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 G · Chapters 40–46 · First Edition

From Records to Insight

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 G

From Records to Insight

Build pipelines, analytical systems and search while preserving the meaning of the source data.

This volume contains Chapters 40–46 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 G

From Records to Insight

Build pipelines, analytical systems and search while preserving the meaning of the source data.

40

CHAPTER

Data Pipelines: Capture, Transform and Deliver

40.0 What this chapter gives you

1. A data pipeline carries information from one representation or system into another. It may copy, validate, reshape or summarise the records, but it must not silently change what they mean.
2. You will follow a frozen export of the shop's four agreed order lines into a small reporting store. The questions are practical: Which source was read? What was accepted? What was changed? Where does progress stop after a failure? Can the result be reproduced?
3. The companion exercise is a local, bounded batch import. The discussion of change capture uses published PostgreSQL and Debezium documentation; it does not claim that a connector, Kafka cluster or production pipeline was deployed.

40.1 Sources and contracts

40.1.1 PLAIN — in simple words

1. A pipeline needs an agreement about its input before it needs a schedule. That agreement describes what one record means, which fields identify it, which values are allowed and which changes can happen later.
2. "Sales data" is not a sufficient description. It could mean agreed orders, dispatched goods, payments or refunds. Those are different events, with different amounts and dates.
3. A receiving system should preserve enough context to explain each result. A total without its population, units, source boundary and transformation rule is difficult to verify, even when the arithmetic is correct.

40.1.2 PLAIN — a picture in your head

1. Mira sends Dev a sealed envelope containing copies of order lines. On the envelope she writes which register supplied them, the export date and the rule used to choose them.
2. Dev writes a second note describing how he grouped the copies. Together, the two notes explain both what arrived and what he did with it.
3. **Where the comparison breaks:** a digital checksum can bind a report to exact bytes, but it cannot prove who created those bytes or that the source claims are true. Authentication and business verification remain separate responsibilities.

40.1.3 PLAIN — a worked example

1. Our new teaching export, PIPE-01, contains the same four agreed lines introduced earlier. It is a frozen copy, not a claim about a live capture time. Its grain is one agreed order line, identified by `(order_id, line_no)`.
2. The quantity unit is individual items. Prices are integer paise in INR. Multiplying each quantity by its agreed unit price gives 15,100, 2,000, 7,550 and 4,000 paise.
3. The input contract therefore predicts four distinct line keys, six units, two orders and 28,650 paise. These are useful reconciliation checks, not evidence that all possible real orders were included.
4. The contract deliberately excludes payments, discounts, taxes, shipping and refunds. A report derived from PIPE-01 must not acquire a label such as "net cash collected" merely because that label sounds useful.

40.1.4 PLAIN — what is really happening inside

1. The producer serialises records according to a schema. The consumer decodes them, checks structural rules and then applies domain rules. These are different stages: valid JSON can still contain an invalid price.
2. An identifier for the schema or transformation helps the consumer select the right interpretation. A later field rename or unit change must not be inferred solely from a familiar-looking value.
3. Provenance records connect the input version, processing code, parameters and output version. This connection makes reproduction and investigation possible without pretending that provenance proves correctness.

40.1.5 TECHNICAL — the engineer's version

1. Define grain, identity, domain types, null semantics, ordering scope, update/delete representation and compatibility policy. A contract should also specify size bounds and what happens to malformed or conflicting input.
2. The pipeline manifest in this book records a byte digest and transformation version. The digest establishes content identity under the selected hash, not origin authenticity or semantic completeness. [S99]
3. Source-table changes and business events must not be conflated. Logical decoding exposes database changes; deciding that a change means "a customer paid" requires additional application semantics. [S166]

40.1.6 WORDS — remember these

1. **Data contract:** the agreement about incoming information — explicit structure, semantics, identity and compatibility rules for a data interface. **Provenance:** the recorded origin and processing history — information linking source entities, transformations and resulting data. **Transformation version:** which processing rule was used — an identifier binding output to a particular implementation and its parameters.

40.2 Batch extraction

40.2.1 PLAIN — in simple words

1. A batch is a bounded collection processed together. Its boundary may be a frozen file, a database snapshot or an explicitly closed interval of source events.

2. Reading a changing database in several pieces can produce a mixture that never existed at one moment. Each piece may be individually valid while the combination is misleading.
3. A successful extraction needs both complete coverage of its intended boundary and a consistent interpretation of that boundary. "The command finished" establishes neither by itself.

40.2.2 PLAIN — a picture in your head

1. Dev photographs the first half of a noticeboard. Mira then replaces several notices before he photographs the second half. Joining the photographs does not create a reliable picture of the board at one time.
2. Freezing the board during both photographs would solve that particular problem. Keeping a versioned copy of the board would solve it differently.
3. **Where the comparison breaks:** database snapshot visibility can avoid stopping all writers. The exact isolation and export mechanisms are engine-specific, and a file copied outside those mechanisms is not automatically consistent.

40.2.3 PLAIN — a worked example

1. Use a separate two-account teaching model, not the shop's financial records. Initially A contains 60 units and B contains 40, for a total of 100.
2. An extractor reads A as 60. Another operation transfers 10 from A to B. The extractor then reads B as 50 and reports 110.
3. The source still contains 100. The error is the inconsistent observation boundary, not an addition mistake. A snapshot that sees either the complete before-state or complete after-state reports 100.
4. PIPE-01 avoids this race by using a fixed byte sequence as its input. That makes the local exercise repeatable, but it does not establish that an arbitrary real export was originally captured consistently.

40.2.4 PLAIN — what is really happening inside

1. A database extractor issues queries under a particular transaction and isolation configuration. Different statements may see different committed versions unless the chosen mechanism supplies a suitable shared snapshot.
2. Pagination is another boundary. An offset into a changing result set can skip or repeat rows as earlier rows appear or disappear. A stable ordering key helps, but a key alone does not freeze the population.
3. A manifest should identify the selection predicate, ordering, snapshot or source cut, row count and any exclusions. When the source cannot provide a consistent cut, the limitation belongs in the result, not only in an operator's memory.

40.2.5 TECHNICAL — the engineer's version

1. PostgreSQL 17 Read Committed gives statement-level snapshots; stronger transaction-level snapshot behaviour requires an appropriate isolation choice and operational design. A multi-statement export must account for that distinction. [S66]
2. Incremental extraction based only on an increasing row ID can miss updates and deletions to older rows. A timestamp cursor also needs tie-breaking, precision, clock and late-write assumptions.
3. This book's immutable batch uses `(batch_id, input_digest, transform_version)` as declared identity. Reusing the batch ID with different input bytes is rejected, even when someone believes

the new bytes are equivalent. That is an original teaching policy, not a universal connector requirement.

40.2.6 WORDS — remember these

1. **Extraction boundary:** exactly which source state or records were selected — the snapshot, interval or immutable object defining a batch's coverage. **Consistent cut:** observations that fit one permitted source history — a boundary that avoids incompatible mixtures of concurrent states. **Incremental extraction:** reading changes since recorded progress — a method whose cursor must account for the source's actual change semantics.

40.3 Change capture

40.3.1 PLAIN — in simple words

1. Instead of repeatedly copying everything, a pipeline can follow changes: a row was inserted, updated or deleted. This is change data capture, usually shortened to CDC.
2. A new subscriber still needs a starting picture. Otherwise, it sees tomorrow's changes without knowing what existed today.
3. Joining that starting picture to the continuing changes is the difficult part. A gap loses information; an overlap can repeat information. Both need an explicit protocol rather than an approximate timestamp guess.

40.3.2 PLAIN — a picture in your head

1. Dev copies the entire catalogue while Mira keeps a numbered list of changes. The copy has a clear marker saying which changes are already reflected in it.
2. Dev then follows the list from the agreed boundary. If a page of the list is missing, he cannot safely claim that his copy is current.
3. **Where the comparison breaks:** database logs can describe physical or logical operations rather than human catalogue edits. Transaction boundaries, replica identity and retention rules determine what a real connector can reconstruct.

40.3.3 PLAIN — a worked example

1. In a separate model, a snapshot at sequence 100 says product X has quantity 5. Sequence 101 changes it to 3; sequence 102 deletes it.
2. Applying both changes after the snapshot leaves X absent. Starting at 102 and missing 101 happens to give the same final presence here, but would lose an intermediate change required by an audit-oriented consumer.
3. Starting at 103 misses the deletion and incorrectly retains X with quantity 5. A final-state check for this key now catches the gap, but a few spot checks cannot prove full coverage.
4. Replaying 101 after 102 must not recreate the old state. The consumer needs a suitable version/order policy, or an ordered replay mechanism that prevents this stale update from becoming current.

40.3.4 PLAIN — what is really happening inside

1. A connector obtains a starting snapshot and records an associated log position. It then consumes changes from a boundary coordinated with that snapshot.

2. Source logs are finite operational resources. A stalled consumer can require old log segments to remain available, increasing storage use or eventually invalidating the ability to resume.
3. A delete requires enough identity to identify the destination record. A partial update may require previous state or configuration-specific fields. The consumer must understand the actual event envelope rather than treating every message as a complete new row.

40.3.5 TECHNICAL — the engineer's version

1. Debezium's PostgreSQL connector documents a coordinated initial snapshot followed by log streaming. Snapshot modes, restart behaviour and captured fields are connector/version concerns, not properties of every pipeline called CDC. [S165]
2. PostgreSQL logical slots can resend recent changes after a crash because persisted progress may precede the last delivered position. Clients must avoid harmful repeated handling. A received message is not proof of a unique business effect. [S166]
3. PostgreSQL logical replication does not replicate schema DDL and sequence state in the same manner as table rows. Schema evolution and failover therefore require separate planning. Do not infer a complete database clone from a table-change feed. [S164]

40.3.6 WORDS — remember these

1. **Change data capture:** following source mutations — obtaining inserts, updates and deletions for downstream processing. **Snapshot-to-stream handover:** joining the initial copy to later changes — a protocol preventing gaps and managing overlap at a capture boundary. **Replication slot:** retained source progress for a consumer — a PostgreSQL mechanism whose log retention and restart behaviour require monitoring.

40.4 Transformations

40.4.1 PLAIN — in simple words

1. A transformation changes representation or derives a new value. It might convert units, choose columns, join descriptions or group lines into totals.
2. Some transformations lose detail. Once four lines become two order totals, the original line quantities cannot be recovered from the totals alone.
3. The processing rule must say what happens to missing, invalid and duplicate data. Silently turning all three into zero makes the output simpler by destroying distinctions the reader may need.

40.4.2 PLAIN — a picture in your head

1. Dev sorts four order slips into two envelopes and writes a total on each. The envelope totals are useful, but they are not substitutes for every detail on the slips.
2. If one slip is unreadable, writing zero on it is an invented correction. Keeping it for review preserves the uncertainty instead.
3. **Where the comparison breaks:** software can retain exact source references and repeat the calculation automatically. That helps only when the code, inputs and policy are recorded and the process does not depend on unrecorded outside state.

40.4.3 PLAIN — a worked example

1. Group PIPE-01 by order ID. O-1042 contributes 15,100 plus 2,000, giving 17,100 paise. O-1043 contributes 7,550 plus 4,000, giving 11,550.

2. The two totals sum to 28,650. Dividing by two orders gives 14,325 paise per order. Dividing by six items instead gives 4,775 paise per item. These answer different questions.
3. Joining the current notebook catalogue price of 8,250 into the calculation would produce the previously discussed hypothetical 30,750 total. That is not the historical agreed total and must not replace it.
4. Joining product tags before summing can repeat line values and produce 51,300. The transformation may be syntactically valid while changing the grain and therefore the answer.

40.4.4 PLAIN — what is really happening inside

1. A deterministic transformation gives the same result for the same complete inputs and rules. Looking up today's exchange rate or catalogue value creates another input, even when it is not written in the main file.
2. Filtering changes the population. Joins can remove rows or multiply them. Rounding can make summing rounded parts differ from rounding a final exact sum.
3. A useful transformation record states the output grain, keys, units, retained exclusions and expected invariants. Its test cases include counterexamples, not only a clean happy-path batch.

40.4.5 TECHNICAL — the engineer's version

1. Treat transformation code, configuration and reference data as versioned inputs. Reproducibility requires all dependencies affecting the result, not merely a script filename.
2. Relational joins and aggregation operate on the selected rows and multiplicities. `DISTINCT` is not a general repair for a wrong join: it can remove legitimate equal-valued facts. [S58] [S80]
3. The local reporting transformation uses integer arithmetic and original line identity. It does not infer a sale's financial, legal or physical completion from an order record. Those remain outside the teaching contract.

40.4.6 WORDS — remember these

1. **Deterministic transformation:** the same complete inputs yield the same output — a rule without uncontrolled time, randomness or external-state dependencies. **Lossy transformation:** some source detail is discarded — a mapping that cannot generally be inverted to recover the original records. **Output grain:** one resulting row means one what — the level at which a transformed record states a fact.

40.5 Delivery and checkpoints

40.5.1 PLAIN — in simple words

1. A checkpoint records where processing can safely resume. It must correspond to work actually preserved, not work merely started or displayed on a progress bar.
2. If the checkpoint moves ahead before the output is saved, a restart can skip missing output. If output is saved before progress is recorded, a restart can repeat it.
3. A correct design either commits output and progress together within a suitable boundary, or makes repeated delivery safe. The correct choice depends on which systems participate.

40.5.2 PLAIN — a picture in your head

1. Dev ticks a slip after filing it. Ticking first risks losing the slip after the tick. Filing first risks filing it again after an interruption.

2. A numbered filing system can recognise an already filed slip and check whether the repeated contents agree. That turns some repeats into harmless replays.
3. **Where the comparison breaks:** a database transaction can bind several local records atomically. Two unrelated remote systems do not become one atomic filing cabinet merely because one program calls them in sequence.

40.5.3 PLAIN — a worked example

1. Our local batch publisher begins a transaction, inserts the four reporting lines and records PIPE-01's identity in the same database. It commits only after all checks pass.
2. Inject an exception after the second insert. Rollback removes the new destination lines and the uncommitted batch record. The retry starts from the same frozen input, not from an invented "half complete" marker.
3. After a successful commit, submit PIPE-01 again with the same digest and transformation version. The stored result is returned without inserting four more lines or doubling 28,650 to 57,300.
4. Submit different bytes under PIPE-01. The publisher rejects the identity conflict. This does not overwrite earlier output, and it does not treat a matching batch label as permission to replace history.

40.5.4 PLAIN — what is really happening inside

1. Local atomic publication stores destination data and the receipt for that data within one transaction. Unique keys constrain repeated insertion, while explicit payload checks distinguish replay from conflict.
2. Remote destinations require their own acknowledgement and replay rules. A checkpoint in a source connector does not automatically prove that every downstream report has incorporated the same prefix.
3. Partial progress also needs a defined scope. A checkpoint for partition A says nothing about partition B unless a higher-level checkpoint records both and explains their relationship.

40.5.5 TECHNICAL — the engineer's version

1. Bind progress to the destination effect it claims. In the local exercise, the transaction owns both rows and batch receipt; no cross-system exactly-once guarantee is asserted. [S25] [S86]
2. A production streaming checkpoint may capture offsets, operator state and sink-commit information. End-to-end correctness depends on compatible source and sink protocols, not just a framework's internal recovery mechanism.
3. Retrying an external side effect requires stable request identity and a policy for unknown outcomes. The outbox/inbox construction from Chapter 26 applies when its local atomicity and consumer deduplication assumptions hold. [S125] [S126]

40.5.6 WORDS — remember these

1. **Checkpoint:** recorded safe progress — a recovery marker bound to preserved state and explicitly scoped effects. **Atomic publication:** output becomes accepted as one local change — committing records and their completion evidence within one transaction boundary. **Replay-safe delivery:** repetition does not duplicate the intended effect — handling a repeated identity consistently while detecting conflicting payloads.

40.6 Reconciliation and replay

40.6.1 PLAIN — in simple words

1. Reconciliation compares the source and result using checks chosen for the transformation. It asks whether the expected records and quantities survived, not merely whether a job reported success.
2. A matching total is useful but incomplete. Two wrong amounts can cancel each other. Missing one record and duplicating another can preserve the row count.
3. Replay rebuilds output from recorded inputs. It is a controlled experiment when the inputs and rules are fixed; it is a different computation when either changes.

40.6.2 PLAIN — a picture in your head

1. Mira checks that Dev filed four slips, then checks their identifiers and values. Counting envelopes alone would miss a slip placed twice and another omitted.
2. She can ask him to repeat the grouping from the sealed copies. If the totals change, they can inspect the procedure rather than arguing from memory.
3. **Where the comparison breaks:** digital equality can compare every included key and value, but it still cannot reveal records that the source never captured or an undisclosed exclusion outside the manifest.

40.6.3 PLAIN — a worked example

1. PIPE-01's report should have exactly the four source keys, quantities totalling six and agreed amounts totalling 28,650. Check each line's amount as well as the aggregate.
2. In a deliberately damaged copy, add 100 paise to one line total and subtract 100 from another. The grand total stays 28,650, but keyed comparison reveals both disagreements.
3. Rebuild the clean output with the same input digest and transformation version. Compare ordered key/value records or a canonical representation, not a hash of unspecified row order.
4. A proposed new report definition should receive a new transformation version and its own results. Keep a comparison explaining the difference rather than relabelling the old report as though it always meant the new thing.

40.6.4 PLAIN — what is really happening inside

1. Reconciliation combines coverage checks, uniqueness checks, value comparisons and domain invariants. The right combination follows the contract: a filtered report should reconcile its included and excluded populations separately.
2. A reproducible run stores parameters, versions and output evidence. This allows failures to be investigated without touching the original source records or silently changing the test expectations.
3. Operational monitoring tracks lag, rejected input, unavailable log history, replay conflicts and destination failures. A green schedule is not enough when useful data is steadily falling behind.

40.6.5 TECHNICAL — the engineer's version

1. Record source and destination watermarks with their meanings. A maximum observed source position is not necessarily a contiguous applied prefix, and offsets from different streams are not universally comparable integers.

2. Use keyed reconciliation to supplement aggregate checks. Canonical serialisation must specify field order, type representation, text encoding and row order before a digest can meaningfully compare runs. [S45] [S99]
3. The lab's failure injection and replay checks establish local behaviour for frozen teaching inputs. They do not test connector failover, source-log loss, unbounded streams, external delivery or every possible malformed-file attack.

40.6.6 WORDS — remember these

1. **Reconciliation:** comparing expected and observed information — a set of coverage, identity, value and invariant checks across representations. **Contiguous progress:** every required earlier position is included — a stronger condition than merely having seen a large position number. **Re-processing:** applying rules again to retained inputs — a controlled rebuild whose interpretation depends on input and transformation versions.

40.97 Practice and worked answers

1. **Question:** Why is a pipeline labelled "sales" ambiguous? **Answer:** Orders, dispatches, payments and refunds describe different facts. Specify which event or state is included before assigning the metric a name.
2. **Question:** An extractor reads 60 from A and later 50 from B after a 10-unit transfer. What failed? **Answer:** The observation boundary mixed before and after states. Addition is correct; the claimed population is not one consistent state.
3. **Question:** Can an increasing primary-key cursor capture all changes? **Answer:** Not by itself. Updates and deletions to existing smaller keys can be missed.
4. **Question:** Why does the batch receipt share a transaction with reporting rows? **Answer:** A receipt must not claim completion without the output, and a retry must recognise already committed output without duplicating it.
5. **Question:** How many rows and paise should a replay of PIPE-01 add? **Answer:** Zero new rows and zero new amount when the stored batch identity agrees. The retained four rows still total 28,650.
6. **Question:** Does a matching grand total prove record equality? **Answer:** No. Opposite 100-paise errors cancel. Compare complete keys and values as well as aggregates.
7. **Question:** Does a source connector's checkpoint prove a dashboard is current? **Answer:** No. Every downstream stage has its own applied progress and publication boundary.
8. **Question:** What should change when the transformation definition changes? **Answer:** Its version and derived result identity, with a recorded comparison and a deliberate replacement or coexistence policy.

40.98 Common wrong ideas

1. **Wrong:** A pipeline is just a scheduled copy. **Right:** It also carries contracts, progress, failure policy and evidence about meaning.
2. **Wrong:** A checksum authenticates the producer. **Right:** It identifies bytes under a hash; authenticity needs a separate trust mechanism.
3. **Wrong:** Every valid input record belongs in the report. **Right:** Eligibility follows the declared population and purpose.

4. **Wrong:** A snapshot followed by any recent log position is sufficient. **Right:** The boundary must prevent gaps and handle overlap.
5. **Wrong:** Repeating a job is harmless because it reads the same file. **Right:** Destination effects need replay safety too.
6. **Wrong:** CDC captures the full meaning of a business action. **Right:** Row changes require application interpretation.
7. **Wrong:** Matching counts and totals prove all records are correct. **Right:** Keyed differences can remain hidden.
8. **Wrong:** A successful local import proves production delivery guarantees. **Right:** It proves only the bounded behaviours actually exercised.

40.99 Chapter summary in 20 lines

1. A pipeline moves and transforms information under a contract.
2. Define source grain before writing extraction code.
3. Preserve units, identity and missing-value meaning.
4. A digest binds bytes, not truth or origin.
5. A batch needs an explicit observation boundary.
6. Changing sources can produce inconsistent multi-query exports.
7. An increasing ID alone does not capture every mutation.
8. CDC follows inserts, updates and deletions.
9. An initial snapshot must connect correctly to the change stream.
10. Log retention limits how a consumer can resume.
11. Redelivery is a normal recovery possibility.
12. Transformations can discard detail and change grain.
13. Historical prices are not today's catalogue prices.
14. Record all inputs that affect a derived answer.
15. Progress must correspond to preserved output.
16. One local transaction can bind output and its receipt.
17. Remote effects need additional delivery and replay rules.
18. Reconcile keys and values as well as counts and totals.
19. Reprocessing with new rules creates a new result version.
20. State exactly which pipeline guarantees the evidence demonstrates.

41

CHAPTER

Streams, Event Time and Late Arrivals

41.0 What this chapter gives you

1. A stream keeps bringing records while a question is being answered. There may be no natural final record telling the system that a day's information is complete.
2. You will separate the time an event claims to describe from the time a processor receives it. Then you will build windowed answers that can be revised when information arrives late, without counting a replay twice.
3. The chapter uses an original deterministic time-window model. Its explicit watermarks, lateness limits and output revisions are teaching policies, not an implementation or performance test of Apache Beam or another distributed stream processor.

41.1 Continuous records

41.1.1 PLAIN — in simple words

1. A stream is a continuing sequence of records. A till, delivery scanner or application can keep producing new information while yesterday's records are still being processed.
2. Continuous arrival does not guarantee continuous progress. A source can pause, a connection can fail, and a processor can fall behind while the producer keeps working.
3. Before processing a stream, define what one record represents and how repetitions, corrections and gaps are recognised. Those responsibilities do not disappear when a system is described as real-time.

41.1.2 PLAIN — a picture in your head

1. Mira receives numbered slips through a slot while she is counting the previous slips. There is no sign saying that the last slip has arrived forever.
2. She can still produce a useful interim answer, provided she says which slips it includes and what might change later.
3. **Where the comparison breaks:** a distributed stream can have several independently ordered inputs rather than one slot. A position in one input is not automatically a position in a single global sequence.

41.1.3 PLAIN — a worked example

1. Suppose source A has processed records through position 50 and source B through position 80. These positions belong to different sequences and cannot be added to mean "global position 130."
2. An honest progress record is a pair: A through 50 and B through 80, together with any required gap checks or source-generation identifiers.

3. If A stops sending, B may reach 500 while A remains at 50. A large maximum position does not establish that a report using both sources is complete.
4. A report can either wait, publish a provisional result with the lag disclosed, or follow an explicit incomplete-data policy. The choice changes the product's meaning, not just its speed.

41.1.4 PLAIN — what is really happening inside

1. A processor reads records, updates state and emits results. Durable progress and state recovery determine which work is repeated after a failure.
2. Partitioning permits parallel work, but each partition may advance independently. Coordination is needed when one answer depends on several partitions.
3. A bounded batch can be viewed as a finite collection. An unbounded stream needs continuing policies for time, state size, late input and result revision rather than assuming that all data will eventually fit in memory.

41.1.5 TECHNICAL — the engineer's version

1. Apache Beam distinguishes bounded and unbounded collections and provides windowing and triggers for grouping and emitting results over continuing input. The runtime details depend on the selected runner and connectors. [S167]
2. Progress is typically partition-scoped. Include stream identity, generation and offset semantics when designing recovery; a numeric offset alone can be ambiguous after recreation or migration.
3. The local exercises use finite sequences to expose streaming rules reproducibly. They do not claim that a finite successful run establishes unbounded operational stability or distributed fault tolerance.

41.1.6 WORDS — remember these

1. **Stream:** an ongoing supply of records — input processed as a continuing sequence rather than only as a completed finite batch. **Unbounded input:** no known final record — a collection whose completion is not generally available during normal operation. **Partition progress:** how far one input segment has advanced — a scoped position that must not be confused with universal completion.

41.2 Event and processing time

41.2.1 PLAIN — in simple words

1. Event time is the time attached to the event being described. Processing time is when a particular system handles the record.
2. They can differ because of offline devices, queues, retries, transmission delays or incorrect source clocks. A late arrival is not necessarily a late real-world event.
3. Choose which time answers the question. "What happened before noon?" and "What did the server learn before noon?" are both useful, but they are different reports.

41.2.2 PLAIN — a picture in your head

1. A delivery driver writes 10:05 on a paper receipt and returns to the shop at 12:20. The delivery time and the time Mira receives the receipt are different facts.
2. Filing the receipt under 12:20 may help explain office workload. Using it to claim the delivery happened at 12:20 would change the event's meaning.

3. **Where the comparison breaks:** a source timestamp may come from a poorly set or manipulated clock. Preserving it does not require treating it as unquestionably accurate.

41.2.3 PLAIN — a worked example

1. Consider a separate stream fixture with integer minutes from an agreed origin. E1 has event time 2 and value 100; E2 has event time 7 and value 200.
2. The processor receives both before it announces completion progress for minute 10. Later it receives E3, whose event time is 6 and value is 50.
3. By event time, all three belong to the first ten-minute interval. By arrival order, E3 follows the first published answer. Both descriptions can be stored without rewriting either fact.
4. These values are invented abstract amounts, not additional sales in the canonical shop ledger. The example must not increase the earlier agreed total of 28,650 paise.

41.2.4 PLAIN — what is really happening inside

1. Timestamp assignment may come from source fields, transport metadata or a processor's clock. The assignment rule is part of the pipeline contract.
2. Conversion must preserve the relevant time zone and precision. A local date without a zone cannot always be mapped to one instant, and a clock reading is not a causal ordering proof.
3. Implausible future timestamps can distort progress estimates if the system blindly uses the largest observed event time. Validating timestamps and recording clock uncertainty is therefore an operational concern, not only a presentation choice.

41.2.5 TECHNICAL — the engineer's version

1. Event-time semantics require a defined timestamp extractor and temporal domain. Processing-time semantics instead follow the executing system's clock and can change when the same input is replayed. [S167]
2. Preserve original event time, ingestion time and processing metadata when their distinct meanings matter. Do not overwrite an inconvenient source timestamp with arrival time and keep the old label.
3. Ordering by timestamp is not equivalent to happened-before. Clock error and concurrent events remain possible, so causal or per-entity ordering may require other identifiers or sequence information. [S41]

41.2.6 WORDS — remember these

1. **Event time:** when the described event is said to have happened — a timestamp assigned under the source and pipeline's semantic contract. **Processing time:** when a processor handles a record — time measured by the executing system, potentially different on replay. **Ingestion time:** when a system accepted the input — a separately useful observation that does not replace the original event time.

41.3 Windows

41.3.1 PLAIN — in simple words

1. A window gives a bounded group over which a stream can answer a question. It might mean each ten-minute interval or each period of activity separated by a long gap.

2. Window membership needs exact boundaries. Otherwise an event precisely at 10:00 can be counted twice or in neither interval.
3. Windowing and publication are separate. Deciding which group contains a record does not decide when the result should be shown or whether it may later change.

41.3.2 PLAIN — a picture in your head

1. Mira places time-stamped slips into trays labelled 09:00 up to, but not including, 09:10; then 09:10 up to, but not including, 09:20.
2. A slip at exactly 09:10 belongs to the second tray. The labels make the handover unambiguous.
3. **Where the comparison breaks:** sliding windows can deliberately place one event into several groups. Session windows can merge after a late event connects periods that previously looked separate.

41.3.3 PLAIN — a worked example

1. With fixed windows of width 10, event times 2, 6 and 7 belong to `[0, 10)`. Time 10 belongs to `[10, 20)`. The square bracket includes the start; the round bracket excludes the end.
2. For non-negative integer time t, the window start is `(t // 10) * 10`. At t = 27, integer division gives 2, so the window is `[20, 30)`.
3. With width 10 and a new sliding window every 5 minutes, time 7 belongs to both `[0, 10)` and `[5, 15)`. Summing the totals of overlapping windows would count some events more than once.
4. In a separate session rule, each event creates an interval `[t, t + 5)`, and overlapping intervals merge. Events at 1 and 4 form `[1, 9)`. An event at 12 forms `[12, 17)`. A late event at 8 bridges them into `[1, 17)`.

41.3.4 PLAIN — what is really happening inside

1. A window function assigns each record to one or more windows. Grouping usually also includes an entity or report key, so one branch's total is not accidentally combined with another's.
2. Fixed windows simplify alignment. Sliding windows answer overlapping recent-history questions. Session windows follow bursts of activity and require rules for merging state and revising previously emitted results.
3. Calendar windows need additional definitions. A local day is not always a fixed duration in regions with clock changes. For a business report, the relevant zone and calendar policy belong in the metric definition.

41.3.5 TECHNICAL — the engineer's version

1. Window assignment, trigger policy, accumulation mode and allowed lateness are distinct configuration dimensions. Matching only the window width does not make two streaming reports equivalent. [S167]
2. The local model supports fixed ten-unit windows and explicitly non-negative integer event times. Its session example is a separate interval-union calculation, not a claim to implement every streaming session-merge protocol.
3. Key the published result by report definition, entity scope and window boundary. A revision identifier can then distinguish successive answers to the same question without treating them as independent amounts to add.

41.3.6 WORDS — remember these

1. **Window:** a defined group of time-related records — a membership rule used to bound aggregation over a stream. **Sliding window:** overlapping intervals — windows whose width exceeds their step, allowing one event to contribute to multiple results. **Session window:** a group linked by activity gaps — a window that may merge when an arriving event connects earlier groups.

41.4 Watermarks

41.4.1 PLAIN — in simple words

1. A watermark is a progress statement about event time. It helps a processor decide that the ordinary waiting period for earlier records has passed.
2. It is not a magical observation of every device in the world. Its strength depends on the source and the assumptions used to produce it.
3. A record arriving behind the watermark is late under that policy. It may still describe a valid event, so "late" and "wrong" must not be used as synonyms.

41.4.2 PLAIN — a picture in your head

1. Mira announces that the normal courier has delivered slips through the morning interval. She publishes a provisional count while keeping a procedure for exceptional late envelopes.
2. If a branch is offline, she must decide whether to wait for it or disclose that the count can still change.
3. **Where the comparison breaks:** some sources can offer stronger progress guarantees than an estimated courier schedule. Other watermarks are heuristic estimates. Their meanings must be read from the actual source and framework contract.

41.4.3 PLAIN — a worked example

1. In our model, an external test driver advances a non-decreasing watermark. When it reaches 10, the model publishes revision 1 for `[0, 10)` using E1 and E2: 100 + 200 = 300.
2. Advancing the watermark is a declared input to the exercise. The model does not derive it from a network, a real clock or an unbounded source.
3. Suppose two required input partitions have watermarks 20 and 7. A conservative combined frontier is 7, because the second partition has not provided the same progress evidence as the first.
4. Marking the slow partition idle may permit progress under a chosen framework policy. When it resumes with older records, those records still need a late-data policy; idleness does not make them impossible.

41.4.4 PLAIN — what is really happening inside

1. Sources and operators maintain progress estimates or guarantees. Downstream stages combine them with outstanding work, timers and state dependencies.
2. A watermark should not move backward within the same progress history. Recovered or recreated sources also need identity and state rules so that an old position is not mistaken for current progress.
3. Monitoring should distinguish event-time lag, processing backlog and idle inputs. A processor may be fast at handling available records while a disconnected source makes its answer incomplete.

41.4.5 TECHNICAL — the engineer's version

1. Beam describes a watermark as its notion of when data for earlier event times can be expected to have arrived. Its guide separates this from trigger configuration and allowed late input. [S167]
2. A common estimate based on observed event time minus a delay is a policy, not a proof of source completeness. A forged future timestamp can advance such an estimate unsafely unless the system applies appropriate validation and source-specific safeguards.
3. Our integer model rejects a decreasing watermark and records late dispositions. Those checks establish internal consistency of the chosen model, not the truth of an externally supplied progress claim.

41.4.6 WORDS — remember these

1. **Watermark:** a statement of event-time progress — a source- and framework-dependent frontier used for window and timer decisions. **Late data:** a record behind the accepted progress frontier — input whose event time precedes the relevant watermark or closed-window boundary. **Idle input:** a source currently producing no records — a condition requiring a policy, not proof that earlier events can never return.

41.5 Late corrections

41.5.1 PLAIN — in simple words

1. Publishing quickly and waiting for every possible late record are competing goals. A system must choose how long to accept revisions and what to do after that limit.
2. A later result should say whether it replaces the earlier result or adds a difference to it. Mixing those meanings can double a report even when each emitted number is correct.
3. Repeated delivery of the same event is different from a new late event. Recognising the difference requires identity and payload checks, not only comparing timestamps.

41.5.2 PLAIN — a picture in your head

1. Mira writes "morning total: 300, revision 1." A late slip adds 50, so she writes "morning total: 350, revision 2."
2. Dev must replace the old total with 350. Adding the two displayed totals would produce 650, even though the actual included amounts sum to only 350.
3. **Where the comparison breaks:** systems can instead emit a +50 change. Both replacement and delta designs can work, but the consumer must know which protocol it receives and how repeats are handled.

41.5.3 PLAIN — a worked example

1. The model permits late updates while the watermark is below window end plus 5. For `[0, 10)`, the retention cutoff is therefore 15.
2. At watermark 11, E3 arrives with event time 6 and value 50. The model accepts it, changes the stored sum from 300 to 350 and publishes replacement revision 2.
3. Repeating E3 with the same identity and content creates no revision and no additional 50. Reusing E3 with value 70 is a conflict, not a correction. A genuine correction needs its own explicit identity and relation to the earlier event.

4. At watermark 15, a new E4 with event time 8 and value 25 is beyond the model's acceptance boundary. It is recorded as too late and does not silently change 350 to 375. This equality-at-cutoff rule is our declared policy.

41.5.4 PLAIN — what is really happening inside

1. A trigger decides when to emit. Accumulating output may contain all accepted data so far; discarding output may contain only a new pane's contribution. Consumers need the matching interpretation.
2. Revisions need stable window identity and ordered version handling. Receiving revision 2 before revision 1 must not allow the older replacement to overwrite the newer answer.
3. Records rejected by the streaming lateness policy may still enter a separately governed batch correction process. That process must preserve the difference between the earlier published view and a later restatement.

41.5.5 TECHNICAL — the engineer's version

1. Beam's trigger and accumulation settings affect when and how output panes are emitted. Allowed lateness provides a bounded opportunity to react to records after the watermark passes a window end. These settings must be coordinated with sink semantics. [S167]
2. Our publisher uses `(window_start, revision, replacement_total)` and a remembered event ID/payload mapping. Duplicate identity with identical content is a replay; conflicting content raises an error without mutating the accepted state.
3. The exercise retains final summaries and disposition evidence for inspection. It is not an unbounded deduplication store or a complete state-expiration implementation. Real retention must account for replay horizons and privacy requirements.

41.5.6 WORDS — remember these

1. **Trigger:** the rule deciding when to publish — a condition for emitting a window result or pane. **Replacement result:** a new complete answer for the same key — output that supersedes an earlier revision rather than adding to it. **Allowed lateness:** the accepted revision interval — a configured policy governing how long late window data may affect retained computation.

41.6 State, checkpoints and replay

41.6.1 PLAIN — in simple words

1. A streaming answer depends on remembered state: partial totals, seen identities, window boundaries and progress. Saving only the total can lose the information needed to resume safely.
2. A restart may repeat records. If the restored deduplication state is older than the restored total, a repeated event can be added again.
3. Recovery should restore a compatible combination of input progress, state and output publication evidence. Separate components must agree on what the checkpoint actually covers.

41.6.2 PLAIN — a picture in your head

1. Mira copies her total into a notebook but forgets to copy the list of slips already counted. After returning from a break, she cannot reliably distinguish counted slips from new ones.
2. Saving the total and the counted-slip list together avoids that particular mismatch. She also needs to know which displayed revision customers may already have seen.

3. **Where the comparison breaks:** distributed systems may checkpoint many operators and coordinate with transactional sinks. A single notebook illustrates the dependency but does not solve the distributed checkpoint protocol.

41.6.3 PLAIN — a worked example

1. After accepting E3, the local state contains sum 350, revision 2, seen identities E1/E2/E3 and watermark 11. Snapshot that complete model state.
2. Restore it into a fresh model and replay E3 unchanged. The result remains 350 at revision 2. A new valid E5 in the same window can still be accepted while the watermark remains below 15.
3. Now imagine restoring sum 350 but only seen identities E1/E2. E3 would look new and could incorrectly produce 400. This counterexample explains why state components must share a recovery boundary.
4. If an external reader already received revision 2 before a crash, resending revision 2 should be recognised as the same result. Saving internal state alone does not prevent a remote consumer from adding a replacement twice.

41.6.4 PLAIN — what is really happening inside

1. Checkpointing serialises the state required to continue the computation. Source replay then reconstructs work after that checkpoint under the same rules.
2. State schema evolution matters too. A newer program must either understand the older checkpoint format or use a reviewed migration/rebuild process.
3. End-to-end correctness includes the sink. A framework can restore its own state consistently while an ordinary external API repeats a side effect that was acknowledged ambiguously.

41.6.5 TECHNICAL — the engineer's version

1. The model snapshot includes its schema version, width, lateness policy, watermark, per-window state and accepted event identities. Restore validates the representation rather than executing arbitrary serialized objects.
2. Use bounded, explicit data encodings for untrusted or persisted state. Python pickle is not a safe interchange format for untrusted inputs because loading it can execute code. [S62]
3. The companion tests cover deterministic replay and revision handling on finite inputs. They do not exercise a distributed checkpoint coordinator, exactly-once external delivery, clock synchronization or recovery from physical power loss.

41.6.6 WORDS — remember these

1. **Operator state:** remembered information used by a computation — partial aggregates, identities, timers and metadata needed to process later input. **State checkpoint:** a recoverable state boundary — a saved representation coordinated with the progress and effects it claims. **Restatement:** a later version of an earlier report — a deliberate correction whose relationship to previously published results remains visible.

41.97 Practice and worked answers

1. **Question:** Does a record received at 12:20 necessarily describe a 12:20 event? **Answer:** No. Event and receipt times are distinct, and the source clock's reliability must also be considered.

2. **Question:** Where does event time 10 belong with fixed width 10? **Answer:** In `[10, 20)`, not `[0, 10)`, under the declared half-open convention.
3. **Question:** Why should overlapping sliding-window totals not simply be added? **Answer:** The same event may belong to several windows, producing deliberate overlap.
4. **Question:** The two input watermarks are 20 and 7. What is the conservative combined frontier? **Answer:** 7, when both inputs are required and no separate idleness policy changes the interpretation.
5. **Question:** What replaces revision 1's 300 after E3 adds 50? **Answer:** Revision 2's complete total 350. Adding replacement totals would be incorrect.
6. **Question:** What happens to E4 at watermark 15 in this model? **Answer:** It is recorded as too late for the first window and does not alter the published total.
7. **Question:** Why save seen event IDs together with totals? **Answer:** Restoring inconsistent versions can cause already included events to be counted again.
8. **Question:** Does a passing deterministic window test prove a source watermark is truthful? **Answer:** No. The test checks the model's response to supplied progress, not the external source's completeness.

41.98 Common wrong ideas

1. **Wrong:** Real-time means no delay. **Right:** Continuing systems still have queues, unavailable inputs and processing lag.
2. **Wrong:** Arrival time and event time are interchangeable. **Right:** They answer different questions.
3. **Wrong:** A watermark makes late records impossible. **Right:** Its meaning depends on the source's guarantee or estimate.
4. **Wrong:** Window membership determines publication timing. **Right:** Triggers and revision policy are separate choices.
5. **Wrong:** Every result pane is an additional amount. **Right:** Some panes replace earlier results.
6. **Wrong:** The same timestamp proves duplicate identity. **Right:** Different events can share a timestamp, and replays can arrive later.
7. **Wrong:** Keeping a final sum is enough for recovery. **Right:** Progress, identities and publication state may also be required.
8. **Wrong:** A framework's state recovery automatically makes an external effect exactly once. **Right:** The sink and retry protocol must support the intended guarantee.

41.99 Chapter summary in 20 lines

1. Streams can continue without a natural final record.
2. Define record identity and correction meaning before processing.
3. Progress is scoped to its source and partition.
4. Event time describes the event under a timestamp contract.
5. Processing time describes the processor's handling of a record.
6. Source clocks can be wrong or manipulated.
7. Windows assign records to bounded groups.
8. Half-open boundaries prevent accidental double membership at an edge.
9. Sliding windows deliberately overlap.

10. Session windows may merge after a bridging arrival.
11. Watermarks express event-time progress under stated assumptions.
12. Slow or idle inputs need an explicit policy.
13. Late does not necessarily mean invalid.
14. Triggers decide when results are emitted.
15. Consumers must distinguish replacements from deltas.
16. Stable identities separate replays from new events.
17. A lateness cutoff bounds revisions, not real-world truth.
18. Recover totals, identities and progress consistently.
19. External sinks have their own acknowledgement and replay limits.
20. Preserve the history of published answers and later restatements.

42

CHAPTER

Warehouses, Lakes and Analytical Tables

42.0 What this chapter gives you

1. The database that accepts one customer's order and the system that studies five years of orders solve different workloads. They may share data, but they need not share every storage layout, query pattern or operating boundary.
2. You will distinguish an analytical model from the files holding it, then connect facts, dimensions, catalogues and snapshots. The aim is to understand responsibilities rather than memorize product labels.
3. The examples retain the shop's four agreed lines and introduce separately labelled analytical fixtures. No warehouse service, object-storage account or Iceberg engine is deployed by the companion exercises.

42.1 Operational versus analytical work

42.1.1 PLAIN — in simple words

1. Operational work asks small questions while doing the business: Can this order be accepted? Which items are available? Has this request already succeeded?
2. Analytical work asks questions across many records: How did quantities vary by month? Which report definitions disagree? What changed after a process revision?
3. The same underlying facts can support both, but a large report should not accidentally prevent the shop from accepting orders. Sharing data creates a need to manage resource use and freshness expectations.

42.1.2 PLAIN — a picture in your head

1. Mira's counter register is arranged for finding and changing one order quickly. Her annual review uses copies spread across a large table so she can compare many months at once.
2. Doing the annual review directly on the busy counter would obstruct customers, even if every calculation were correct.
3. **Where the comparison breaks:** one database can support mixed workloads with suitable design. Separate operational and analytical systems are an architectural choice, not a law requiring every small shop to buy two products.

42.1.3 PLAIN — a worked example

1. Suppose the counter issues 100 short requests per second, while a monthly report scans 50 million historical lines. These are invented workload assumptions, not measurements of KedByte's systems.

2. The report's acceptable completion time might be minutes, while a counter request has a much shorter service objective. Running both without resource controls can make their interests conflict.
3. A reporting copy could isolate some work, but it introduces a delay between accepted orders and report availability. A label such as "data applied through 10:30" becomes part of the report's interpretation.
4. The canonical four-line exercise is much smaller. It does not justify a distributed warehouse. Its value is teaching how the responsibilities change as the workload and organization grow.

42.1.4 PLAIN — what is really happening inside

1. Operational designs often emphasize constrained updates and targeted access. Analytical designs often emphasize scanning selected columns, joining descriptive data and aggregating large populations.
2. Copying data separates some resource and failure boundaries, but adds capture, reconciliation and access-management duties. A second system is not a free performance improvement.
3. Materialized results can reduce repeated computation. Their usefulness depends on whether the refresh policy is compatible with the question's required freshness.

42.1.5 TECHNICAL — the engineer's version

1. OLTP and OLAP are workload categories, not universal product classes. Characterize transaction size, scan volume, concurrency, update patterns, latency targets and consistency requirements before choosing an architecture.
2. PostgreSQL materialized views store query results for later access and may be stale until refreshed. A materialized result is not automatically synchronized with every source change. [S169]
3. Isolation through replicas or separate stores must be evaluated against capture lag, query visibility and recovery procedures. A report's source cut is part of its correctness claim, not merely an operations statistic.

42.1.6 WORDS — remember these

1. **Operational workload:** queries and changes that run the business — work often focused on small, current, constrained transactions. **Analytical workload:** queries that study populations of records — work often involving scans, grouping and historical comparison. **Materialized view:** a stored query result — derived data whose refresh policy determines how current it is.

42.2 Warehouse modelling

42.2.1 PLAIN — in simple words

1. An analytical model organizes measurable facts and the descriptions used to group them. An order-line quantity is a fact; a product category is a description that might help group facts.
2. Start by saying what one fact row means. A row for an order line cannot be mixed casually with a row for an entire order or a daily stock snapshot.
3. Historical descriptions need a policy. A customer changing category today does not automatically mean that last year's report should use the new category.

42.2.2 PLAIN — a picture in your head

1. Mira places each order slip beside cards describing its product, branch and date. The slip holds the agreed quantity and price; the cards help arrange the slips for different questions.

2. If a descriptive card changes, she must decide whether old slips keep their earlier description or use today's classification.
3. **Where the comparison breaks:** a dimension join can multiply rows when several description versions match one fact. Correct analytical modelling requires enforced or checked uniqueness of the matching interpretation.

42.2.3 PLAIN — a worked example

1. Define the fact table's grain as one agreed order line. The four canonical lines retain quantities 2, 1, 1 and 2 and line amounts 15,100, 2,000, 7,550 and 4,000 paise.
2. Grouping by product gives notebook quantity 3 and amount 22,650, and pen quantity 3 and amount 6,000. These still reconcile to six items and 28,650 paise.
3. Introduce a separate customer C-DEMO whose category is Retail for times `[0, 100)` and Trade from 100 onward. A synthetic fact at time 90 matches Retail; a fact at 110 matches Trade under an event-time classification policy.
4. If the Retail interval accidentally ends at 120, the time-110 fact matches both versions. A join now duplicates it. The defect lies in overlapping validity intervals, not in the addition function.

42.2.4 PLAIN — what is really happening inside

1. A star-shaped model places a fact table near descriptive dimension tables. Keys connect the fact to one intended dimension interpretation.
2. Some dimensions overwrite attributes and answer questions using current descriptions. Others retain versions with validity boundaries, often called slowly changing dimensions. The selected history policy changes the answer.
3. Measures also have aggregation rules. Line amounts can be added within compatible currency and scope. A daily closing-stock value cannot be summed across days and called current stock, because each day is another observation of a state.

42.2.5 TECHNICAL — the engineer's version

1. Declare fact grain, dimensional keys, measurement units and additive scope. A measure may be additive across products but not across time or incompatible currencies; a ratio usually requires preserving its numerator and denominator.
2. In the versioned-dimension example, enforce or validate one matching interval for each fact's key and reference time. A surrogate dimension key identifies a version but does not prove that its business-time assignment is correct.
3. Relational joins preserve the multiplicities implied by their matching conditions. The model's correctness therefore includes dimension coverage and non-overlap checks, not only a foreign key to an existing dimension row. [S58] [S74]

42.2.6 WORDS — remember these

1. **Fact table:** measurable observations at a declared grain — analytical records whose keys and units define what may be aggregated. **Dimension:** descriptive context for grouping facts — attributes connected to facts under a current or historical interpretation. **Slowly changing dimension:** a policy for evolving descriptions — a modelling approach that may overwrite attributes or preserve dated versions.

42.3 Files and catalogues

42.3.1 PLAIN — in simple words

1. A collection of files does not automatically form a reliable table. A reader needs to know which files belong, what their fields mean and which versions are current.
2. A catalogue supplies organized metadata about tables and their locations or definitions. It is a map, not the data itself and not automatically an access-control system.
3. "Lake" commonly describes a storage approach built around collections of data files. "Warehouse" commonly emphasizes managed analytical tables and queries. Real architectures can combine these ideas, so labels alone do not settle capabilities.

42.3.2 PLAIN — a picture in your head

1. A storeroom contains boxes of order copies. The boxes have value, but a researcher cannot know which ones form the approved September collection without a catalogue.
2. A catalogue entry says which boxes to read and which schema explains their contents. It can also distinguish an old collection from a new one.
3. **Where the comparison breaks:** files can be uploaded concurrently, replaced or left orphaned after failed work. A directory listing is not necessarily the same thing as an atomic table publication protocol.

42.3.3 PLAIN — a worked example

1. Suppose a table version consists of immutable files F1 and F2. A writer prepares replacement files F3 and F4 but has not yet published the new table metadata.
2. A reader that scans every visible file can accidentally combine F1, F2, F3 and F4, duplicating facts. A reader using the committed manifest reads only the files named by its chosen table version.
3. An abandoned upload F5 should not become part of the table simply because it exists beside the other files. It needs a successful publication boundary or a cleanup decision.
4. These file labels are an original conceptual model. The exercise does not rely on claims about a particular cloud provider's directory-listing consistency or object-storage API.

42.3.4 PLAIN — what is really happening inside

1. Data files store encoded records. Table metadata identifies schema, partitioning and the files or manifests making up a table state. A catalogue helps locate and update that metadata.
2. Readers follow a selected committed metadata version rather than guessing from whatever objects happen to be visible. This separates preparation from publication.
3. Storage permissions, catalogue permissions and query-engine permissions may be different boundaries. A user denied access through a dashboard might still read raw files if the underlying storage is broadly exposed.

42.3.5 TECHNICAL — the engineer's version

1. Apache Iceberg specifies table metadata, snapshots and manifests above data-file formats. It distinguishes the logical table state from individual stored objects. [S168]
2. Apache Parquet specifies a columnar file representation with metadata and column chunks. A Parquet file alone does not provide a multi-file transaction manager or a universal business schema. [S170]

3. Evaluate catalogues for identity, atomic update support, concurrency handling, permissions and interoperability. An architectural label such as lakehouse is not itself evidence that each requirement has been implemented and tested.

42.3.6 WORDS — remember these

1. **Catalogue:** organized metadata about data assets — a service or representation locating tables and describing their definitions and versions. **Manifest:** an explicit inventory for a data version — metadata naming the files or entries that belong to a committed state. **Orphan file:** a stored object not referenced by an accepted table state — prepared or abandoned data requiring a deliberate retention or cleanup policy.

42.4 Table snapshots

42.4.1 PLAIN — in simple words

1. A table snapshot identifies one coherent version of an analytical table. A reader can continue using that version while a writer prepares another.
2. Publishing a new snapshot should not accidentally discard another writer's accepted work. The update needs a rule for detecting that its starting version has become stale.
3. Keeping older snapshots allows historical reading only while the required files and metadata still exist and access remains permitted. A snapshot name is not an eternal recovery guarantee.

42.4.2 PLAIN — a picture in your head

1. Mira publishes a numbered catalogue page listing the boxes for edition 10. Dev reads that page while she prepares edition 11 elsewhere.
2. Another assistant also starts from edition 10. Before publishing, both must check whether edition 10 is still the current starting point; otherwise the second publication could erase the first assistant's additions.
3. **Where the comparison breaks:** real table systems use storage and catalogue-specific atomic operations and conflict checks. A paper edition number explains the condition but does not implement distributed concurrency control.

42.4.3 PLAIN — a worked example

1. Our metadata model starts at version 10 with files `{F1, F2}`. Writer A prepares `{F1, F2, F3}` based on version 10 and publishes version 11.
2. Writer B prepares `{F1, F2, F4}`, also based on 10. An unconditional replacement would lose F3 from the current table view.
3. A compare-and-set publication checks that the current version still equals the expected base. B's expected 10 no longer matches 11, so the stale publication is rejected without changing the current manifest.
4. B can review the new state and prepare a compatible result. Whether appending F4 can be safely combined depends on the operation and conflict policy; not every failed transaction can be retried by blindly taking a set union.

42.4.4 PLAIN — what is really happening inside

1. Immutable data and metadata versions permit readers to hold a stable view while writers produce new objects. The accepted metadata pointer selects which version becomes current.

2. Field identity matters during schema changes. A field renamed from `unit_price` to `agreed_unit_price` should not be confused with a newly introduced field that happens to reuse an old name or position.
3. Expiration and cleanup remove historical states according to policy. A file still required by a retained snapshot cannot simply be deleted because the latest snapshot no longer uses it.

42.4.5 TECHNICAL — the engineer's version

1. Iceberg uses field IDs to preserve column identity across schema naming and ordering changes. Its metadata-commit operation must atomically replace the version on which a change was based; the catalogue-specific atomic mechanism is outside the core format specification. [S168]
2. The local compare-and-set manifest example tests a stale-writer condition only. It does not implement Iceberg's file-level conflict validation, delete files, catalogues, storage durability or full schema evolution.
3. A time-travel query is bounded by retained metadata and files, supported reader versions and permissions. Recovery, audit retention and deletion obligations require separate design decisions beyond retaining a snapshot list.

42.4.6 WORDS — remember these

1. **Table snapshot:** a coherent table version — metadata selecting the files and interpretation visible to a reader. **Atomic metadata publication:** one accepted version replaces its expected predecessor — a concurrency boundary preventing incompatible partial or stale updates. **Field ID:** stable column identity independent of a display name — metadata used to preserve meaning through supported schema changes.

42.5 Governance boundaries

42.5.1 PLAIN — in simple words

1. Copying data for analysis creates another place where it must be protected, explained and eventually retained or removed according to policy.
2. A clean dashboard does not prove that the raw files, temporary exports and query logs are equally controlled. Governance follows the information through all relevant representations.
3. Ownership should identify who defines a metric, who may change its meaning and who handles a disagreement. Otherwise two carefully built reports can become competing versions of the truth without an agreed resolution process.

42.5.2 PLAIN — a picture in your head

1. Mira locks the main register but leaves its photocopies on an open table. Protecting the original does not protect the copies.
2. She also needs labels explaining whether a report is provisional, corrected or approved for a particular purpose. A polished cover is not a governance state.
3. **Where the comparison breaks:** digital data can be copied into caches, notebooks and exported files automatically. A complete inventory needs technical discovery and process ownership, not only visible office shelves.

42.5.3 PLAIN — a worked example

1. An analyst needs monthly product quantities. The four canonical order lines are sufficient to derive quantities 3 and 3 for notebook and pen; customer names are unnecessary for this particular calculation.
2. Including every customer attribute in the export increases exposure without helping that stated question. A narrower approved dataset can reduce the information handled.
3. A report row should retain its definition and source version, so a later reviewer can trace 22,650 notebook paise to the three agreed notebook units rather than a current-price lookup.
4. If the underlying line data is later removed under policy, the system must state which aggregate evidence remains and whether detailed reconstruction is still possible. It must not claim reproducibility from inputs it no longer holds.

42.5.4 PLAIN — what is really happening inside

1. Governance combines metadata, access rules, retention decisions, ownership and evidence. No single catalogue field enforces all of these automatically.
2. Derived tables may preserve sensitive information even when direct identifiers are removed. Small groups, unique combinations and retained links can make a supposedly anonymous aggregate revealing.
3. Testing should include access through storage paths, query engines, extracts and recovery environments. Restoring an old snapshot may reintroduce information that has since been restricted or deleted unless the restore process accounts for that history.

42.5.5 TECHNICAL — the engineer's version

1. Treat lineage as a dependency graph from source versions through transformations to outputs. Record both the useful audit relationships and the access/retention implications of keeping those relationships.
2. Iceberg snapshot expiration describes removal of historical table references and eventual eligibility of unused files for cleanup. It is not a universal guarantee of erasure from every backup, export or consumer. [S168]
3. The chapter provides engineering boundaries, not legal retention periods or a certification of anonymization. Actual retention, exceptions and access decisions need the organization's applicable requirements and authorized review.

42.5.6 WORDS — remember these

1. **Data lineage:** how outputs depend on inputs — a recorded graph of source versions, transformations and derived assets. **Metric owner:** the accountable interpreter of a reported measure — a role responsible for definition changes, exclusions and disputes. **Data minimization:** limiting information to a justified purpose — a design practice reducing unnecessary collection, copying and exposure.

42.6 Selecting a design by workload

42.6.1 PLAIN — in simple words

1. Select a design by the questions, scale, update rate and operational responsibilities it must handle. A fashionable label is not a requirement.

2. A small consistent database with clear reports may be better suited to a modest task than a complicated collection of distributed services. A much larger workload may justify additional machinery.
3. Compare complete operating systems of work: ingestion, queries, corrections, access, recovery and cost. Fast query demonstrations alone leave many important responsibilities untested.

42.6.2 PLAIN — a picture in your head

1. Mira does not build a warehouse complex to store four slips. She starts with a system she can understand and maintain.
2. When she adds many branches and years of history, she measures which part becomes difficult before expanding the design.
3. **Where the comparison breaks:** software migrations can be costly even before physical scale becomes large. Compatibility, team skills and future requirements matter, but they should be recorded as assumptions rather than treated as certainty.

42.6.3 PLAIN — a worked example

1. Define three hypothetical requirements: refresh within 15 minutes, answer the monthly quantity report within 20 seconds under ten concurrent readers, and reproduce a selected earlier report from retained inputs.
2. Candidate A might use one database with controlled reporting queries. Candidate B might use a reporting replica. Candidate C might use captured files and a separate analytical engine.
3. For each candidate, test the same input, report definition and freshness condition. Include the cost of maintaining captures, updates, deletions and recovery, not only query elapsed time.
4. A candidate that answers in two seconds but omits the latest hour fails the stated freshness requirement. Another that meets both timing goals but cannot explain its input version fails the stated reproduction requirement.

42.6.4 PLAIN — what is really happening inside

1. Architecture selection is a constraint problem. Some designs reduce scan cost but complicate updates; others simplify transactions but make very large historical scans expensive.
2. Separate reversible decisions from difficult commitments. File format, catalogue semantics, key design and source-history retention can constrain later migration.
3. Record failure cases as part of evaluation. A system should be observed while a load is late, a writer conflicts, a source changes schema and a restore is required, not only while every component is healthy.

42.6.5 TECHNICAL — the engineer's version

1. A workload specification should include data volume and distribution, query mix, concurrency, update/delete rates, service objectives, source lag, retention and recovery requirements. Avoid treating row count as a complete capacity model.
2. File layout, table format, catalogue and query engine are separate components with separate compatibility and failure properties. Measure their combined behaviour rather than assuming a capability from a format's name. [S168] [S170]
3. The book's analytical examples validate selected arithmetic, model boundaries and snapshot predicates. They are not a benchmark ranking or a procurement recommendation for a particular managed service.

42.6.6 WORDS — remember these

1. **Workload specification:** a description of the actual work — volumes, distributions, query patterns and operating requirements used to evaluate a design. **Freshness objective:** how current a result must be — an explicit bound connecting publication to its required source state. **Architecture trade-off:** a benefit obtained with another cost or constraint — a choice evaluated against the complete task rather than one isolated metric.

42.97 Practice and worked answers

1. **Question:** Why separate operational and analytical workloads? **Answer:** They can compete for resources and have different response, history and freshness needs. Separation is a choice with additional copy-management costs.
2. **Question:** What is the canonical fact-table grain? **Answer:** One agreed order line, not one order, payment or daily stock snapshot.
3. **Question:** Why can a historical dimension join double a fact? **Answer:** Overlapping validity intervals can make several dimension versions match the same fact.
4. **Question:** Does a Parquet file provide table transactions by itself? **Answer:** No. Multi-file table state and publication require additional metadata and coordination.
5. **Question:** What should writer B do after its expected base version 10 no longer matches current version 11? **Answer:** Reject the stale publication, inspect the new state and retry only under a compatible conflict policy.
6. **Question:** Can an expired snapshot always be queried later? **Answer:** No. Its metadata or required files may no longer be retained or accessible.
7. **Question:** Does the monthly product-quantity report need customer names? **Answer:** Not for the stated calculation. Additional fields need a separate justified purpose.
8. **Question:** A query is fast but one hour stale against a 15-minute requirement. Does it pass? **Answer:** No. Speed does not substitute for freshness.

42.98 Common wrong ideas

1. **Wrong:** OLTP and OLAP must always use different products. **Right:** They are workload categories; architecture follows measured requirements.
2. **Wrong:** A warehouse fact table can mix grains freely. **Right:** Mixed grains produce ambiguous aggregation.
3. **Wrong:** Today's category must describe every historical fact. **Right:** Historical interpretation requires an explicit policy.
4. **Wrong:** Every file in a folder belongs to the current table. **Right:** Committed metadata defines accepted membership.
5. **Wrong:** A catalogue automatically protects raw storage. **Right:** Access boundaries must be checked separately.
6. **Wrong:** A snapshot identifier guarantees permanent recovery. **Right:** Recovery depends on retained files, metadata, compatibility and permission.
7. **Wrong:** An aggregate contains no sensitive information. **Right:** Small or linkable groups can still reveal information.
8. **Wrong:** The most elaborate architecture is the most complete solution. **Right:** Unnecessary components add responsibilities and failure modes.

42.99 Chapter summary in 20 lines

1. Operational and analytical work ask different kinds of questions.
2. Shared resources can make those workloads interfere.
3. Reporting copies introduce freshness and reconciliation duties.
4. Materialized results must have a refresh policy.
5. State one fact row's grain precisely.
6. Dimensions provide descriptive grouping context.
7. Historical dimension changes need an interpretation rule.
8. Overlapping dimension versions can multiply facts.
9. Measures have explicit additive scopes and units.
10. A collection of files is not automatically a coherent table.
11. Catalogues describe and locate data assets.
12. Manifests identify accepted file membership.
13. Snapshots give readers a selected table version.
14. Atomic metadata updates detect stale writers.
15. Stable field IDs separate identity from naming.
16. Snapshot retention bounds time travel and cleanup.
17. Governance follows raw and derived copies alike.
18. Minimize data to the justified analytical purpose.
19. Evaluate query speed together with freshness and recovery.
20. Choose machinery by workload evidence, not architectural fashion.

43

CHAPTER

Columnar Storage and Compression

43.0 What this chapter gives you

1. A report often needs a few fields from many records. Columnar storage arranges values so the engine can avoid reading unrelated fields and can exploit similarities among values of the same kind.
2. You will work through row and column layouts, dictionary and run-length encodings, compression blocks, predicate pushdown and conservative skipping. Every saving will be tied to an explicit workload and representation.
3. The companion byte-layout experiment is an original fixed-width format using Python's standard library. It is not a Parquet writer, a database benchmark or evidence that columnar storage always outperforms row storage.

43.1 Rows versus columns

43.1.1 PLAIN — in simple words

1. Row storage keeps the fields of one record close together. Column storage keeps values of one field close together across many records.
2. A row-oriented layout can suit fetching most fields of one order. A column-oriented layout can suit adding one amount field across millions of orders.
3. The benefit comes from avoiding unnecessary work, not from changing the answer. Both layouts must preserve the same record relationships and values.

43.1.2 PLAIN — a picture in your head

1. Mira can store complete order slips in a stack, or copy all quantities into one list and all amounts into another list with matching positions.
2. To add amounts, the second arrangement lets her read only the amount list. To inspect one complete order, she must gather corresponding values from several lists.
3. **Where the comparison breaks:** real columnar formats include row groups, metadata, null representation and nested structures. They are not simply separate text files with the same line numbers.

43.1.3 PLAIN — a worked example

1. Imagine 10 million records with 50 fixed-width fields of 8 bytes each. Ignoring all headers and compression, the data occupies $10{,}000{,}000 \times 50 \times 8 = 4{,}000{,}000{,}000$ bytes, or 4 decimal GB.
2. A query needing only two fields has 160,000,000 bytes of relevant field values. Ideal projection avoids 48 of 50 fields, or 96% of that raw field payload.

3. That is an opportunity, not a promise of 25 times faster execution. Metadata reads, filtering, decompression, computation, result handling and storage access patterns still contribute to elapsed time.
4. The local experiment uses 10,000 rows and four signed 64-bit fields. A complete row representation contains 320,000 raw bytes. The branch and amount columns together contain 160,000 bytes before metadata or compression.

43.1.4 PLAIN — what is really happening inside

1. A reader selects the fields required by the query and locates their chunks. Values from unneeded columns may remain unread.
2. The engine combines corresponding positions to reconstruct selected records or compute aggregates. Preserving row alignment is essential; independently sorting columns would destroy the original relationships.
3. Row groups divide a file into manageable horizontal pieces, within which columns are stored separately. This permits some parallelism and bounded reads without requiring one enormous chunk for each column.

43.1.5 TECHNICAL — the engineer's version

1. Parquet's file organization contains row groups and column chunks, with metadata describing their locations and interpretation. Projection can avoid unrelated column payloads when the reader and query support it. [S170]
2. A column layout is not inherently a transaction or update policy. Point updates, deletes, snapshots and concurrency may be implemented by an engine or table layer above the file format.
3. The laboratory counts bytes that its explicit access algorithm inspects and verifies equal query results across layouts. It does not measure operating-system read amplification, real storage-device traffic or a Parquet engine's performance.

43.1.6 WORDS — remember these

1. **Columnar storage:** values grouped by field — a layout that can reduce irrelevant reads for analytical projections. **Row group:** a horizontal subset stored in column pieces — a unit connecting corresponding column chunks within a columnar file. **Projection pruning:** avoiding unneeded fields — selecting only the columns required to compute the query result.

43.2 Encodings

43.2.1 PLAIN — in simple words

1. An encoding represents values in another form. It may save space by replacing repeated strings with short codes or by recording how values change rather than repeating their full representations.
2. The decoder must recover the same intended values. A smaller representation that silently changes a price or loses missing-value information is not an equivalent storage improvement.
3. Different patterns favour different encodings. Repeated values, small ranges and sorted sequences offer different opportunities; random-looking unique values may offer few.

43.2.2 PLAIN — a picture in your head

1. Instead of writing "north" five hundred times, Mira writes a small dictionary saying code 1 means north and then records the code on each relevant slip.
2. For a long uninterrupted run of code 1, she might write "1 repeated 500 times." That is another encoding layered on the repeated pattern.
3. **Where the comparison breaks:** digital encodings require precise widths, byte order, length fields and error handling. A verbal shortcut is not a complete interoperable file specification.

43.2.3 PLAIN — a worked example

1. A column contains 500 instances of `north`, 300 of `south` and 200 of `west`. The raw UTF-8 character payload is $500 \times 5 + 300 \times 5 + 200 \times 4 = 4{,}800$ bytes.
2. A dictionary's three strings need 14 character bytes, and 1,000 one-byte codes need 1,000 bytes. The combined 1,014 bytes exclude lengths, counts and other framing; they are not a claimed complete file size.
3. If the codes appear in three runs, a toy run-length format using one byte for the code and four for its count needs $3 \times 5 = 15$ run bytes. Alternating codes 1 and 2 for 1,000 values instead needs 5,000 run bytes, worse than the uncompressed codes.
4. A delta example stores 1,000 once, then changes +2, +2 and +3 to represent 1,000, 1,002, 1,004 and 1,007. The benefit depends on how those deltas and their metadata are actually encoded.

43.2.4 PLAIN — what is really happening inside

1. Dictionary encoding separates distinct values from their occurrences. Its usefulness decreases when most values are unique or when dictionary lookup and storage costs dominate.
2. Run-length encoding records consecutive repetitions. Sorting or clustering can improve runs, but changing record order must preserve the relationship among all fields.
3. Delta and bit-packing schemes exploit small differences or bounded ranges. The decoder needs the appropriate base, bit width and signedness rules; these are part of the format contract.

43.2.5 TECHNICAL — the engineer's version

1. Parquet specifies several encodings, including plain values, dictionary codes, run-length/bit-packing hybrids and delta representations. Their actual bitstream rules must be read from the specification, not inferred from the teaching examples. [S171]
2. The toy run codec validates code range and positive bounded counts before decoding. Resource limits matter: a tiny encoded input claiming billions of repetitions should not allocate unbounded memory.
3. Encoding and general-purpose compression are distinct stages. An encoding may reduce size directly or rearrange bytes so that a later compressor finds more repetition. Neither should be credited with the other's effect without measuring the pipeline.

43.2.6 WORDS — remember these

1. **Dictionary encoding:** replacing values with reusable codes — a representation containing a distinct-value dictionary and an occurrence-code sequence. **Run-length encoding:** storing repetitions as a value and count — an encoding effective when identical values are consecutive. **Delta encoding:** storing changes from earlier values — a representation whose savings depend on the size and regularity of differences.

43.3 Compression blocks

43.3.1 PLAIN — in simple words

1. Compression reduces a byte sequence using patterns that a decoder can reconstruct. Lossless compression must return the original bytes exactly.
2. Compression usually works within blocks or pages. To read one value, the reader may need to fetch and decompress a larger block containing it.
3. Larger blocks can provide more repeated context and fewer headers, but can also increase the minimum work for a small read. Block size is therefore a trade-off, not a universal "bigger is better" setting.

43.3.2 PLAIN — a picture in your head

1. Mira packs many slips into a tightly sealed bundle. The bundle occupies less shelf space, but extracting one slip may require opening and unpacking the whole bundle.
2. Several smaller bundles can make selective access easier, at the cost of more wrapping and sometimes less efficient packing.
3. **Where the comparison breaks:** software compression uses mathematical representations, not physical squeezing. Its CPU cost, memory use and exact framing depend on the codec and implementation.

43.3.3 PLAIN — a worked example

1. Suppose a raw 1 MB block compresses to 250 kB. Reading and decoding that block to obtain one 8-byte value may still require 250 kB of input and roughly 1 MB of decompressed output handling.
2. If a query needs every value in the block, that overhead may be worthwhile because three quarters of the stored payload was avoided. If it needs one scattered value from each of many blocks, the trade-off is different.
3. In the companion experiment, the same generated records are packed in row-major and column-major byte order, then compressed with the same standard-library zlib settings.
4. The output records actual encoded lengths and verifies exact decompression equality. It does not assume in advance which layout has the better compression ratio, and it does not infer application throughput from file size alone.

43.3.4 PLAIN — what is really happening inside

1. The compressor encodes patterns in a bounded input unit. The decoder reconstructs that unit before or while the reader interprets the values inside it.
2. Compressed length, uncompressed length and integrity metadata serve different purposes. A length field helps frame data but must not be trusted as permission for unlimited memory allocation.
3. A reader may decompress only selected pages when metadata allows it. Efficient access depends on the arrangement of blocks, the query and which metadata the writer actually supplied.

43.3.5 TECHNICAL — the engineer's version

1. Keep the stages explicit: logical values, value encoding, compression and file/container framing. Distinguish stored bytes, bytes read, bytes decompressed and values examined in a measurement.

2. A codec benchmark should state implementation, level, block size, data distribution and whether encoding/decoding time is included. Comparing different data or unequal settings confounds the result.
3. Parquet pages are the units whose encoded payloads may be compressed under the file's codec settings. The teaching zlib blob has no Parquet page headers, dictionary-page rules or compatible reader contract. [S170] [S171]

43.3.6 WORDS — remember these

1. **Lossless compression:** a smaller representation with exact recovery — byte encoding whose decoder reproduces the original input. **Compression block:** a unit compressed and decoded together — a boundary affecting storage efficiency and selective-read cost. **Read amplification:** reading more bytes than the requested logical data — additional work caused by layout, blocks, metadata or access patterns.

43.4 Predicate pushdown

43.4.1 PLAIN — in simple words

1. A predicate is a condition such as "branch equals 1" or "amount is at least 5,000." Pushdown means applying useful parts of that condition closer to the stored data.
2. This can avoid returning or decoding data that the rest of the query would immediately discard. It is helpful only when the lower layer understands the condition with compatible semantics.
3. Not every expression can be pushed down safely. A custom function, different text comparison or missing-value rule can change which rows qualify.

43.4.2 PLAIN — a picture in your head

1. Dev asks the storeroom clerk for only the boxes belonging to branch 1 rather than carrying every box to the office and sorting them there.
2. The clerk must use the same branch definition and labels. A convenient approximation can send the wrong boxes or omit needed ones.
3. **Where the comparison breaks:** engines may combine metadata pruning, value filtering and later verification. "Pushed down" does not necessarily mean the storage layer proves the entire query condition without reading values.

43.4.3 PLAIN — a worked example

1. The toy file contains four 8-byte fields per row: sequence, branch, quantity and amount. A query asks for the sum of amount where branch is 1.
2. A full-row scan examines 32 bytes per row, or 320,000 bytes for 10,000 rows. A column-oriented scan can inspect the branch and amount columns, totalling 160,000 raw bytes in this simple implementation.
3. A more selective implementation could first inspect branch values, then fetch amount values only for matching positions. Whether that reduces physical reads depends on block layout and the access API, so the simple lab does not claim it automatically.
4. Both query paths must return exactly the same sum for the same generated records. A smaller byte count with a different result is a correctness failure, not an optimization.

43.4.4 PLAIN — what is really happening inside

1. The query planner determines which conditions the storage reader can evaluate. The reader uses partition information, statistics or decoded values to reduce candidates.
2. Conditions left unresolved continue through the execution plan and are evaluated later. Correctness requires retaining all possible qualifying rows until the relevant condition is decided.
3. Filtering also changes downstream work. Fewer candidate rows can reduce joins, aggregation memory and network transfer, even when the initial file read saves little.

43.4.5 TECHNICAL — the engineer's version

1. Pushdown requires semantic compatibility across layers, including data types, null handling, collation and supported operators. A functionally similar expression is not necessarily equivalent for every input.
2. Distinguish partition pruning, row-group/page pruning and row-level filtering. They remove work at different granularities and rely on different metadata or decoded values. [S172]
3. Validate optimized output against a simpler reference query, including boundary values, missing fields and unusual distributions. An execution-plan label indicates a selected mechanism, not its correctness for an undocumented data contract.

43.4.6 WORDS — remember these

1. **Predicate:** a condition used to select records — an expression whose truth determines query eligibility under the relevant semantics. **Predicate pushdown:** evaluating selection closer to storage — moving supported filtering work earlier to reduce later processing. **Candidate row:** a record not yet ruled out — input retained until the necessary conditions can be evaluated correctly.

43.5 Statistics and skipping

43.5.1 PLAIN — in simple words

1. A block may carry a summary such as its smallest and largest value. A query can skip the block when that summary proves that no value can match.
2. The word "proves" matters. If the summary merely suggests that matching values are unlikely, skipping could lose valid results.
3. Statistics can be missing or imprecise. A safe reader then reads more data rather than inventing a stronger exclusion claim than the metadata supports.

43.5.2 PLAIN — a picture in your head

1. A box labelled "order numbers 100 through 199" cannot contain order 250 when the label is reliable and uses the same numbering system.
2. A box labelled "mostly old orders" does not justify skipping it while looking for a recent order. One exception would invalidate that shortcut.
3. **Where the comparison breaks:** binary formats have type-specific comparison and null rules. A text prefix bound, a floating-point special value or a missing statistic needs more care than a simple integer label.

43.5.3 PLAIN — a worked example

1. Three toy blocks have integer ranges `[0, 9]`, `[10, 19]` and `[20, 29]`. A query selects values from 12 through 16 inclusive.
2. The first block can be skipped because its maximum 9 is below 12. The third can be skipped because its minimum 20 is above 16. The middle remains a candidate and its actual values must be checked.
3. A range of `[10, 19]` does not prove that 12, 13, 14, 15 or 16 actually occurs. It only fails to rule them out. Candidate selection and confirmed matches are different stages.
4. If the middle block has no statistics, read it. If its bounds are corrupt, treating them as authoritative can lose rows; metadata integrity and validation remain part of the storage system's responsibility.

43.5.4 PLAIN — what is really happening inside

1. Writers compute summaries while creating files or pages. Readers compare the query bounds with those summaries before fetching unnecessary payloads.
2. Clustering similar values can make block ranges narrower and improve skipping. Randomly mixing every date into every block may leave each block spanning the whole report interval.
3. Null counts, bloom filters and additional indexes can support other decisions, but each has a defined scope. A bloom filter's possible match still requires verification; absence is useful only when the filter is valid and correctly interpreted.

43.5.5 TECHNICAL — the engineer's version

1. Parquet's optional page index separates value-bound information from page offsets so readers can skip suitable pages without first reading every page header. The index is not a general secondary B-tree index. [S172]
2. Conservative interval exclusion for an inclusive query `[low, high]` is `block_max < low` or `block_min > high`, assuming trustworthy comparable integer bounds. Equality at a boundary must not be excluded.
3. The lab tests boundary, missing-statistic and overlapping-range cases against direct value filtering. It does not validate every Parquet logical type, truncated string bound, NaN convention or encrypted-file metadata path.

43.5.6 WORDS — remember these

1. **Zone map:** summary bounds for a data region — metadata such as minimum and maximum used for conservative elimination. **Conservative pruning:** skipping only what cannot match — a rule that may retain extra candidates but must not discard valid results. **Data clustering:** placing similar values near one another — a layout choice that can tighten summaries and improve selective access.

43.6 CPU, memory and input-output trade-offs

43.6.1 PLAIN — in simple words

1. Saving storage reads may require more CPU work to decode values. Saving disk space may increase memory needed for dictionaries or decompressed blocks.
2. The best balance depends on what is scarce and what the query actually does. A CPU-bound report and a slow-storage scan can favour different settings.

3. Measure the entire question under a stated environment. A compression ratio, a byte count and a latency figure describe different aspects of the system.

43.6.2 PLAIN — a picture in your head

1. Mira can carry fewer tightly packed bundles from the storeroom, but Dev may spend longer unpacking them at his desk.
2. If walking is the slow part, tight packing helps. If unpacking already keeps Dev busy while the storeroom is next door, another packing method might be preferable.
3. **Where the comparison breaks:** real engines can overlap I/O and computation, use vectorized decoding and cache data. Adding isolated stage times does not always predict end-to-end elapsed time.

43.6.3 PLAIN — a worked example

1. Suppose reading raw input takes an assumed 8 seconds and processing takes 2 seconds without overlap. A compressed alternative takes 2 seconds to read and 5 seconds to decompress and process.
2. Under this deliberately serial model, the first path takes 10 seconds and the second 7. Compression saves time despite increasing CPU work.
3. Change only the raw-read time to 1 second on a much faster storage path. The uncompressed model now totals 3 seconds; the compressed path is not automatically preferable. Real measurements would need updated compressed-read time and overlap behaviour too.
4. The companion report therefore records generated row count, exact layout lengths, compressed lengths, round-trip checks and equal query results separately. It does not convert these into a production capacity claim.

43.6.4 PLAIN — what is really happening inside

1. Execution can process batches of values efficiently, reducing per-value dispatch and improving locality. The benefit depends on operators, representation and available hardware.
2. Large dictionaries, many concurrent scans and decompressed buffers can consume substantial memory. A query that fits alone may spill or slow under concurrent load.
3. Updates and small writes introduce another cost. An analytical file arrangement may require writing replacement files, maintaining delete information or compacting many small files before queries remain efficient.

43.6.5 TECHNICAL — the engineer's version

1. Report workload and measurement scope: cold or warm data, projected columns, selected fraction, ordering, compression settings, concurrency and result consumption. Avoid comparing an in-memory decoded array with a disk-backed compressed file as though layout were the only variable.
2. Distinguish logical payload, encoded size, compressed size, bytes transferred and CPU/memory use. A query can save one of these while increasing another.
3. The file-format and encoding specifications explain available mechanisms; they do not guarantee a workload-specific speedup. Verify correctness first, then measure the selected implementation under representative conditions. [S170] [S171] [S102]

43.6.6 WORDS — remember these

1. **Vectorized execution:** processing batches of values together — an implementation approach reducing overhead and exploiting suitable data layout. **Decode cost:** work to reconstruct usable values — CPU and memory effort introduced by an encoding or compression layer. **End-to-end measurement:** timing the complete defined operation — evidence that includes all stages required to produce and consume the intended result.

43.97 Practice and worked answers

1. **Question:** How much raw field payload is needed for two of fifty 8-byte columns across ten million rows? **Answer:** 160,000,000 bytes, excluding metadata and compression.
2. **Question:** Why must columns preserve row alignment? **Answer:** A quantity and price at corresponding positions must still belong to the same record. Independently sorting columns destroys that relationship.
3. **Question:** When does the toy run-length encoding become larger? **Answer:** Alternating codes produce one five-byte run per value, so 1,000 values become 5,000 run bytes instead of 1,000 one-byte codes.
4. **Question:** Does a 250 kB compressed block allow an 8-byte physical read for one value? **Answer:** Not generally. The reader may need the whole compressed block and substantial decoded data.
5. **Question:** Can a block with bounds 10 through 19 be skipped for a query 12 through 16? **Answer:** No. The bounds overlap the query, even though they do not prove an actual match.
6. **Question:** What should a safe reader do when useful statistics are absent? **Answer:** Retain the block as a candidate and inspect it rather than inventing an exclusion.
7. **Question:** Is a smaller compressed file necessarily faster to query? **Answer:** No. Decoding, memory, access granularity and the workload affect the outcome.
8. **Question:** What does the companion format establish about Parquet interoperability? **Answer:** Nothing. It is a labelled original byte-layout exercise, not a Parquet implementation.

43.98 Common wrong ideas

1. **Wrong:** Columnar storage changes the logical rows. **Right:** It changes representation while preserving their relationships.
2. **Wrong:** Projection savings directly predict equal latency savings. **Right:** Other stages still contribute to time.
3. **Wrong:** Dictionary encoding always compresses data. **Right:** High cardinality and metadata can reduce or reverse its benefit.
4. **Wrong:** More compression is always better. **Right:** Decode and access costs can dominate.
5. **Wrong:** Pushdown means the whole predicate was decided without reading values. **Right:** Some conditions only reduce candidates.
6. **Wrong:** An overlapping min/max range proves a match. **Right:** It only prevents safe exclusion.
7. **Wrong:** Missing statistics justify guessing. **Right:** Safe pruning becomes less aggressive.
8. **Wrong:** A toy byte experiment is a production engine benchmark. **Right:** Its evidence applies only to the stated representation and operations.

43.99 Chapter summary in 20 lines

1. Row layouts group a record’s fields together.

2. Column layouts group values of one field together.
3. Analytical projections can avoid unrelated columns.
4. Row alignment must survive every representation change.
5. Row groups bound columnar processing units.
6. Encodings exploit repeated or predictable value patterns.
7. Dictionary codes need their dictionary and framing.
8. Run-length encoding benefits consecutive repetitions.
9. Delta encoding benefits suitable differences between values.
10. Compression must be evaluated separately from value encoding.
11. Blocks determine the minimum decoding work for some reads.
12. Pushdown moves supported filtering closer to storage.
13. Semantics must agree across execution layers.
14. Metadata can rule out blocks conservatively.
15. Overlapping bounds identify candidates, not confirmed matches.
16. Missing statistics should reduce pruning, not correctness.
17. Clustering can improve the usefulness of block summaries.
18. CPU, memory and I/O costs can move in opposite directions.
19. Compare equal results under equal workload assumptions.
20. A format specification supplies mechanisms, not guaranteed speedups.

44

CHAPTER

Full-Text Search and Relevance

44.0 What this chapter gives you

1. Looking up an exact order ID and finding a useful document are different tasks. Text search must decide how to split language, which expressions match and how to order several plausible results.
2. You will build an inverted index by hand, distinguish word presence from phrases, inspect a ranking rule and evaluate results against explicit information needs. You will also see why a search result is not proof that the retrieved statement is correct.
3. The companion examples use a small invented English corpus and basic SQLite FTS5 where the runtime supports it. They do not claim multilingual search quality, a complete search service or the live KedByte website's search implementation.

44.1 Words and documents

44.1.1 PLAIN — in simple words

1. A document is the unit a search system returns or indexes: a page, chapter, product description or smaller passage. Choosing that unit changes what the search result means.
2. A query expresses an information need, but the words typed may only approximate it. Someone asking about a pen repair may not benefit from every product page containing the word pen.
3. Search begins by defining the searchable collection and who may see each document. A relevant private result is still inappropriate for a reader who lacks permission to access it.

44.1.2 PLAIN — a picture in your head

1. Mira labels six cards and lets Dev search their descriptions. Each result points to a particular card, not to an unexplained sentence detached from its source.
2. If a long manual is split into passages, each passage should still identify the manual, section and edition so the reader can recover context.
3. **Where the comparison breaks:** digital systems can index many overlapping fragments automatically. Without stable document and passage identity, updates and deletions can leave stale or duplicate search results.

44.1.3 PLAIN — a worked example

1. Our synthetic corpus contains D1 `blue pen`, D2 `blue notebook`, D3 `red pen`, D4 `pen repair guide`, D5 `notebook backup guide` and D6 `pen blue`.
2. A query for documents containing both blue and pen should return D1 and D6 under a simple word-presence rule. A query for the exact phrase `blue pen` should return only D1.

3. A person asking how to repair a pen has a different need. In this deliberately tiny labelled example, D4 is the relevant answer even though several other cards contain pen.
4. These cards are invented teaching text. They are not claims about actual products, repair procedures or the content of any published KedByte chapter.

44.1.4 PLAIN — what is really happening inside

1. The indexing pipeline chooses document boundaries, extracts text and attaches metadata such as title, language, source version and access scope.
2. Retrieval identifies candidates according to the query model. Ranking then orders those candidates using a selected scoring rule.
3. The display layer should expose enough source context to help the reader judge usefulness. Search snippets are shortcuts into documents, not independent verification of their claims.

44.1.5 TECHNICAL — the engineer's version

1. Full-text retrieval operates over a defined document collection and query representation. PostgreSQL separates document normalization into `tsvector` values and query representation into `tsquery` values. [S174]
2. Document identity, passage boundaries and versioning are application decisions. Record them so indexing updates can replace or remove the intended material without leaving ambiguous fragments.
3. Apply access restrictions to all exposed information, including result counts, titles and snippets where those reveal protected content. A later click-time authorization check does not undo an earlier disclosure in search results.

44.1.6 WORDS — remember these

1. **Document:** the searchable unit — an identified body of content, potentially a whole page or a bounded passage. **Information need:** what the reader is trying to learn — the task that a query attempts to express. **Candidate retrieval:** finding possible answers — selecting documents for further scoring and verification under a query model.

44.2 Tokenisation

44.2.1 PLAIN — in simple words

1. A tokenizer decides which pieces of text count as searchable units. Splitting on spaces is one possible rule, but it is not sufficient for every language or identifier.
2. Normalization decides which differences to ignore. Case, accents, punctuation and word endings can help or hinder matching depending on the task.
3. The query and documents must be interpreted compatibly. Searching a normalized query against differently normalized content can hide documents that a reader reasonably expects to find.

44.2.2 PLAIN — a picture in your head

1. Dev cuts each card into word labels before sorting them. If he treats `blue-pen` as one label in documents but two labels in queries, some matches disappear.
2. If he removes every mark from an order code, two distinct codes can become indistinguishable. A rule useful for ordinary prose may be unsafe for exact identifiers.

3. **Where the comparison breaks:** natural languages differ in writing systems, morphology and segmentation. One English classroom tokenizer is not a universal language processor.

44.2.3 PLAIN — a worked example

1. The toy tokenizer lowercases the six ASCII-English cards and extracts runs of letters or digits. `Blue pen` and `blue PEN` therefore produce the same two tokens.
2. `blue-pen` becomes blue and pen under this declared rule. An exact identifier search for `O-1042`, however, should use its own identifier-aware path rather than assume prose tokenization preserves the full key.
3. Removing accents might make some users' queries easier to match, but it can also collapse distinctions. A system should state whether it preserves or removes those differences and test the languages it actually serves.
4. A stemmer might map related English endings to a common form. That can increase matches, but it may also combine words whose meanings differ in a particular technical context.

44.2.4 PLAIN — what is really happening inside

1. Text extraction and Unicode normalization occur before or alongside tokenization. The pipeline then may apply case folding, stop-word removal, stemming or language-specific dictionaries.
2. Position information records where tokens occur. Removing tokens or changing token boundaries affects phrase matching and snippets, so these operations must follow the engine's documented conventions.
3. Language detection is itself fallible. Mixed-language documents, code examples and product identifiers may need different analyzers or retained original text for exact matching.

44.2.5 TECHNICAL — the engineer's version

1. SQLite FTS5 provides tokenizer choices such as `unicode61`, `ascii` and the Porter wrapper, with documented token-character and diacritic behaviour. The exercise uses simple English inputs and does not validate every tokenizer feature. [S173]
2. PostgreSQL text-search configurations connect parsers and dictionaries to lexeme normalization. Query construction functions differ in how they interpret operators, phrases and ordinary text. [S175]
3. Maintain analyzer/version metadata with the index. Changing tokenization without rebuilding or reconciling indexed content can mix incompatible representations and make result differences difficult to explain.

44.2.6 WORDS — remember these

1. **Tokenisation:** splitting text into searchable units — applying a specified segmentation rule to content and queries. **Lexeme:** a normalized searchable form — a dictionary or analyzer output that may combine several written word forms. **Analyzer:** the text interpretation pipeline — extraction, tokenization and normalization rules used for indexing and querying.

44.3 Inverted indexes

44.3.1 PLAIN — in simple words

1. An inverted index turns "document to words" into "word to documents." Instead of reading every card, the searcher opens the list for the requested word.

2. Several word lists can be combined. AND keeps documents appearing in every required list; OR keeps documents appearing in at least one.
3. Word membership alone cannot prove a phrase. For that, the index or a later check needs the words' order and positions.

44.3.2 PLAIN — a picture in your head

1. Mira makes a drawer for each word. Inside the blue drawer are references to cards D1, D2 and D6. Inside pen are D1, D3, D4 and D6.
2. Dev intersects the two lists to find D1 and D6. He then checks positions to distinguish `blue pen` from `pen blue`.
3. **Where the comparison breaks:** real posting lists are compressed and may include frequencies, positions or other metadata. Efficient query execution can skip portions rather than literally opening every reference in order.

44.3.3 PLAIN — a worked example

1. The blue posting set is `{D1, D2, D6}`. The pen posting set is `{D1, D3, D4, D6}`. Their intersection is `{D1, D6}` and their union contains five documents.
2. Using zero-based token positions, D1 has blue at 0 and pen at 1. D6 has pen at 0 and blue at 1. Requiring pen immediately after blue selects D1 only.
3. For the word notebook, the posting set is `{D2, D5}`. Intersecting notebook with pen gives an empty set in this corpus, a legitimate result rather than an index failure.
4. Deleting D1 requires removing its eligibility from the search result. A stale posting pointing to a deleted document must not be treated as proof that the document still exists or may be displayed.

44.3.4 PLAIN — what is really happening inside

1. Index construction collects token occurrences by document. Query evaluation merges or intersects the relevant postings and may verify phrases or other conditions against positions or source content.
2. Updates can create new index segments that are later merged. Readers need a consistent view of which document versions and deletion markers are effective.
3. Some index structures return possible matches that need rechecking. This is acceptable when the recheck removes false candidates before the final result, not when approximate candidates are presented as exact matches.

44.3.5 TECHNICAL — the engineer's version

1. PostgreSQL GIN text-search indexes store lexemes with matching locations; GiST uses signatures that can require rechecking false matches. The access method's exactness and stored metadata affect execution. [S184]
2. The toy index records token positions and compares its AND, OR and phrase results with direct scans of the same tokenized documents. It is not a compressed posting-list engine or a concurrent index implementation.
3. In FTS5 external-content configurations, applications must keep content and index updates consistent using the documented mechanisms. An index definition alone does not automatically repair a previously inconsistent external-content history. [S173]

44.3.6 WORDS — remember these

1. **Inverted index:** a word-to-document map — an access structure locating documents containing searchable terms. **Posting list:** occurrences associated with one term — document references, often accompanied by frequency or position information. **Positional index:** an index retaining term locations — metadata supporting phrases and proximity conditions beyond word presence.

44.4 Scoring and ranking

44.4.1 PLAIN — in simple words

1. A search score orders candidates according to a rule. It is not a probability that a document is true, and scores from different queries or engines may not be directly comparable.
2. Useful rules can consider how often a term occurs, how common it is across documents and whether it appears in a title or a long body.
3. Repetition should not always grow the score without limit. A page repeating pen a thousand times is not necessarily more useful than a clear repair guide using the word twice.

44.4.2 PLAIN — a picture in your head

1. Dev gives some weight to words that distinguish a card from the collection. A rare technical term can be more informative than a word printed on nearly every card.
2. He also notices where the match occurs. A title devoted to repair may be a stronger clue for a repair question than a passing reference in a long unrelated document.
3. **Where the comparison breaks:** relevance depends on the reader's task and context. A numerical formula can rank clues, but it cannot guarantee that the top document answers the actual question correctly.

44.4.3 PLAIN — a worked example

1. In our six-document corpus, pen appears in four documents and blue in three. A simple inverse-frequency weight `ln(N / df)` gives pen about 0.405 and blue about 0.693.
2. A presence-only score for the two query terms gives D1 and D6 about 1.099, D2 about 0.693, and D3/D4 about 0.405. This is a toy rule, not SQLite's BM25 output.
3. A term-frequency saturation factor `(k + 1) * f / (k + f)` with k = 1.2 gives 1 when frequency f is 1 and about 1.571 when f is 3. Tripling repetition therefore does not triple this contribution.
4. D1 and D6 still tie under the presence rule. A deterministic document-ID tie-break makes output repeatable, but does not turn the tie-break into a claim of greater relevance.

44.4.4 PLAIN — what is really happening inside

1. Ranking computes features over the candidate set and collection statistics. Document length, term frequency and field weighting can affect the result.
2. Changing the corpus can change scores even when a particular document stays unchanged, because collection statistics such as document frequency change.
3. Ranking parameters should be evaluated on representative queries and separate development/test sets. Adjusting a rule until it succeeds on one demonstration can overfit that demonstration.

44.4.5 TECHNICAL — the engineer's version

1. BM25 combines term weighting with frequency saturation and document-length normalization. The exact formula and parameter choices should be stated for the implementation being used. [S182]
2. SQLite FTS5 negates its BM25 score so numerically smaller values sort first for better matches under that implementation. A generic assumption that every search score should be sorted descending is therefore wrong. [S173]
3. Bound SQL values with parameters, but remember that a full-text query string may still have its own operators and resource costs. Define whether the interface accepts literal words or an advanced query language, and handle invalid expressions without exposing internal details.

44.4.6 WORDS — remember these

1. **Document frequency:** how many documents contain a term — a collection statistic used to distinguish common from rarer terms. **BM25:** a family of lexical ranking rules — scoring that combines term weighting, frequency saturation and document-length effects. **Tie-break:** a rule ordering equal-scored results — a deterministic convention that does not itself establish relevance.

44.5 Language and spelling

44.5.1 PLAIN — in simple words

1. Readers may use spelling variants, abbreviations or different words for the same concept. Search can help by recognizing selected alternatives, but every expansion can also introduce unwanted matches.
2. Language-specific processing matters. An English stemming rule should not be presented as a solution for Hindi, Japanese or mixed-language technical writing.
3. Exact identifiers deserve special treatment. Correcting a misspelled ordinary word can be helpful; silently changing a receipt ID can retrieve the wrong person's record.

44.5.2 PLAIN — a picture in your head

1. Mira knows that a customer asking for a writing instrument may be looking for a pen. She can suggest that connection while still allowing the customer to inspect the original query.
2. She would not silently change order O-1042 to O-1043 because the second card happened to be easier to find.
3. **Where the comparison breaks:** automatic synonym and spelling systems cannot rely on human situational understanding. They need bounded rules, confidence policies and evaluation across the intended users' language patterns.

44.5.3 PLAIN — a worked example

1. A query `notebok` differs from notebook by one missing o. An edit-distance suggestion can propose notebook, but the interface should distinguish the suggestion from an exact match.
2. Expanding notebook to computer might help a device-shopping query while damaging a stationery query. The corpus and task determine whether that synonym is useful.
3. In our corpus, `notebook backup guide` is intentionally ambiguous: notebook could refer to a computer or written notes. Word overlap does not resolve the intended sense by itself.

4. A bilingual book reader should evaluate actual language examples and technical terms. The small English lab is evidence about its declared tokens only, not proof that every language's segmentation and spelling needs are covered.

44.5.4 PLAIN — what is really happening inside

1. Query expansion can add synonyms, spelling candidates or related forms before retrieval. This often increases the candidate set and can reduce precision while improving recall.
2. The expansion history should be observable when it changes the meaning of the query. A reader can then distinguish what they typed from what the system searched.
3. Search snippets also need safe rendering. Source text can contain markup or code; highlighting must not turn untrusted content into executable browser instructions.

44.5.5 TECHNICAL — the engineer's version

1. PostgreSQL's text-search configurations and dictionaries are language- and task-dependent. Normalization choices are not interchangeable, and appropriate configuration must accompany stored and queried text. [S174] [S175]
2. Highlighting functions can return text containing markup or source characters that need safe output handling. Treat a snippet as untrusted content and use a deliberate escaping/rendering strategy rather than inserting it blindly as HTML. [S175]
3. Keep exact-key retrieval separate from fuzzy natural-language search when identity is consequential. A suggestion is not authorization to replace the requested key or to expose nearby private records.

44.5.6 WORDS — remember these

1. **Query expansion:** adding related search terms — a recall-oriented technique whose extra candidates can change precision and meaning. **Edit distance:** the number of selected character edits between strings — a spelling similarity measure, not a proof of intended identity. **Snippet:** a displayed excerpt from a result — contextual text that requires source attribution and safe rendering.

44.6 Evaluating search quality

44.6.1 PLAIN — in simple words

1. Search quality must be checked against actual information needs. Finding a word is not the same as helping the reader answer a question.
2. Precision asks how many returned results are relevant. Recall asks how much of the known relevant material was returned. The labels and the chosen result cutoff matter.
3. A useful evaluation includes queries with no answer, rare terms, spelling problems and access restrictions. A few successful showcase queries do not establish general quality.

44.6.2 PLAIN — a picture in your head

1. Mira gives Dev a list of questions and independently marks which cards help answer each one. She then compares his results with those judgements.
2. If Dev returns every card for every question, he finds all relevant cards but also burdens the reader with irrelevant material.

3. **Where the comparison breaks:** relevance judgements can be incomplete or disputed. Evaluation should preserve who labelled what, the intended task and the uncertainty in the labels rather than treating them as infallible ground truth.

44.6.3 PLAIN — a worked example

1. For the labelled query "how to repair a pen," suppose only D4 is considered relevant. A hypothetical result order is D1, D3, D4.
2. Precision at three is 1 relevant result divided by 3 returned, or one third. Recall at three is 1 found divided by 1 known relevant, or 1.
3. The first useful result is at rank 3, so reciprocal rank is 1/3. Moving D4 first improves this navigation measure without changing the three-result set's precision or recall.
4. If there are no known relevant documents for another query, recall's denominator is zero. Record the chosen evaluation convention or report it as undefined; do not silently count an arbitrary 100% success.

44.6.4 PLAIN — what is really happening inside

1. Evaluation stores a corpus version, queries, relevance judgements, retrieval configuration and result lists. Repeatability requires all of those, not only the scoring formula.
2. Candidate generation and ranking can fail separately. If a useful document never enters the candidate set, a perfect reranker cannot recover it from nothing.
3. Operational quality includes response time, index freshness, deletion propagation and authorization. A relevant answer drawn from an outdated or unauthorized source can still fail the product's requirements.

44.6.5 TECHNICAL — the engineer's version

1. Precision and recall compare retrieved documents with judged relevance; they focus on useful retrieval rather than accuracy dominated by a large irrelevant population. [S181]
2. Use stable tie-breaking and avoid duplicate document IDs when computing set-based metrics. State whether the unit is a whole document, passage or source, because overlapping passages can inflate apparent coverage.
3. The companion checks establish posting-set correctness, phrase behaviour, selected FTS5 queries and metric arithmetic on a tiny labelled corpus. Independent user evaluation and multilingual, accessibility and production-load testing remain separate tasks.

44.6.6 WORDS — remember these

1. **Precision:** how much of the returned material is relevant — relevant retrieved items divided by retrieved items under a stated cutoff and unit. **Recall:** how much relevant material was found — relevant retrieved items divided by the known relevant population. **Reciprocal rank:** a measure of how soon the first useful answer appears — one divided by its rank, with an explicit no-answer convention.

44.97 Practice and worked answers

1. **Question:** Which cards contain both blue and pen? **Answer:** D1 and D6. Only D1 contains the phrase blue immediately followed by pen.

2. **Question:** Why does tokenization belong in the index's version record? **Answer:** A changed analyzer can alter boundaries and matches even when the source text is unchanged.
3. **Question:** What does a posting list contain? **Answer:** References to documents containing a term, potentially with frequencies and positions.
4. **Question:** Why is a search score not a truth probability? **Answer:** It measures the selected ranking features, not the correctness of every statement in a document.
5. **Question:** Which direction sorts SQLite FTS5 BM25 better matches first? **Answer:** Ascending, because that implementation returns numerically smaller scores for better matches.
6. **Question:** May spelling correction silently replace an exact order ID? **Answer:** No. A suggested nearby identifier is not the same request and may expose unrelated information.
7. **Question:** What are precision and recall for one relevant result among three returned when only one relevant document is known? **Answer:** Precision 1/3 and recall 1.
8. **Question:** Can ranking recover a useful document omitted by candidate retrieval? **Answer:** Not unless another retrieval stage introduces it. Ranking only orders what it receives.

44.98 Common wrong ideas

1. **Wrong:** A word match proves a useful answer. **Right:** Relevance depends on the information need and context.
2. **Wrong:** Space splitting works for every language. **Right:** Tokenization is language- and task-dependent.
3. **Wrong:** AND word matching proves an exact phrase. **Right:** Phrase matching needs order and position information.
4. **Wrong:** More repetitions always mean a better result. **Right:** Ranking rules often limit repetition's contribution.
5. **Wrong:** Every score should be sorted descending. **Right:** Read the implementation's score convention.
6. **Wrong:** Fuzzy search is appropriate for every identifier. **Right:** Exact identity needs a separate contract.
7. **Wrong:** Returning everything is excellent search because recall is high. **Right:** Precision and reader effort also matter.
8. **Wrong:** Search-result permission can wait until the click. **Right:** Titles, counts and snippets can already disclose protected information.

44.99 Chapter summary in 20 lines

1. Choose the document unit and collection boundary.
2. A query approximates a reader's information need.
3. Retrieval and ranking are separate stages.
4. Tokenization defines the searchable units.
5. Normalization changes which distinctions are retained.
6. Query and document analyzers must be compatible.
7. Inverted indexes map terms to documents.
8. Posting intersections implement simple AND conditions.
9. Phrase matching requires order and positions.

10. Index updates must track document versions and deletions.
11. Ranking scores reflect a selected rule, not truth.
12. Rare terms can contribute different evidence from common ones.
13. Frequency saturation limits repeated-word influence.
14. Score direction and tie-breaking are implementation details.
15. Language and spelling expansion need task-specific evaluation.
16. Exact identifiers should not be silently fuzzy-matched.
17. Snippets require context, permission and safe rendering.
18. Precision and recall have different denominators.
19. Candidate omissions cannot be repaired by ordering alone.
20. Evaluate usefulness, freshness, access and operational behaviour together.

45

CHAPTER

Vector Search: Similarity Is Not Truth

45.0 What this chapter gives you

1. Vector search represents items as lists of numbers and retrieves items that are close under a chosen mathematical rule. It can find useful similarities even when the exact query words do not appear in the result.
2. You will calculate distances, compare exact and approximate retrieval, inspect index trade-offs and test permission filters. The central boundary is simple: closeness in a representation is not proof of factual correctness, identity or authorization.
3. The companion exercises use small explicit vectors and direct Python arithmetic. They do not train an embedding model, implement HNSW or run Faiss, and they do not claim production semantic-search quality.

45.1 Representing items as vectors

45.1.1 PLAIN — in simple words

1. A vector is an ordered list of numbers. For search, a procedure turns each document, image or other item into such a list so similar items may occupy nearby positions.
2. The numbers gain meaning from the procedure that created them. Two lists of the same length are not automatically comparable if they came from different models or preprocessing rules.
3. A vector is a representation, not the original item. Keep a reliable link to the source and its version so a result can be inspected rather than trusted because a number looks impressive.

45.1.2 PLAIN — a picture in your head

1. Mira makes a map where cards about similar topics tend to sit near one another. A new question is placed on the same map, and she inspects nearby cards.
2. The map helps find candidates, but a nearby card might be outdated, misleading or about the wrong sense of a word.
3. **Where the comparison breaks:** learned vectors usually have many dimensions that are not individually labelled like streets. A coordinate does not necessarily correspond to a human-readable topic, and the geometry depends on the training objective.

45.1.3 PLAIN — a worked example

1. Begin with an invented two-dimensional representation rather than a trained model. Let the query be Q = (1, 0), and items A = (1, 0), B = (0.8, 0.6), C = (0, 1), D = (-1, 0) and E = (2, 0).
2. These coordinates are teaching data. No assertion is made that a real document about pens naturally receives any of these numbers.

3. A search algorithm can rank the items once a metric is chosen. It cannot decide whether A's underlying text is correct merely because A's vector equals the query vector.
4. If a different encoder maps the same source into another coordinate system, combining its output with the old index without compatibility checks can make the ranking meaningless.

45.1.4 PLAIN — what is really happening inside

1. An embedding pipeline extracts and preprocesses content, runs an encoder and stores the resulting vector with source identity and metadata.
2. Query encoding must use a compatible procedure. Some systems use different query and document encoders trained to work together; compatibility is therefore more specific than simply using identical software names.
3. Content updates, chunking changes or encoder revisions can require re-embedding. An index containing mixed generations needs an explicit compatibility policy and evaluation rather than accidental coexistence.

45.1.5 TECHNICAL — the engineer's version

1. Store encoder identity/version, vector dimension, numerical type, normalization policy, source revision and chunk boundaries. These attributes are part of the retrieval contract.
2. Validate dimensions and finite numeric values before distance computation. NaN, infinity or a zero vector can invalidate a selected comparison, especially cosine normalization.
3. Faiss exposes several index and metric choices rather than one universal meaning of vector proximity. Its documented metric conventions must be matched to the representation and task. [S176] [S177]

45.1.6 WORDS — remember these

1. **Embedding:** an item's numerical representation — an ordered vector produced by a defined encoder and preprocessing pipeline. **Embedding space:** the coordinate system used for comparison — a representation whose geometry depends on its model and training objective. **Representation compatibility:** vectors can be compared meaningfully — agreement between encoder, dimension, normalization and metric contracts.

45.2 Distance and similarity

45.2.1 PLAIN — in simple words

1. Euclidean distance measures straight-line separation. Inner product multiplies corresponding coordinates and adds the results. Cosine similarity compares direction after accounting for vector length.
2. These rules can rank the same items differently. Choosing one is part of defining what "similar" means for the application.
3. A smaller distance is usually better under a distance rule, while a larger similarity is usually better under a similarity rule. Always check the actual API's returned quantity and direction.

45.2.2 PLAIN — a picture in your head

1. Imagine arrows from the centre of a map. Two arrows can point in the same direction but have different lengths.

2. Cosine treats direction as important and ignores that length difference. Euclidean distance still sees the separation between the arrow tips.
3. **Where the comparison breaks:** the choice of geometry is not a universal model of human meaning. A trained representation may rely on length, direction or another scoring rule, so changing the metric can damage retrieval.

45.2.3 PLAIN — a worked example

1. For Q = (1, 0) and B = (0.8, 0.6), squared Euclidean distance is `(1 - 0.8)^2 + (0 - 0.6)^2 = 0.4`. The distance itself is the square root, about 0.632.
2. The squared distances from Q to A, B, E, C and D are respectively 0, 0.4, 1, 2 and 4. Exact Euclidean ranking therefore places them in that order, with a defined tie-break if needed.
3. Inner products with Q are A = 1, B = 0.8, C = 0, D = -1 and E = 2. Maximum-inner-product search prefers E to A in this example.
4. Cosine similarities are A = 1, B = 0.8, C = 0, D = -1 and E = 1. A and E point in the same direction despite their different lengths. A zero vector has no defined cosine direction and is rejected by our model.

45.2.4 PLAIN — what is really happening inside

1. Distance evaluation performs arithmetic across dimensions. Exact search repeats this comparison for every eligible stored vector unless another exact structure can safely eliminate candidates.
2. Squared Euclidean distance preserves Euclidean ordering because the square root is monotonic for non-negative inputs. It does not preserve the numerical unit of distance itself.
3. Normalizing vectors changes their length to one. For unit vectors, squared Euclidean distance equals `2 - 2 × inner_product`, connecting cosine and Euclidean rankings under those assumptions.

45.2.5 TECHNICAL — the engineer's version

1. Faiss reports squared L2 distance for its L2 metric. Its inner-product search is not automatically cosine similarity; cosine comparison requires suitable normalization of both queries and stored vectors. [S176]
2. The local functions check equal dimensions, finite values and nonzero norms where required. They use deterministic ID tie-breaking so repeated results can be compared without interpreting tie order as evidence of relevance.
3. A score threshold must be evaluated for the chosen encoder, metric and task. A value such as 0.8 has no universal meaning of "80% factually correct" or "the same person."

45.2.6 WORDS — remember these

1. **Euclidean distance:** straight-line separation — the square root of summed squared coordinate differences. **Inner product:** coordinate products added together — a similarity score affected by stored-vector magnitudes as well as direction. **Cosine similarity:** directional agreement — inner product divided by the product of nonzero vector norms.

45.3 Exact and approximate search

45.3.1 PLAIN — in simple words

1. Exact nearest-neighbour search returns the closest eligible items under the declared metric. It says nothing about whether that metric captures the reader's real need perfectly.
2. Approximate search saves work by examining a selected subset or a compressed representation. It can miss a mathematically nearer item.
3. Approximation is useful when its measured cost-quality trade-off fits the application. Calling it approximate should lead to evaluation, not to assuming that errors are either impossible or harmless.

45.3.2 PLAIN — a picture in your head

1. Mira can measure the distance from a question to every card on the map, or search only promising neighbourhoods first.
2. The neighbourhood shortcut is faster when it avoids much work, but the closest card might be just outside the searched area.
3. **Where the comparison breaks:** different indexes choose candidates using graphs, partitions, quantized codes or other structures. The map picture does not establish the recall or runtime of any particular algorithm.

45.3.3 PLAIN — a worked example

1. In a separate fixture, Q = (0, 0), A = (0.1, 0), B = (0.2, 0), C = (5, 5) and D = (5.1, 5). The exact top two under Euclidean distance are A and B.
2. Suppose a deliberately limited candidate stage returns only A and C. Reranking those candidates exactly still returns A and C; it cannot recover B because B was never considered.
3. The overlap with the exact top-two set is one item out of two, so neighbour recall at two is 0.5. This is approximation recall against the metric's exact answer, not human relevance recall.
4. Expanding the candidate set to A, B and C recovers the exact top two in this tiny example. It does not prove that a fixed candidate budget will achieve the same recall on a different collection.

45.3.4 PLAIN — what is really happening inside

1. Candidate generation reduces the number of full comparisons. Approximate indexes attempt to locate useful regions or neighbours without exhaustively scanning all vectors.
2. Reranking can improve ordering within the candidate set, especially when the first stage used compressed scores. Its maximum possible recovery remains limited by candidate coverage.
3. An exact baseline is valuable for measuring approximation loss on representative queries. Human-labelled relevance is another baseline, because exact geometric neighbours can still be poor answers.

45.3.5 TECHNICAL — the engineer's version

1. Faiss distinguishes exhaustive flat indexes from non-exhaustive structures such as IVF and HNSW. Index parameters affect the balance between search work and recovered neighbours. [S177]
2. Define recall@k against a fixed exact result set, including tie handling and eligible filters. A candidate stage and a reranker should be measured separately when diagnosing omissions.
3. The companion approximation example intentionally restricts a candidate list. It demonstrates the coverage limit, not the internal behaviour or performance of an ANN library.

45.3.6 WORDS — remember these

1. **Nearest neighbour:** an item closest under a metric — a geometric result whose usefulness depends on the representation. **Approximate nearest-neighbour search:** a shortcut to likely close items — retrieval that may miss exact neighbours to reduce cost. **Candidate recall:** how much required material reaches the later stage — coverage that limits what reranking can ultimately recover.

45.4 Index trade-offs

45.4.1 PLAIN — in simple words

1. A vector index organizes stored vectors so a query can search more selectively. Building and maintaining that organization costs memory, computation and update work.
2. Some designs search connected neighbours in a graph. Others first choose coarse groups and search selected groups. Compression can reduce stored vector size while introducing approximation.
3. There is no single best setting independent of data, queries and operating requirements. A setting that works well on one demonstration may behave differently after growth or distribution changes.

45.4.2 PLAIN — a picture in your head

1. Mira can connect each card to nearby cards and follow those links, or divide the map into districts and inspect selected districts.
2. More links or more inspected districts can improve the chance of finding the right card, but consume additional space or query work.
3. **Where the comparison breaks:** graph construction and partition training have precise algorithms and implementation details. A human map does not prove search complexity, deletion support or behaviour under concurrent updates.

45.4.3 PLAIN — a worked example

1. One million vectors with 768 coordinates stored as 32-bit floats require `1,000,000 × 768 × 4 = 3,072,000,000` raw vector bytes, or 3.072 decimal GB.
2. That excludes identifiers, metadata, graph links, allocator overhead and replicated copies. Quoting only the raw vector size as the service's total memory requirement would be incomplete.
3. A hypothetical product-quantized representation using 96 one-byte subvector codes has 96 code bytes per vector instead of 3,072 float bytes. Codebooks, IDs and optional original vectors still add storage, and distance estimates may become approximate.
4. A coarse-group index that probes 10 groups instead of 2 usually performs more candidate work. The actual improvement in recall must be measured; the group count alone does not guarantee a fixed accuracy level.

45.4.4 PLAIN — what is really happening inside

1. HNSW builds a hierarchy of neighbour graphs and navigates from sparse upper layers toward a more detailed lower layer. Its search effort is controlled by algorithm and implementation parameters.
2. Inverted-file vector indexes assign vectors to coarse regions. A query selects regions to probe and evaluates candidates within them, potentially using compressed codes before exact reranking.

3. Updates, deletions and model changes require maintenance policies. Some index forms rebuild, mark deletions or support limited mutation patterns; the application must not assume every library supports the same lifecycle.

45.4.5 TECHNICAL — the engineer's version

1. The original HNSW research describes hierarchical graph navigation for approximate neighbour retrieval. Its empirical results and design do not constitute a universal latency or recall guarantee for every modern implementation. [S178]
2. Faiss documents distinct storage formulas and supported operations for its index classes. Use the selected class's actual behaviour and measured overhead when planning capacity or update handling. [S177]
3. Benchmark build time, index memory, query latency distributions, recall, filtered retrieval and update/deletion behaviour. Report the exact encoder, metric, data distribution, parameters and hardware before comparing results.

45.4.6 WORDS — remember these

1. **HNSW:** hierarchical neighbour-graph search — an approximate indexing approach navigating graphs at several levels of detail. **IVF index:** search through selected coarse groups — an inverted-file organization assigning vectors to regions before candidate evaluation. **Product quantization:** compact codes for vector parts — a representation replacing subvectors with codebook references for storage and distance estimation.

45.5 Filtering and evaluation

45.5.1 PLAIN — in simple words

1. Search may need to consider only a permitted tenant, language, document type or time range. Those filters define the eligible population before usefulness is judged.
2. Taking the global top few and filtering afterward can return too few eligible results even when good permitted results exist nearby.
3. Permission filtering is not merely a quality adjustment. Protected content, scores, titles and snippets must not leak through intermediate or fallback paths.

45.5.2 PLAIN — a picture in your head

1. Mira has separate cabinets for two clients. A client asks for the nearest two cards in their own cabinet.
2. Dev first chooses the nearest two from both cabinets, then removes the other client's cards. He may return nothing while two suitable cards remain in the correct cabinet.
3. **Where the comparison breaks:** actual indexes support filters in different ways, sometimes before search, during graph traversal or after candidate generation. Correct access enforcement and recall under filtering both require verification.

45.5.3 PLAIN — a worked example

1. Let Q = (0, 0). Documents V1 and V2 belong to tenant A and lie at distances 0.1 and 0.2. V3 and V4 belong to tenant B and lie at distances 0.3 and 0.4.
2. A tenant-B request for two results should consider V3 and V4. Searching the global top two first yields V1 and V2; filtering afterward leaves zero results.

3. Searching the eligible tenant-B set exactly returns V3 and V4. Overfetching more global candidates can help this example, but a fixed overfetch count does not guarantee sufficient eligible results for every distribution.
4. The tenant scope in the lab is supplied by an explicit trusted test context. It is not accepted as arbitrary proof from a caller, and the exercise does not implement a production login or authorization provider.

45.5.4 PLAIN — what is really happening inside

1. Metadata filters interact with index traversal and candidate limits. The eligible population can be much smaller or differently distributed than the overall corpus.
2. Evaluation should measure recall after the actual filters, not only on unfiltered queries. Rare tenants or restrictive time ranges can expose failures hidden by aggregate results.
3. Empty-result and fallback behaviour matters. Relaxing a permission filter to produce an answer is not an acceptable relevance improvement; changing an optional user preference requires a visible policy.

45.5.5 TECHNICAL — the engineer's version

1. Distinguish mandatory authorization predicates from optional relevance filters. The former must remain enforced through candidate retrieval, reranking, caching and display.
2. Report exact-neighbour recall and human relevance metrics separately. The first measures approximation relative to geometry; the second measures whether the retrieved source helps with the labelled task. [S181]
3. Test filter-before-limit behaviour, cross-tenant requests, stale metadata, deleted sources and empty eligible sets. Passing vector-distance tests alone does not establish these boundaries.

45.5.6 WORDS — remember these

1. **Eligible set:** items permitted by the request's complete conditions — the population within which retrieval should answer the question. **Post-filtering:** removing candidates after retrieval — an approach that can reduce returned coverage when the candidate budget was spent on ineligible items. **Overfetching:** retrieving extra candidates before later selection — a mitigation that must be evaluated rather than assumed to guarantee filtered recall.

45.6 Retrieval limits

45.6.1 PLAIN — in simple words

1. A nearby document may be wrong, obsolete or about a subtly different issue. Similarity is a route to evidence, not a replacement for reading that evidence.
2. When a retrieved passage is supplied to a text-generating system, the generated answer adds another stage that can misinterpret, omit or invent information.
3. Good retrieval should preserve source identity, version, context and permission. A plausible answer with no trustworthy connection to its sources is difficult to verify.

45.6.2 PLAIN — a picture in your head

1. Mira finds a card near the question on her topic map. She opens the full source before using it to make a decision.

2. A helper who writes a fluent summary can still misunderstand the card. The map and the summary each need their own checks.
3. **Where the comparison breaks:** retrieved documents can contain untrusted instructions as well as facts. A system using external text must not grant that text authority to change access rules or execute operations.

45.6.3 PLAIN — a worked example

1. Suppose a passage about backup creation is geometrically close to a question about restore verification. It may discuss related vocabulary but omit the steps needed to establish that a restore actually works.
2. A retrieval score of 0.92 does not fill that gap. The answer must distinguish what the passage supports from what still needs another source or an experiment.
3. If the retrieved document belongs to an older software version, its commands may not apply to the current environment. Source version and consultation date are relevant evidence, not decorative metadata.
4. A system can correctly retrieve all three passages selected by its metric and still answer the wrong question. Evaluate retrieval, source interpretation and final factual support as separate stages.

45.6.4 PLAIN — what is really happening inside

1. Retrieval pipelines often combine lexical and vector candidates, merge them and rerank. Each stage can improve one failure mode while introducing another, such as duplicate passages crowding out diverse sources.
2. A source-grounded answer must link claims to actual supporting text and retain relevant limitations. A citation to a merely related passage is not adequate support.
3. Updating or deleting a source requires corresponding treatment of embeddings, indexes, caches and retained outputs. Removing the original file alone does not establish that every derived copy has disappeared.

45.6.5 TECHNICAL — the engineer's version

1. Separate four contracts: encoder semantics, approximate retrieval quality, access eligibility and answer support. Success at one layer does not establish the others.
2. Treat retrieved text as data from its actual trust level. It must not override system instructions, authorize external actions or bypass tenant restrictions merely because it was returned by an index.
3. The laboratory proves selected geometry, candidate-coverage and filtering examples under declared finite inputs. It makes no claim that vector proximity establishes identity, truth, safety or universal semantic understanding.

45.6.6 WORDS — remember these

1. **Hybrid retrieval:** combining different candidate methods — often lexical and vector search followed by merging or reranking. **Grounded answer:** a response supported by inspected evidence — claims linked to sources that actually establish them, with limitations retained. **Retrieval boundary:** what finding a source does and does not prove — the distinction between candidate similarity and verified, authorized factual support.

45.97 Practice and worked answers

1. **Question:** Can vectors from two unrelated encoders be compared merely because both have 768 coordinates? **Answer:** No. Dimension equality does not establish representation compatibility.
2. **Question:** What is squared distance from (1, 0) to (0.8, 0.6)? **Answer:** 0.4. The Euclidean distance is approximately 0.632.
3. **Question:** Why do A = (1, 0) and E = (2, 0) tie under cosine with Q = (1, 0)? **Answer:** They point in the same direction; cosine removes their length difference.
4. **Question:** Why reject a zero vector for cosine in the lab? **Answer:** Its norm is zero, so the normalization denominator and directional comparison are undefined.
5. **Question:** Exact top two are A/B, but candidates contain A/C. What is neighbour recall at two? **Answer:** One recovered exact neighbour divided by two, or 0.5.
6. **Question:** What does the raw 3.072 GB estimate omit? **Answer:** IDs, metadata, index structures, memory overhead, replicas and any additional stored representations.
7. **Question:** Why can global top-two retrieval followed by tenant filtering return nothing? **Answer:** The candidate budget may be filled entirely by ineligible documents before filtering.
8. **Question:** Does similarity 0.92 mean a passage is 92% true? **Answer:** No. It is a metric-dependent score, not a calibrated factual-correctness probability.

45.98 Common wrong ideas

1. **Wrong:** A vector contains an item's complete meaning. **Right:** It is a representation learned or defined for particular objectives.
2. **Wrong:** Equal dimensions make models interchangeable. **Right:** Encoder and preprocessing compatibility must be established.
3. **Wrong:** Inner product is always cosine. **Right:** Normalization and nonzero norms matter.
4. **Wrong:** Exact vector search guarantees a relevant answer. **Right:** It guarantees the selected geometric result, not human usefulness.
5. **Wrong:** Reranking can repair every candidate omission. **Right:** Missing candidates remain unavailable.
6. **Wrong:** An index's raw-vector size is total service memory. **Right:** Index and operating overhead can be substantial.
7. **Wrong:** Permission filters may be relaxed to improve recall. **Right:** Authorization defines the allowed population.
8. **Wrong:** A high-scoring citation proves the generated answer. **Right:** The cited text must actually support each claim.

45.99 Chapter summary in 20 lines

1. Vectors are ordered numerical representations.
2. Their meaning depends on the encoder and preprocessing.
3. Preserve source identity and representation version.
4. Validate dimensions and finite values.
5. Euclidean distance measures separation.
6. Squared distance preserves order but changes the returned quantity.
7. Inner product depends on vector magnitudes.
8. Cosine compares nonzero vector directions.

9. Exact search is exact only under its declared metric.
10. Approximate retrieval can omit closer items.
11. Reranking cannot recover absent candidates.
12. Measure geometric recall separately from human relevance.
13. Graphs and coarse groups trade work for candidate coverage.
14. Compression saves space with additional accuracy and decoding trade-offs.
15. Capacity includes more than raw vector bytes.
16. Filters define the eligible retrieval population.
17. Post-filtering after a small global limit can lose useful results.
18. Retrieved content remains subject to access and trust boundaries.
19. Similarity is not truth, identity or authorization.
20. Inspect the source and verify what the answer actually supports.

46

CHAPTER

Metrics, Bias and Honest Analysis

46.0 What this chapter gives you

1. A metric is a precisely defined question expressed as a number. The calculation can be flawless while the definition, population or interpretation is wrong.
2. You will inspect denominators, missing observations, selection effects and causal claims. You will also calculate an uncertainty interval while keeping clear what the interval's assumptions do and do not cover.
3. All operational comparisons in this chapter are invented teaching examples. They are not observations of KedByte customers, published business performance or evidence that a particular real process causes an improvement.

46.1 A metric is a definition

46.1.1 PLAIN — in simple words

1. "Sales," "success" and "active user" are labels, not complete calculations. A useful metric specifies which events count, which period they belong to and what is excluded.
2. A number should carry its unit and grain. Orders, order lines, items, money and customers are different quantities even when they come from the same database.
3. Changing a definition can change the result without any change in the underlying business. Such a change should be recorded, not presented as an unexplained improvement or decline.

46.1.2 PLAIN — a picture in your head

1. Mira asks how busy the shop was. Dev can count customers, orders, items or hours of queueing. Each answer describes a different part of "busy."
2. Before comparing two weeks, they agree to count the same thing under the same rules.
3. **Where the comparison breaks:** digital dashboards can hide definitions behind attractive labels and automatically refreshed charts. The visible number needs a traceable specification, not just a tooltip saying it is accurate.

46.1.3 PLAIN — a worked example

1. The canonical fixture contains two orders, four order lines, six items and 28,650 paise of agreed line amount. None of those counts can be substituted for another without changing the question.
2. Average agreed amount per order is 28,650 / 2 = 14,325 paise. Per line it is 28,650 / 4 = 7,162.5 paise. Per item it is 28,650 / 6 = 4,775 paise.
3. Calling 28,650 "cash received" would assert information the fixture does not contain. The database stores agreed order facts, not evidence of payment settlement.

4. A metric card should therefore state: agreed line amount, INR paise, these four canonical lines, no taxes/discounts/refunds/shipping included, and the selected source and transformation versions.

46.1.4 PLAIN — what is really happening inside

1. A query operationalizes a definition using filters, joins and aggregates. Every step can change the population or multiplicity of contributing records.
2. A metric registry connects a human definition to the query, owner, version and test cases. It helps prevent two teams from using one label for incompatible calculations.
3. Reconciliation should link the displayed value back to its inputs. A dashboard total is more trustworthy when its contributing keys and exclusions can be inspected under appropriate permissions.

46.1.5 TECHNICAL — the engineer's version

1. Specify numerator, denominator, eligibility, time basis, unit, grain, exclusions, missingness policy and aggregation level. Preserve the distinction between an observation and an inference.
2. SQL aggregates operate on the rows produced by prior relational operations. A wrong join or filter can create a coherent but unintended statistic. [S58] [S80]
3. Version metric definitions and record restatements. Comparing an old definition's historical series with a new definition's current value requires an explicit bridge or a recomputed comparable series.

46.1.6 WORDS — remember these

1. **Metric definition:** the exact question behind a number — rules for population, units, timing, inclusion and calculation. **Operationalization:** turning an idea into a measurable procedure — implementing a definition through observable records and calculations. **Restated series:** historical results recalculated under a new rule — a versioned view distinct from what was previously published.

46.2 Denominators and populations

46.2.1 PLAIN — in simple words

1. A percentage needs a denominator: successful out of which opportunities? Changing that population can change the rate substantially.
2. Combining percentages requires attention to group sizes. The average of two group percentages is not generally the percentage across both groups.
3. A comparison also needs comparable populations. If one process handles mostly simple cases and another mostly difficult cases, their overall rates can reflect the case mix rather than the process itself.

46.2.2 PLAIN — a picture in your head

1. Mira compares two counters. One handles ninety simple enquiries and ten difficult ones; the other handles twenty simple and eighty difficult ones.
2. Looking only at each counter's overall completion rate can hide the difference in work assigned to them.

3. **Where the comparison breaks:** "simple" and "difficult" must be defined before the outcome and applied consistently. Inventing categories afterward can create another misleading comparison.

46.2.3 PLAIN — a worked example

1. In a synthetic two-method example, method A completes 81 of 90 easy cases and 1 of 10 hard cases. Its subgroup rates are 90% and 10%, and its overall rate is 82/100 = 82%.
2. Method B completes 19 of 20 easy cases and 16 of 80 hard cases. Its subgroup rates are 95% and 20%, while its overall rate is 35/100 = 35%.
3. B's observed rate is higher within both defined subgroups, but A's overall rate is higher because A handled a much larger share of easy cases. This is a constructed aggregation reversal, often called Simpson's paradox.
4. Under a hypothetical common mix of half easy and half hard, the standardized rates are 50% for A and 57.5% for B. These are reweighted descriptive calculations, not proof that changing methods would cause those outcomes.

46.2.4 PLAIN — what is really happening inside

1. Overall rates weight subgroup rates by each subgroup's share of the denominator. When those shares differ, aggregate comparisons answer a different question from within-group comparisons.
2. Standardization applies a chosen common population mix. It can clarify one source of difference, but the selected weights and subgroup definitions must be disclosed.
3. Not every variable should automatically be adjusted for. A variable affected by the process or a selection rule can change the causal question. Descriptive reweighting is not a substitute for a justified causal design.

46.2.5 TECHNICAL — the engineer's version

1. For subgroup counts, compute a pooled rate as `sum(successes) / sum(eligible_count)`, not an unweighted mean of rates unless equal group weighting is the actual question.
2. Preserve numerator and denominator in stored aggregates so later regrouping remains possible. An average without its count may be insufficient to compute a correct combined average.
3. The Simpson example is an original arithmetic construction. It demonstrates sensitivity to population composition; it does not estimate a treatment effect or validate the easy/hard classifier for real operations.

46.2.6 WORDS — remember these

1. **Denominator:** the population a rate is out of — the eligible count or exposure giving a numerator its meaning. **Case mix:** the composition of the observed population — subgroup proportions that can influence aggregate comparisons. **Standardization:** applying a common comparison population — reweighting subgroup results under explicitly chosen weights.

46.3 Missingness

46.3.1 PLAIN — in simple words

1. Missing information is not automatically zero, failure or success. It means the required value is unavailable under the current observation process.
2. Ignoring missing records changes the denominator. That may be reasonable for a clearly labelled observed-only statistic, but it does not automatically describe the entire eligible population.

3. The reason information is missing matters. If difficult cases are less likely to be recorded, the observed subset can systematically look better than the full population.

46.3.2 PLAIN — a picture in your head

1. Mira receives delivery-status slips for only ninety of one hundred eligible deliveries. Nine of the returned slips say late and eighty-one say on time.
2. The ten absent slips could change the full-population rate. They cannot be treated as on time merely because no complaint was filed.
3. **Where the comparison breaks:** missingness can arise at several technical stages: capture, transmission, parsing, joins or permission filters. An apparently empty cell does not reveal which stage failed.

46.3.3 PLAIN — a worked example

1. There are 100 eligible deliveries, 81 known on time, 9 known late and 10 with unknown status. The observed-only rate is 81 / 90 = 90%.
2. If all unknown statuses are late, the full-population on-time rate is 81%. If all are on time, it is 91%. The data alone therefore bounds the rate between 81% and 91% under the stated binary-status assumption.
3. This range is not a statistical confidence interval. It is a deterministic sensitivity bound over the unresolved records.
4. A dashboard can report "90% among 90 observed statuses; 10 of 100 statuses missing." That statement preserves both the useful observed statistic and its limitation.

46.3.4 PLAIN — what is really happening inside

1. SQL aggregates often ignore NULL values in specific ways. `COUNT(*)` counts rows, while counting a nullable expression counts its non-null values. The denominator must match the definition.
2. Joins can create or hide missingness. An inner join can silently remove records lacking a matching status row; an outer join retains them so a policy can be applied explicitly.
3. Imputation fills missing values under a model or rule. It can be useful, but the filled values are inferred, not newly observed facts, and their uncertainty should not disappear from the analysis.

46.3.5 TECHNICAL — the engineer's version

1. Separate unknown, not applicable, not yet observed, invalid and deliberately withheld states when the workflow needs those distinctions. A single NULL field may be insufficient without status metadata.
2. Missingness assumptions affect inference. An observed-only rate represents the observed subset directly; extending it to unobserved cases requires justification beyond the division itself.
3. The companion checks compare observed-only rates with explicit best/worst bounds and reject impossible counts. They do not implement a general missing-data model or assert that an imputation rule recovers the true values. [S80]

46.3.6 WORDS — remember these

1. **Missingness:** required information is unavailable — a property of the observation process whose causes affect interpretation. **Observed-only statistic:** a result using records with available values — a measure whose population excludes unresolved observations. **Sensitivity bound:** a range un-

der alternative assumptions — an explicit calculation showing how unknown information could change a conclusion.

46.4 Selection effects

46.4.1 PLAIN — in simple words

1. The records available for analysis may not represent the population we care about. People, systems or filters can determine which cases become visible.
2. Studying only successful requests cannot reveal the experience of requests that failed before logging. Studying only customers who stayed cannot fully explain those who left.
3. More records do not automatically remove this problem. A huge systematically selected dataset can support a very precise answer to the wrong population question.

46.4.2 PLAIN — a picture in your head

1. Mira asks only customers who return next week whether they liked last week's service. Customers who decided never to return are absent from the survey.
2. The collected opinions are real, but they describe a selected group rather than all customers served.
3. **Where the comparison breaks:** selection can be implemented invisibly by query filters, telemetry sampling and retention rules. The analyst may not notice it by reading the final table alone.

46.4.3 PLAIN — a worked example

1. Suppose 1,000 requests are attempted. The application logs only the 900 that reach its success handler. Of those, 891 complete within the chosen time bound.
2. The logged-only rate is 891 / 900 = 99%. Treating all 1,000 attempts as the required denominator gives 891 / 1,000 = 89.1% known timely successes, with the other outcomes needing classification.
3. A dashboard based only on the success table can therefore look excellent while hiding 100 attempts. The arithmetic is not the defect; the observation and eligibility boundaries differ.
4. The remedy is to instrument the complete request lifecycle and reconcile attempts, successes, failures and unknown outcomes under a consistent identity scheme, not simply to rename 99% as reliability.

46.4.4 PLAIN — what is really happening inside

1. Data reaches an analytical table through capture, sampling, filtering and retention. Each stage can select cases based on properties related to the outcome.
2. Duplicate logs can bias the other direction by giving some events extra weight. The intended statistical unit might be a request, user or day rather than one stored log row.
3. Evaluation data can also be contaminated by information unavailable at the intended decision time. A predictor using a field written after completion may appear accurate while being unusable when the real decision must be made.

46.4.5 TECHNICAL — the engineer's version

1. Document the sampling frame and all inclusion/exclusion stages. Trace a record's route into the analysis and inspect cases lost before the final query.

2. For temporal evaluation, preserve the feature-availability boundary: the model or rule should use only information available at the relevant prediction time. Keep development and evaluation decisions separate to reduce overfitting.
3. Repeated observations from one subject or time series can be dependent. NIST's autocorrelation discussion illustrates why order and dependence matter; treating every row as an independent draw can overstate information. [S179]

46.4.6 WORDS — remember these

1. **Selection bias:** the observed group systematically differs from the target — distortion caused by how cases enter or remain in the dataset. **Sampling frame:** the cases available for selection — the actual population from which observations can be drawn. **Information leakage:** using knowledge unavailable at the intended decision point — contamination that can make evaluation results unrealistically strong.

46.5 Association versus cause

46.5.1 PLAIN — in simple words

1. Two quantities changing together establishes an observed association, not automatically that one caused the other.
2. Another factor can influence both, or the direction can run the other way. The data-collection process can also create an apparent relationship.
3. A causal question asks what would change under an intervention. Answering it requires a justified design and assumptions, not just a chart with a sloping line.

46.5.2 PLAIN — a picture in your head

1. Mira notices that busy days have longer queues. She should not conclude that deliberately making people queue will create more demand.
2. Demand may contribute to queue length, while staffing, promotions or holidays influence both. Several stories can fit the same observed pattern.
3. **Where the comparison breaks:** causal reasoning can be formalized with experiments and explicit models. The point is not that causes are unknowable, but that the evidence must distinguish plausible explanations.

46.5.3 PLAIN — a worked example

1. Consider a proposed new checkout workflow. Comparing this month's rate with last month's rate also changes season, staffing, customer mix and perhaps measurement rules.
2. Randomly assigning eligible cases to old and new workflows can help separate assignment from those pre-existing differences, provided the experiment is properly designed and implemented.
3. Randomization does not guarantee identical finite groups, and shared queues can create interference between them. Missing outcomes, noncompliance and changed logging can still damage the interpretation.
4. The synthetic A/B case-mix table in section 46.2 was not randomized. Its subgroup and standardized percentages remain descriptive calculations, not experimentally established causal effects.

46.5.4 PLAIN — what is really happening inside

1. Observational data records what happened under existing selection and assignment processes. Counterfactual outcomes under another action are not directly present in the same record.
2. Experiments define interventions, assignment, outcomes and analysis rules. Observational causal studies require additional assumptions about confounding, timing and selection.
3. Good reporting separates the observed difference from the causal interpretation and states the design limitations. It does not dismiss useful associations, but it does not promote them into stronger claims without evidence.

46.5.5 TECHNICAL — the engineer's version

1. A scatter plot can reveal patterns and possible relationships but does not establish cause and effect. NIST explicitly distinguishes this diagnostic use from causal proof. [S180]
2. Define the estimand: the population, intervention, outcome and contrast being sought. Choosing covariates or subgroups should follow that causal question rather than automatically adjusting for every available field.
3. The book does not estimate a real treatment effect. Its examples teach the separation between arithmetic, descriptive association and causal inference, leaving a real study's assumptions and review to its actual context.

46.5.6 WORDS — remember these

1. **Association:** quantities vary together in observations — a statistical relationship that does not by itself establish intervention effects. **Confounder:** a factor affecting the causal comparison — a variable related to both assignment/exposure and outcome in a relevant causal structure. **Estimand:** the precise quantity a study seeks — a defined population-level contrast under specified interventions or conditions.

46.6 Communicating uncertainty

46.6.1 PLAIN — in simple words

1. A reported number can be uncertain because only a sample was observed, measurements are imperfect or records are missing. These are different sources of uncertainty.
2. An interval calculated from a sampling model covers only the uncertainty represented by that model. It does not repair biased selection, incorrect labels or a changing process.
3. Explain the result with its denominator, assumptions and practical limits. Extra decimal places should not create the appearance of knowledge the evidence does not contain.

46.6.2 PLAIN — a picture in your head

1. Mira checks a sample of packages rather than every package. She can calculate how much the observed proportion might vary under repeated comparable sampling.
2. If she checked only the easiest packages to reach, a narrow interval around that selected sample does not solve the selection problem.
3. **Where the comparison breaks:** statistical coverage is a property of a procedure under assumptions. It is not a promise that every particular interval contains the fixed unknown quantity.

46.6.3 PLAIN — a worked example

1. In a separate idealized sample, 90 of 100 independent, comparably sampled cases satisfy a binary condition. The observed proportion is 0.9.
2. Using a Wilson interval with z = 1.96 gives an approximate two-sided 95% interval from 0.826 to 0.945. This calculation assumes a suitable binomial sampling model and is not the same as the missing-status bound from section 46.3.
3. The Wilson centre is `(p + z²/(2n)) / (1 + z²/n)`. Its half-width is `z × sqrt(p(1-p)/n + z²/(4n²)) / (1 + z²/n)`. Substituting p = 0.9 and n = 100 produces the stated rounded limits.
4. The companion code checks the arithmetic, edge cases and valid range. It does not establish that real operational observations satisfy independence, stable probability or representative sampling.

46.6.4 PLAIN — what is really happening inside

1. Sampling intervals describe repeated-sampling performance under a model. Measurement uncertainty instead concerns how values are attributed from instruments and observations; missingness bounds explore unresolved cases.
2. Dependence reduces the information obtained from repeated observations compared with an ideal independent sample of the same row count. A thousand correlated readings are not automatically a thousand independent pieces of evidence.
3. Report uncertainty together with decision relevance. A small numerical difference may be operationally unimportant, while a modest uncertainty range near a safety or capacity limit may matter greatly.

46.6.5 TECHNICAL — the engineer's version

1. NIST presents the Wilson interval and discusses alternatives for small samples or few failures. Avoid the naive interpretation that a confidence interval assigns a 95% posterior probability to a fixed parameter without a Bayesian model. [S183]
2. A confidence procedure's nominal coverage is conditional on its statistical assumptions and can be approximate. Selection, dependence, repeated unplanned testing and definition changes can invalidate a simplistic interpretation. [S179]
3. Preserve raw counts, calculation method, source version and rounding policy. Label unresolved measurement and data-quality limitations separately instead of folding them into an unjustified single "confidence" percentage.

46.6.6 WORDS — remember these

1. **Confidence interval:** a range from a sampling procedure — an interval method with stated repeated-sampling coverage under assumptions. **Wilson interval:** a binomial-proportion interval — a score-based method whose limits remain within the valid probability range. **Statistical unit:** the entity providing an observation — the request, person, item or time unit whose independence and weighting need definition.

46.97 Practice and worked answers

1. **Question:** What does 28,650 paise mean in the canonical fixture? **Answer:** The agreed amount of the four included order lines, not verified cash collection.
2. **Question:** Why does averaging subgroup percentages often fail? **Answer:** It ignores subgroup denominator sizes unless equal group weighting is the intended question.

3. **Question:** What explains the A/B aggregate reversal? **Answer:** Different easy/hard case mixtures. The constructed arithmetic does not establish a causal effect.
4. **Question:** With 81 on time, 9 late and 10 unknown out of 100, what is observed-only performance? **Answer:** 90% among the 90 observed statuses; the full-population binary bound is 81% to 91%.
5. **Question:** Is that 81%–91% range a confidence interval? **Answer:** No. It is a sensitivity bound over missing outcomes.
6. **Question:** Why can success-only logs mislead? **Answer:** They omit attempts that never reached the success handler, changing the denominator and population.
7. **Question:** Does a sloping scatter plot establish cause? **Answer:** No. It shows association; causal interpretation needs additional design and assumptions.
8. **Question:** What does the Wilson calculation fail to verify? **Answer:** Whether the real observations satisfy the sampling, independence, measurement and stability assumptions required for its interpretation.

46.98 Common wrong ideas

1. **Wrong:** A familiar metric name is a complete definition. **Right:** Population, timing, units and exclusions must be explicit.
2. **Wrong:** A correct formula guarantees a correct conclusion. **Right:** The formula may answer the wrong question.
3. **Wrong:** Group percentages can always be averaged directly. **Right:** Denominator weights matter.
4. **Wrong:** Missing means failure or zero. **Right:** Missingness needs a declared interpretation and may require separate status.
5. **Wrong:** More rows automatically remove bias. **Right:** Systematic selection can persist at any scale.
6. **Wrong:** Association is evidence enough for intervention effects. **Right:** Causal conclusions require stronger assumptions or design.
7. **Wrong:** Every repeated measurement is independent. **Right:** Dependence can reduce effective information.
8. **Wrong:** A confidence interval covers every source of uncertainty. **Right:** It covers only what the selected statistical model represents.

46.99 Chapter summary in 20 lines

1. A metric is a defined question, not only a label.
2. Preserve the distinction between orders, lines, items and money.
3. State the numerator, denominator and eligible population.
4. Keep units and time basis explicit.
5. Version definition changes and historical restatements.
6. Pool counts before calculating a pooled rate.
7. Case mix can reverse aggregate comparisons.
8. Standardization is a declared reweighting, not automatic causal proof.
9. Missing observations do not become known outcomes by convenience.
10. Report observed-only statistics with their coverage.
11. Sensitivity bounds and confidence intervals answer different questions.
12. Selection can occur before data reaches the final table.

13. Success-only logs can hide unsuccessful attempts.
14. Statistical units are not always individual stored rows.
15. Association does not by itself establish cause.
16. Define the intervention and estimand for causal questions.
17. Sampling intervals rely on explicit assumptions.
18. Dependence and selection are not cured by extra decimal places.
19. Preserve counts, methods and unresolved limitations.
20. Communicate the strongest conclusion the evidence actually supports.

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

Allowed lateness

the accepted revision interval — a configured policy governing how long late window data may affect retained computation.

Read in context: 41.5.6

Analytical workload

queries that study populations of records — work often involving scans, grouping and historical comparison.

Read in context: 42.1.6

Analyzer

the text interpretation pipeline — extraction, tokenization and normalization rules used for indexing and querying.

Read in context: 44.2.6

Approximate nearest-neighbour search

a shortcut to likely close items — retrieval that may miss exact neighbours to reduce cost.

Read in context: 45.3.6

Architecture trade-off

a benefit obtained with another cost or constraint — a choice evaluated against the complete task rather than one isolated metric.

Read in context: 42.6.6

Association

quantities vary together in observations — a statistical relationship that does not by itself establish intervention effects.

Read in context: 46.5.6

Atomic metadata publication

one accepted version replaces its expected predecessor — a concurrency boundary preventing incompatible partial or stale updates.

Read in context: 42.4.6

Atomic publication

output becomes accepted as one local change — committing records and their completion evidence within one transaction boundary.

Read in context: 40.5.6

B

BM25

a family of lexical ranking rules — scoring that combines term weighting, frequency saturation and document-length effects.

Read in context: 44.4.6

C

Candidate recall

how much required material reaches the later stage — coverage that limits what reranking can ultimately recover.

Read in context: 45.3.6

Candidate retrieval

finding possible answers — selecting documents for further scoring and verification under a query model.

Read in context: 44.1.6

Candidate row

a record not yet ruled out — input retained until the necessary conditions can be evaluated correctly.

Read in context: 43.4.6

Case mix

the composition of the observed population — subgroup proportions that can influence aggregate comparisons.

Read in context: 46.2.6

Catalogue

organized metadata about data assets — a service or representation locating tables and describing their definitions and versions.

Read in context: 42.3.6

Change data capture

following source mutations — obtaining inserts, updates and deletions for downstream processing.

Read in context: 40.3.6

Checkpoint

how far processing has reached — a recorded boundary of incorporated source input.

a recorded starting boundary for recovery — coordinated data and metadata progress that limits required redo under an engine's protocol.

recorded safe progress — a recovery marker bound to preserved state and explicitly scoped effects.

Read in context: 40.5.6

Columnar storage

values grouped by field — a layout that can reduce irrelevant reads for analytical projections.

Read in context: 43.1.6

Compression block

a unit compressed and decoded together — a boundary affecting storage efficiency and selective-read cost.

Read in context: 43.3.6

Confidence interval

a range from a sampling procedure — an interval method with stated repeated-sampling coverage under assumptions.

Read in context: 46.6.6

Confounder

a factor affecting the causal comparison — a variable related to both assignment/exposure and outcome in a relevant causal structure.

Read in context: 46.5.6

Conservative pruning

skipping only what cannot match — a rule that may retain extra candidates but must not discard valid results.

Read in context: 43.5.6

Consistent cut

observations that fit one permitted source history — a boundary that avoids incompatible mixtures of concurrent states.

Read in context: 40.2.6

Contiguous progress

every required earlier position is included — a stronger condition than merely having seen a large position number.

Read in context: 40.6.6

Cosine similarity

directional agreement — inner product divided by the product of nonzero vector norms.

Read in context: 45.2.6

D

Data clustering

placing similar values near one another — a layout choice that can tighten summaries and improve selective access.

Read in context: 43.5.6

Data contract

the agreement about incoming information — explicit structure, semantics, identity and compatibility rules for a data interface.

Read in context: 40.1.6

Data lineage

how outputs depend on inputs — a recorded graph of source versions, transformations and derived assets.

Read in context: 42.5.6

Data minimization

limiting information to a justified purpose — a design practice reducing unnecessary collection, copying and exposure.

Read in context: 42.5.6

Decode cost

work to reconstruct usable values — CPU and memory effort introduced by an encoding or compression layer.

Read in context: 43.6.6

Delta encoding

storing changes from earlier values — a representation whose savings depend on the size and regularity of differences.

Read in context: 43.2.6

Denominator

what the division is over; technically, the quantity that defines the base of a ratio or average.

what the total is divided by — the explicitly defined count or weight underlying a ratio.

the population a rate is out of — the eligible count or exposure giving a numerator its meaning.

Read in context: 46.2.6

Deterministic transformation

the same complete inputs yield the same output — a rule without uncontrolled time, randomness or external-state dependencies.

Read in context: 40.4.6

Dictionary encoding

replacing values with reusable codes — a representation containing a distinct-value dictionary and an occurrence-code sequence.

Read in context: 43.2.6

Dimension

the physical kind of a quantity; technically, its expression in powers of the chosen base quantities, such as length or time.

descriptive context for grouping facts — attributes connected to facts under a current or historical interpretation.

Read in context: 42.2.6

Document

the searchable unit — an identified body of content, potentially a whole page or a bounded passage.

Read in context: 44.1.6

Document frequency

how many documents contain a term — a collection statistic used to distinguish common from rarer terms.

Read in context: 44.4.6

E

Edit distance

the number of selected character edits between strings — a spelling similarity measure, not a proof of intended identity.

Read in context: 44.5.6

Eligible set

items permitted by the request's complete conditions — the population within which retrieval should answer the question.

Read in context: 45.5.6

Embedding

an item's numerical representation — an ordered vector produced by a defined encoder and preprocessing pipeline.

Read in context: 45.1.6

Embedding space

the coordinate system used for comparison — a representation whose geometry depends on its model and training objective.

Read in context: 45.1.6

End-to-end measurement

timing the complete defined operation — evidence that includes all stages required to produce and consume the intended result.

Read in context: 43.6.6

Estimand

the exact quantity a study seeks — the defined performance or outcome measure to which observations are intended to speak.

the precise quantity a study seeks — a defined population-level contrast under specified interventions or conditions.

Read in context: 46.5.6

Euclidean distance

straight-line separation — the square root of summed squared coordinate differences.

Read in context: 45.2.6

Event time

when the described occurrence happened; technically, the timestamp assigned to the event under its source's semantics.

when the described event is said to have happened — a timestamp assigned under the source and pipeline's semantic contract.

Read in context: 41.2.6

Extraction boundary

exactly which source state or records were selected — the snapshot, interval or immutable object defining a batch's coverage.

Read in context: 40.2.6

F

Fact table

measurable observations at a declared grain — analytical records whose keys and units define what may be aggregated.

Read in context: 42.2.6

Field ID

stable column identity independent of a display name — metadata used to preserve meaning through supported schema changes.

Read in context: 42.4.6

Freshness objective

how current a result must be — an explicit bound connecting publication to its required source state.

Read in context: 42.6.6

G

Grounded answer

a response supported by inspected evidence — claims linked to sources that actually establish them, with limitations retained.

Read in context: 45.6.6

H

HNSW

hierarchical neighbour-graph search — an approximate indexing approach navigating graphs at several levels of detail.

Read in context: 45.4.6

Hybrid retrieval

combining different candidate methods — often lexical and vector search followed by merging or reranking.

Read in context: 45.6.6

I

Idle input

a source currently producing no records — a condition requiring a policy, not proof that earlier events can never return.

Read in context: 41.4.6

Incremental extraction

reading changes since recorded progress — a method whose cursor must account for the source's actual change semantics.

Read in context: 40.2.6

Information leakage

using knowledge unavailable at the intended decision point — contamination that can make evaluation results unrealistically strong.

Read in context: 46.4.6

Information need

what the reader is trying to learn — the task that a query attempts to express.

Read in context: 44.1.6

Ingestion time

when a system accepted the input — a separately useful observation that does not replace the original event time.

Read in context: 41.2.6

Inner product

coordinate products added together — a similarity score affected by stored-vector magnitudes as well as direction.

Read in context: 45.2.6

Inverted index

a word-to-document map — an access structure locating documents containing searchable terms.

Read in context: 44.3.6

IVF index

search through selected coarse groups — an inverted-file organization assigning vectors to regions before candidate evaluation.

Read in context: 45.4.6

L

Late data

a record behind the accepted progress frontier — input whose event time precedes the relevant watermark or closed-window boundary.

Read in context: 41.4.6

Lexeme

a normalized searchable form — a dictionary or analyzer output that may combine several written word forms.

Read in context: 44.2.6

Lossless compression

a smaller representation with exact recovery — byte encoding whose decoder reproduces the original input.

Read in context: 43.3.6

Lossy transformation

some source detail is discarded — a mapping that cannot generally be inverted to recover the original records.

Read in context: 40.4.6

M

Manifest

the record of which storage files belong to the current state — metadata coordinating installed runs and their lifecycle.

an explicit inventory for a data version — metadata naming the files or entries that belong to a committed state.

Read in context: 42.3.6

Materialized view

a stored query result — derived data whose refresh policy determines how current it is.

Read in context: 42.1.6

Metric definition

the exact question behind a number — rules for population, units, timing, inclusion and calculation.

Read in context: 46.1.6

Metric owner

the accountable interpreter of a reported measure — a role responsible for definition changes, exclusions and disputes.

Read in context: 42.5.6

Missingness

required information is unavailable — a property of the observation process whose causes affect interpretation.

Read in context: 46.3.6

N

Nearest neighbour

an item closest under a metric — a geometric result whose usefulness depends on the representation.

Read in context: 45.3.6

O

Observed-only statistic

a result using records with available values — a measure whose population excludes unresolved observations.

Read in context: 46.3.6

Operational workload

queries and changes that run the business — work often focused on small, current, constrained transactions.

Read in context: 42.1.6

Operationalization

turning an idea into a measurable procedure — implementing a definition through observable records and calculations.

Read in context: 46.1.6

Operator state

remembered information used by a computation — partial aggregates, identities, timers and metadata needed to process later input.

Read in context: 41.6.6

Orphan file

a stored object not referenced by an accepted table state — prepared or abandoned data requiring a deliberate retention or cleanup policy.

Read in context: 42.3.6

Output grain

one resulting row means one what — the level at which a transformed record states a fact.

Read in context: 40.4.6

Overfetching

retrieving extra candidates before later selection — a mitigation that must be evaluated rather than assumed to guarantee filtered recall.

Read in context: 45.5.6

P

Partition progress

how far one input segment has advanced — a scoped position that must not be confused with universal completion.

Read in context: 41.1.6

Positional index

an index retaining term locations — metadata supporting phrases and proximity conditions beyond word presence.

Read in context: 44.3.6

Post-filtering

removing candidates after retrieval — an approach that can reduce returned coverage when the candidate budget was spent on ineligible items.

Read in context: 45.5.6

Posting list

occurrences associated with one term — document references, often accompanied by frequency or position information.

Read in context: 44.3.6

Precision

repeated readings agreeing closely; technically, closeness among replicate indications or measured values under specified conditions.

how much of the returned material is relevant — relevant retrieved items divided by retrieved items under a stated cutoff and unit.

Read in context: 44.6.6

Predicate

a condition used to choose records — an expression whose truth controls qualification in a query operation.

a condition used to select records — an expression whose truth determines query eligibility under the relevant semantics.

Read in context: 43.4.6

Predicate pushdown

evaluating selection closer to storage — moving supported filtering work earlier to reduce later processing.

Read in context: 43.4.6

Processing time

when a processor handles a record — time measured by the executing system, potentially different on replay.

Read in context: 41.2.6

Product quantization

compact codes for vector parts — a representation replacing subvectors with codebook references for storage and distance estimation.

Read in context: 45.4.6

Projection pruning

avoiding unneeded fields — selecting only the columns required to compute the query result.

Read in context: 43.1.6

Provenance

the history behind a record; technically, information about entities, activities, agents and derivations involved in producing it.

where information came from and how it was handled — recorded source, capture and transformation context, not automatic proof of truth.

the recorded origin and processing history — information linking source entities, transformations and resulting data.

Read in context: 40.1.6

Q

Query expansion

adding related search terms — a recall-oriented technique whose extra candidates can change precision and meaning.

Read in context: 44.5.6

R

Read amplification

extra storage work for a logical read — a defined ratio or probe count measured at a specified layer.

reading more bytes than the requested logical data — additional work caused by layout, blocks, metadata or access patterns.

Read in context: 43.3.6

Recall

the share of true matches found; technically, true positives divided by the actual positive cases under the evaluation labels.

how much relevant material was found — relevant retrieved items divided by the known relevant population.

Read in context: 44.6.6

Reciprocal rank

a measure of how soon the first useful answer appears — one divided by its rank, with an explicit no-answer convention.

Read in context: 44.6.6

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: 40.6.6

Replacement result

a new complete answer for the same key — output that supersedes an earlier revision rather than adding to it.

Read in context: 41.5.6

Replay-safe delivery

repetition does not duplicate the intended effect — handling a repeated identity consistently while detecting conflicting payloads.

Read in context: 40.5.6

Replication slot

retained source progress for a consumer — a PostgreSQL mechanism whose log retention and restart behaviour require monitoring.

Read in context: 40.3.6

Representation compatibility

vectors can be compared meaningfully — agreement between encoder, dimension, normalization and metric contracts.

Read in context: 45.1.6

Reprocessing

applying rules again to retained inputs — a controlled rebuild whose interpretation depends on input and transformation versions.

Read in context: 40.6.6

Restated series

historical results recalculated under a new rule — a versioned view distinct from what was previously published.

Read in context: 46.1.6

Restatement

a later version of an earlier report — a deliberate correction whose relationship to previously published results remains visible.

Read in context: 41.6.6

Retrieval boundary

what finding a source does and does not prove — the distinction between candidate similarity and verified, authorized factual support.

Read in context: 45.6.6

Row group

a horizontal subset stored in column pieces — a unit connecting corresponding column chunks within a columnar file.

Read in context: 43.1.6

Run-length encoding

storing repetitions as a value and count — an encoding effective when identical values are consecutive.

Read in context: 43.2.6

S

Sampling frame

the set from which observations can be selected; technically, the operational list or mechanism used to reach candidate sample units.

the cases available for selection — the actual population from which observations can be drawn.

Read in context: 46.4.6

Selection bias

the selection process distorting the intended picture; technically, systematic differences caused by how observations enter the analysed sample relative to the target population.

the observed group systematically differs from the target — distortion caused by how cases enter or remain in the dataset.

Read in context: 46.4.6

Sensitivity bound

a range under alternative assumptions — an explicit calculation showing how unknown information could change a conclusion.

Read in context: 46.3.6

Session window

a group linked by activity gaps — a window that may merge when an arriving event connects earlier groups.

Read in context: 41.3.6

Sliding window

overlapping intervals — windows whose width exceeds their step, allowing one event to contribute to multiple results.

Read in context: 41.3.6

Slowly changing dimension

a policy for evolving descriptions — a modelling approach that may overwrite attributes or preserve dated versions.

Read in context: 42.2.6

Snapshot-to-stream handover

joining the initial copy to later changes — a protocol preventing gaps and managing overlap at a capture boundary.

Read in context: 40.3.6

Snippet

a displayed excerpt from a result — contextual text that requires source attribution and safe rendering.

Read in context: 44.5.6

Standardization

applying a common comparison population — reweighting subgroup results under explicitly chosen weights.

Read in context: 46.2.6

State checkpoint

a recoverable state boundary — a saved representation coordinated with the progress and effects it claims.

Read in context: 41.6.6

Statistical unit

the entity providing an observation — the request, person, item or time unit whose independence and weighting need definition.

Read in context: 46.6.6

Stream

an ongoing supply of records — input processed as a continuing sequence rather than only as a completed finite batch.

Read in context: 41.1.6

T

Table snapshot

a coherent table version — metadata selecting the files and interpretation visible to a reader.

Read in context: 42.4.6

Tie-break

a rule ordering equal-scored results — a deterministic convention that does not itself establish relevance.

Read in context: 44.4.6

Tokenisation

splitting text into searchable units — applying a specified segmentation rule to content and queries.

Read in context: 44.2.6

Transformation version

identify the calculation rules used — a versioned definition of parsing, filtering, joining and aggregation semantics.

which processing rule was used — an identifier binding output to a particular implementation and its parameters.

Read in context: 40.1.6

Trigger

the rule deciding when to publish — a condition for emitting a window result or pane.

Read in context: 41.5.6

U

Unbounded input

no known final record — a collection whose completion is not generally available during normal operation.

Read in context: 41.1.6

V

Vectorized execution

processing batches of values together — an implementation approach reducing overhead and exploiting suitable data layout.

Read in context: 43.6.6

W

Watermark

a statement of event-time progress — a source- and framework-dependent frontier used for window and timer decisions.

Read in context: 41.4.6

Wilson interval

a binomial-proportion interval — a score-based method whose limits remain within the valid probability range.

Read in context: 46.6.6

Window

a defined group of time-related records — a membership rule used to bound aggregation over a stream.

Read in context: 41.3.6

Workload specification

a description of the actual work — volumes, distributions, query patterns and operating requirements used to evaluate a design.

Read in context: 42.6.6

Z

Zone map

summary bounds for a data region — metadata such as minimum and maximum used for conservative elimination.

Read in context: 43.5.6

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