2026

https://www.sqlite.org/lang_vacuum.html

[S146] PostgreSQL 17 documentation: amcheck

PostgreSQL Global Development Group

Table/index structural verification and limitations.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/amcheck.html

[S147] PostgreSQL 17 documentation: Log-Shipping Standby Servers

PostgreSQL Global Development Group

Physical streaming, lag, slots and synchronous standby acknowledgement boundaries.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/warm-standby.html

[S148] PostgreSQL 17 documentation: Logical Replication

PostgreSQL Global Development Group

Publication/subscription, initial synchronization, identities and conflicts.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/logical-replication.html

[S149] PostgreSQL 17 documentation: Write Ahead Log settings

PostgreSQL Global Development Group

synchronous_commit modes, local/remote flush and replay distinctions.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/runtime-config-wal.html

[S150] PostgreSQL 17 documentation: Failover

PostgreSQL Global Development Group

Promotion, external failure detection and preventing two active primaries.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/warm-standby-failover.html

[S151] PostgreSQL 17 documentation: pg_rewind

PostgreSQL Global Development Group

Divergent histories, prerequisites and recovery/reconfiguration cautions.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/app-pgrewind.html

[S152] PostgreSQL 17 documentation: Table Partitioning

PostgreSQL Global Development Group

Range/list/hash table partitions, pruning and implementation restrictions; not automatic multi-host sharding.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-partitioning.html

[S153] Perspectives on the CAP Theorem

Seth Gilbert and Nancy A. Lynch

Authors' retrospective explaining atomic read/write consistency, availability, unreliable communication and the proof sketch.

Consulted: 25 September 2026

https://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf

[S154] In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout

Raft terms, durable voting, log matching, current-term commitment, membership and client-read safeguards.

Consulted: 25 September 2026

https://raft.github.io/raft.pdf

[S155] PostgreSQL 17 documentation: Two-Phase Transactions

PostgreSQL Global Development Group

Prepared-transaction state and the external transaction-manager boundary.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/two-phase.html

[S156] PostgreSQL 17 documentation: PREPARE TRANSACTION

PostgreSQL Global Development Group

Prepared state, retained locks, completion responsibilities and operational cautions.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-prepare-transaction.html

[S157] Sagas (1987)

Hector Garcia-Molina and Kenneth Salem

Original paper on long-lived transactions, interleaving and application-defined compensation.

Consulted: 25 September 2026

https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf

[S158] Consumer Acknowledgements and Publisher Confirms

RabbitMQ project

Separate confirmation boundaries, delivery tags, redelivery and bounded unacknowledged work; unversioned documentation.

Consulted: 25 September 2026

https://www.rabbitmq.com/docs/confirms

[S159] Queues

RabbitMQ project

Ordering scope, multiple consumers, redelivery and queue properties; unversioned documentation.

Consulted: 25 September 2026

https://www.rabbitmq.com/docs/queues

[S160] Client-side caching reference

Redis

Invalidation races, connection loss and cache-resource bounds; unversioned documentation.

Consulted: 25 September 2026

https://redis.io/docs/latest/develop/reference/client-side-caching/

[S161] Cache-Aside pattern

Microsoft

Loading/invalidation trade-offs and consistency limits of derived caches.

Consulted: 25 September 2026

https://learn.microsoft.com/en-us/azure/architecture/patterns/cache-aside

[S162] Dynamo: Amazon's Highly Available Key-value Store (2007)

Giuseppe DeCandia and co-authors

Original Dynamo design: consistent hashing, versions, sloppy quorums and reconciliation; not a statement about current DynamoDB.

Consulted: 25 September 2026

https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[S163] The Chubby lock service for loosely-coupled distributed systems (2006)

Mike Burrows

Lock generations and recipient-validated sequencers for rejecting obsolete operations.

Consulted: 25 September 2026

https://storage.googleapis.com/gweb-research2023-media/pubtools/4444.pdf

[S164] PostgreSQL 17 documentation: Logical Replication Restrictions

PostgreSQL Global Development Group

DDL, sequence and object coverage boundaries and subscriber compatibility.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/logical-replication-restrictions.html

[S165] PostgreSQL connector documentation, stable page showing version 3.6 when consulted

Debezium project

Initial snapshot/change-stream boundary, offsets and connector failure behaviour; no connector deployment is performed.

Consulted: 25 September 2026

https://debezium.io/documentation/reference/stable/connectors/postgresql.html

[S166] PostgreSQL 17 documentation: Logical Decoding Concepts

PostgreSQL Global Development Group

Slots, retained log positions and possible change redelivery following a crash.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/logicaldecoding-explanation.html

[S167] Apache Beam Programming Guide: windows, watermarks, triggers and state

Apache Software Foundation

Event-time membership, late-data policy, triggers and accumulation modes; the small window model is original, not a Beam runtime.

Consulted: 25 September 2026

https://beam.apache.org/documentation/programming-guide/

[S168] Apache Iceberg table specification

Apache Software Foundation

Field IDs, snapshots, manifests, atomic metadata replacement and snapshot retention. Discussion is limited to these concepts; not an implementation of the entire evolving specification.

Consulted: 25 September 2026

https://iceberg.apache.org/spec/

[S169] PostgreSQL 17 documentation: Materialized Views

PostgreSQL Global Development Group

Stored query results and refresh/freshness trade-offs.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/rules-materializedviews.html

[S170] Apache Parquet: File Format

Apache Software Foundation

File metadata, row groups and column chunks; companion byte-layout model does not produce Parquet files.

Consulted: 25 September 2026

https://parquet.apache.org/docs/file-format/

[S171] Apache Parquet: Encodings

Apache Software Foundation

Plain, dictionary, run-length and delta encoding concepts; actual Parquet bitstream rules are distinct from the toy codec.

Consulted: 25 September 2026

https://parquet.apache.org/docs/file-format/data-pages/encodings/

[S172] Apache Parquet: Page Index

Apache Software Foundation

Optional statistics and offsets for conservative page skipping.

Consulted: 25 September 2026

https://parquet.apache.org/docs/file-format/pageindex/

[S173] SQLite FTS5 Extension

SQLite project

Tokenizers, phrase queries, external-content responsibilities and negative BM25 scores. Only basic available FTS5 facilities are exercised.

Consulted: 25 September 2026

https://www.sqlite.org/fts5.html

[S174] PostgreSQL 17 documentation: Full Text Search Introduction

PostgreSQL Global Development Group

Documents, lexemes and full-text query representation.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/textsearch-intro.html

[S175] PostgreSQL 17 documentation: Controlling Text Search

PostgreSQL Global Development Group

Parsing, normalization, query construction, ranking and safe display limitations.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/textsearch-controls.html

[S176] MetricType and distances

Faiss project

Squared Euclidean distance, inner product, cosine and normalization. The lab uses direct Python arithmetic, not Faiss.

Consulted: 25 September 2026

https://github.com/facebookresearch/faiss/wiki/MetricType-and-distances

[S177] Faiss indexes

Faiss project

Exhaustive versus approximate indexes, vector-memory estimates and index trade-offs.

Consulted: 25 September 2026

https://github.com/facebookresearch/faiss/wiki/Faiss-indexes

[S178] Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

Yu. A. Malkov and D. A. Yashunin

Original HNSW research, first submitted 2016; graph navigation concept, not a claim about every present product or workload.

Consulted: 25 September 2026

https://arxiv.org/abs/1603.09320

[S179] e-Handbook of Statistical Methods: Autocorrelation

NIST / SEMATECH

Dependence between observations across lags; repeated observations are not automatically independent.

Consulted: 25 September 2026

https://www.itl.nist.gov/div898/handbook/eda/section3/eda35c.htm

[S180] e-Handbook of Statistical Methods: Scatter Plot

NIST / SEMATECH

Association can be inspected graphically but does not establish cause. All business examples in the chapter are synthetic.

Consulted: 25 September 2026

https://www.itl.nist.gov/div898/handbook/eda/section3/scatterp.htm

[S181] Introduction to Information Retrieval: Evaluation of unranked retrieval sets

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze

Authors-hosted textbook definitions of precision and recall; all local query judgements are synthetic.

Consulted: 25 September 2026

https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-unranked-retrieval-sets-1.html

[S182] Introduction to Information Retrieval: Okapi BM25, a non-binary model

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze

Term-frequency saturation, document-length normalization and development-set tuning.

Consulted: 25 September 2026

https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model-1.html

[S183] e-Handbook of Statistical Methods: Confidence intervals for a proportion

NIST / SEMATECH

Wilson interval formula, binomial assumptions and small-sample cautions. Numerical examples in the book are original.

Consulted: 25 September 2026

https://www.itl.nist.gov/div898/handbook/prc/section2/prc241.htm

[S184] PostgreSQL 17 documentation: Preferred Index Types for Text Search

PostgreSQL Global Development Group

GIN inverted indexes and GiST signatures/rechecking; PostgreSQL examples are documented, not executed.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/textsearch-indexes.html

[S185] Authorization Cheat Sheet

OWASP Foundation

Default-deny authorization, permission checks and tests; not authentication implementation.

Consulted: 25 September 2026

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

[S186] PostgreSQL 17: Privileges

PostgreSQL Global Development Group

Role privileges and object ownership.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-priv.html

[S187] PostgreSQL 17: Row Security Policies

PostgreSQL Global Development Group

Row-policy semantics, default denial and privileged bypass; not executed in the SQLite labs.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/ddl-rowsecurity.html

[S188] Multi-Tenant Security Cheat Sheet

OWASP Foundation

Tenant-aware authorization and isolation across storage and caches.

Consulted: 25 September 2026

https://cheatsheetseries.owasp.org/cheatsheets/Multi_Tenant_Security_Cheat_Sheet.html

[S189] Secrets Management Cheat Sheet

OWASP Foundation

Secret inventory, lifecycle, rotation and avoiding leakage.

Consulted: 25 September 2026

https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html

[S190] PostgreSQL 17: Routine Vacuuming

PostgreSQL Global Development Group

Dead tuples, reuse of storage and cleanup; not proof of media sanitization.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/routine-vacuuming.html

[S191] PRAGMA secure_delete

SQLite project

Secure-delete configuration and limitations, including virtual tables and separate copies.

Consulted: 25 September 2026

https://www.sqlite.org/pragma.html#pragma_secure_delete

[S192] SP 800-88 Revision 2: Guidelines for Media Sanitization (September 2025)

NIST

Publication landing record and sanitization scope. Not a claim that a device was sanitized or the full publication independently audited.

Consulted: 25 September 2026

https://csrc.nist.gov/pubs/sp/800/88/r2/final

[S193] Object Model

OpenLineage project

Datasets, jobs, runs and metadata used to describe lineage.

Consulted: 25 September 2026

https://openlineage.io/docs/spec/object-model/

[S194] PostgreSQL 17: ALTER TABLE

PostgreSQL Global Development Group

Schema changes, constraint validation and lock behavior in the cited version.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-altertable.html

[S195] PostgreSQL 17: CREATE INDEX

PostgreSQL Global Development Group

Concurrent index creation restrictions and operational caveats.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/sql-createindex.html

[S196] ALTER TABLE

SQLite project

Supported alterations and table-rebuild procedure; version-sensitive documentation.

Consulted: 25 September 2026

https://www.sqlite.org/lang_altertable.html

[S197] Service Level Objectives

Google SRE

Indicators, objectives and measurement boundaries.

Consulted: 25 September 2026

https://sre.google/sre-book/service-level-objectives/

[S198] Signals

OpenTelemetry project

Roles of traces, metrics, logs and related telemetry signals.

Consulted: 25 September 2026

https://opentelemetry.io/docs/concepts/signals/

[S199] Handling Overload

Google SRE

Load, saturation and overload-control considerations.

Consulted: 25 September 2026

https://sre.google/sre-book/handling-overload/

[S200] PostgreSQL 17: The Cumulative Statistics System

PostgreSQL Global Development Group

Engine-specific statistics and observation context.

Consulted: 25 September 2026

https://www.postgresql.org/docs/17/monitoring-stats.html

[S201] Python 3.13: unittest

Python Software Foundation

Test discovery, assertions, subtests and skipped-test semantics.

Consulted: 25 September 2026

https://docs.python.org/3.13/library/unittest.html