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 A · Chapters 1–6 · First Edition

Meaning Before Machines

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 A

Meaning Before Machines

Learn what a record says, what it leaves out, and what a system can safely conclude from it.

This volume contains Chapters 1–6 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 A

Meaning Before Machines

Learn what a record says, what it leaves out, and what a system can safely conclude from it.

CHAPTER 1

The Whole Data System in One Look

1.0 What this chapter gives you

By the end of this chapter, you should be able to follow one sale from a person's action to a stored record and then to a report. You should also be able to explain why a correct-looking screen is not enough evidence that the whole operation succeeded.

You will distinguish a record from the event it describes, a database from the program that manages it, and an original record from an answer calculated from it. You will learn where the word "saved" needs a more precise question. No programming installation is needed to read the chapter.

The shop we will grow with

Mira runs a fictional stationery shop called Mira's Corner. Dev helps at the counter. At first they can remember most of the day's work. Later they will need several tills, an online shop, deliveries and reports. We will add machinery only when a specific problem creates a reason for it.

All people, orders, stock counts, identifiers and amounts in this case study are invented teaching data. They are not observations of a real KedByte customer or a live commercial system.

Our first order is **O-1042**. A customer buys two notebooks at INR 75.50 each and one pen at INR 20.00. The total is INR 171.00. For now there are no taxes, discounts, refunds or delivery charges in the calculation. Those omissions are deliberate boundaries, not a suggestion that a real checkout can ignore them.

1.1 One sale, many responsibilities

1.1.1 PLAIN — in simple words

1. Selling an item and recording the sale are different actions. A notebook can leave the shelf without the computer learning about it. A computer can also contain a sale that never actually happened. A useful data system connects the work people do with records that other people can interpret and check.

2. For our order, Mira needs answers to several separate questions: What was requested? What price was agreed? What was handed over? What stock should remain? A later payment record may answer another question, but the presence of an order alone does not answer it.

3. The first job is therefore not "put everything in a database." It is to decide which facts the shop needs, what each fact means, and how an action will produce the right record.

1.1.2 PLAIN — a picture in your head

1. Imagine a counter with four trays. One holds requests, one holds checked order slips, one holds stock movements and one holds reports. Moving a slip into the order tray does not magically move a physical notebook or update every other tray.

2. The trays make responsibilities visible. They do not prescribe four separate databases. **Where the comparison stops:** software may store several kinds of records together and change some of them in one controlled operation. The important separation is meaning, not furniture.

1.1.3 PLAIN — a worked example

Order line	Item	Quantity	Unit price	Line amount
1	Notebook	2	INR 75.50	INR 151.00
2	Pen	1	INR 20.00	INR 20.00
Total	Two different items	3 units	Not applicable	INR 171.00

1. First multiply 2 by 75.50 to obtain 151.00. Then multiply 1 by 20.00 to obtain 20.00. Add the two line amounts: 151.00 + 20.00 = 171.00.

2. If the notebook count starts at 12 and both notebooks are handed over, the expected count becomes 10. If the pen count starts at 25 and one pen is handed over, it becomes 24. These are consequences of the stated example, not claims that every inventory system changes stock at the same business moment.

3. Notice three different counts: one order, two order lines and three physical units. They are all correct. They answer different questions.

1.1.4 PLAIN — what is really happening inside

1. Our proposed application receives the chosen item identifiers and quantities. It checks them against the shop's rules, determines the agreed prices, and prepares the records that should result. A storage component accepts or rejects the changes it is asked to make. The application then presents an outcome to Dev.

2. Each arrow can introduce a mistake. The wrong item can be selected. A correct item can be paired with an outdated price. A response can disappear after the records have changed. A report can add the wrong field even when the underlying order is correct.

3. A system diagram is valuable because it gives those possible mistakes a location. "The computer is wrong" becomes "the price selected before saving was wrong" or "the report counted lines rather than orders." Those are problems somebody can investigate.

1.1.5 TECHNICAL — the engineer's version

1. We will call the screen and workflow code the **application**, the accepted stored facts the **database**, and the software responsible for database operations the **database management system**, or DBMS. An application can use an embedded engine inside its process or communicate with a server. Python's SQLite interface is an example of access to an embedded SQL engine. [S20]

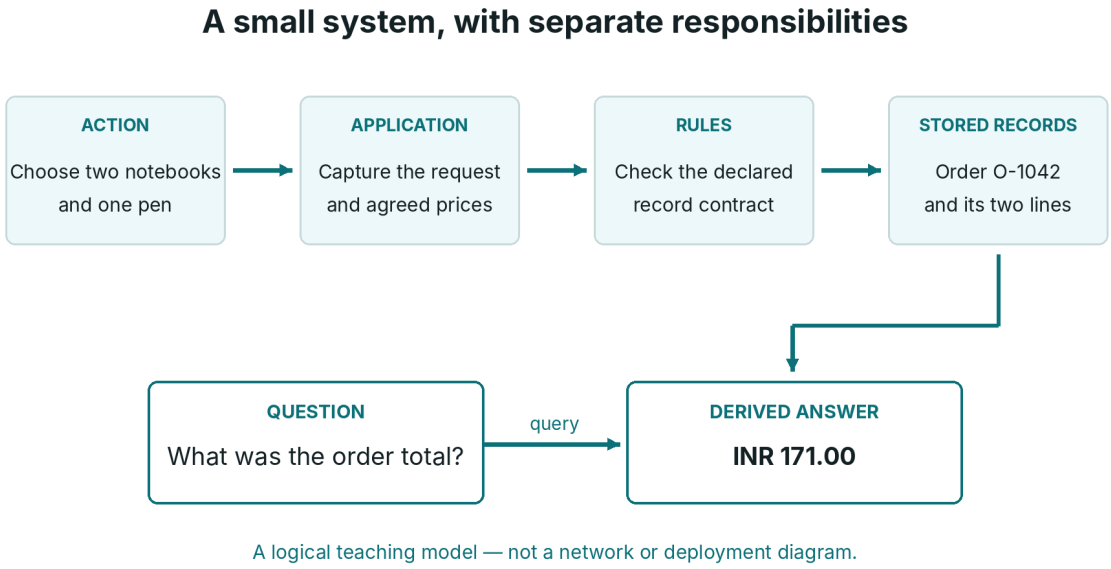

Figure 1.1: Figure 1.1. The teaching system: actions become records; questions produce derived answers. The diagram is a logical model, not a deployment design.

2. The toy order has two related structures. An order header identifies the overall order. Its lines identify the individual pricing decisions within that order. The tuple `(order_id, line_no)` will identify a line in our examples; the same product could legitimately appear on two different lines.

3. Our initial invariant is that the order total equals the sum of quantity multiplied by agreed unit price across its lines. An **invariant** is a rule we intend every accepted state to satisfy. Naming the rule does not enforce it. Later chapters will examine how to enforce it, including when requests overlap.

4. The boundary matters: a database transaction can group its participating database changes. It does not, merely by existing, turn handing over goods, sending a message and charging an external service into the same atomic action. [S01]

1.1.6 WORDS — remember these

1. **Application:** the program people use; technically, the software that implements a workflow and requests operations from other components.

2. **Database:** the organised records; technically, a managed collection of data interpreted under a model and its rules.

3. **DBMS:** the record-management machinery; technically, the engine providing database operations and its documented guarantees.

4. **Invariant:** a rule that must keep holding; technically, a predicate required of accepted states or permitted transitions.

1.2 A record is not the world

1.2.1 PLAIN — in simple words

1. A record is a statement we have preserved. It may describe an observation, a request, an estimate, a plan or a decision. It is not automatically proof that the described event occurred.

2. If Dev types "12 notebooks," the system can preserve that statement perfectly while there are actually 11 notebooks on the shelf. Perfect storage and accurate observation are different achievements.

3. This is not a reason to distrust every record. It is a reason to ask what evidence supports the particular claim you intend to make from it.

1.2.2 PLAIN — a picture in your head

1. A photograph of a shelf is not the shelf. It captures one view at one time. It may hide an item behind another, and it cannot show what happened after the photograph was taken.

2. A stock record has a similar boundary. **Where the comparison stops:** records can describe things no photograph captures well, such as a reservation or an intended delivery. Some records create an administrative state rather than merely observe a physical scene.

1.2.3 PLAIN — a worked example

1. At 09:00 Dev counts 12 notebooks and records the count. At 09:30 two notebooks are sold under O-1042. At 10:00 a new count finds only nine.

2. The expected count is 12 - 2 = 10. The observed count is nine. The discrepancy is one notebook. The arithmetic identifies a disagreement; it does not identify the cause.

3. Possible explanations include a counting mistake, an unrecorded sale, damage or an incorrectly recorded starting count. In the story we do not yet know which explanation is true. Changing the computer's count to nine may reconcile the current number, but it does not establish what happened.

1.2.4 PLAIN — what is really happening inside

1. The system receives values through a boundary: a form, a scanner, a file or another program. It can check some properties of those values. For example, it can require a count to be a non-negative whole number. That rule rejects "twelve-ish" but still accepts an incorrect count of 12.

2. We therefore separate **validation** from **verification against evidence**. Validation asks whether a record obeys declared rules. Verification asks whether the claim is supported by an appropriate observation or other evidence. Our usage is practical; a particular discipline may use these terms more formally.

3. W3C's data-publication guidance treats quality information and provenance as things that should accompany data, not as properties readers should infer merely from a file being available. [S06]

1.2.5 TECHNICAL — the engineer's version

1. For the shop, distinguish three predicates. `well_formed(record)` means the record can be interpreted under the format. `valid(record)` means it satisfies the declared domain rules. `supporte`

`d(record, evidence)` means the available evidence supports a stated claim. These are teaching predicates, not functions supplied automatically by a database.

2. An integer stock count may satisfy a type rule and a range rule while failing the third predicate. Conversely, a physically observed count can arrive in a malformed file and fail the first predicate. Different defects need different remedies.

3. In PostgreSQL, a check constraint can enforce an expression over the row being checked. That mechanism does not independently observe a shelf. The difference between an enforceable row rule and an external fact is essential to the design review. [S02]

4. A useful discrepancy record would identify the affected item, the expected quantity, the observed quantity, the relevant time, the source of the observation and the disposition. Recording "investigation pending" is more accurate than inventing a cause to make the record look complete.

1.2.6 WORDS — remember these

1. **Observation:** something someone or something noticed; technically, an acquisition of information under a stated method and context.

2. **Validation:** checking the stated rules; here, testing whether a representation meets declared structural or domain requirements.

3. **Evidence:** what supports a claim; technically, information whose relevance and limitations must be assessed for that claim.

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

1.3 Files, databases and the software between them

1.3.1 PLAIN — in simple words

1. A file is a place where a program can keep a sequence of bytes. A spreadsheet file can hold a useful small dataset. A database engine provides a different level of service: programs ask it to manage records according to its rules rather than each inventing their own way to edit the underlying storage.

2. The distinction is not "files are bad, databases are good." A database itself uses storage representations. The question is who coordinates the work and which guarantees that component supplies.

3. For Mira alone, a carefully maintained file may be enough. Once several tills change the same stock at once, the coordination requirement becomes much more demanding.

1.3.2 PLAIN — a picture in your head

1. A shared notebook lets people read and write. A record office adds an agreed process for accepting forms, identifying records, answering requests and controlling changes. The office represents the service provided around the records.

2. **Where the comparison stops:** a database engine is not a clerk who understands your intentions. It applies the rules and operations you have actually defined. It can execute a mistaken instruction correctly.

1.3.3 PLAIN — a worked example

1. Suppose two tills both read a file saying there are 10 notebooks. Till A sells two and prepares a replacement count of eight. Till B sells one and prepares a replacement count of nine. If A saves eight and B later overwrites it with nine, the final file says nine. The expected count is seven.

2. The failure is not that subtraction stopped working. Both tills subtracted from the same old starting point and one result replaced the other. Putting the same unprotected pattern behind a database connection is not, on its own, a complete correction.

3. We will later write the operation as a controlled change to the current value, with conditions and a concurrency policy. For now, recognising the overlapping sequence is more important than memorising a particular command.

1.3.4 PLAIN — what is really happening inside

1. A request to a database engine is expressed through an interface. The engine interprets the request, checks applicable rules, locates or changes records, and returns a result or an error. The physical arrangement can differ substantially between engines.

2. A CSV file describes a tabular exchange format, not a shared-editing protocol. Its commas and quotation rules do not specify how two tills coordinate updates or how an interrupted operation recovers. RFC 4180 documents common CSV conventions as an informational RFC. [S08]

3. Similarly, JSON provides a way to represent structured values. A JSON object containing an order does not by itself specify authority, concurrency, retention or whether an amount has been paid. Those are contracts above the representation. [S07]

1.3.5 TECHNICAL — the engineer's version

1. Separate four layers in a design: the **data model**, the **operations**, the **engine** and the **storage format**. The model says what entities and relationships mean. Operations specify the permitted questions and changes. The engine implements those operations. The format represents data at a storage or exchange boundary.

2. SQL means **Structured Query Language**. It is a language used by many database engines, not the name of one engine. Product-specific behaviour still matters. SQLite, for example, uses dynamic typing in ordinary tables, and its documented type-affinity rules differ from the rigid assumptions readers sometimes bring from other products. [S21]

3. The foundations laboratory uses Python and an in-memory SQLite database. It demonstrates stated examples only. PostgreSQL references explain PostgreSQL behaviour; running the SQLite laboratory is not a PostgreSQL compatibility test.

4. That separation is a habit worth developing early: name the product, the relevant configuration, the operation and the observed result before generalising.

1.3.6 WORDS — remember these

1. **File format:** how bytes are arranged; technically, the syntax and interpretation rules for a representation.

2. **Interface:** how one component asks another to do something; technically, the exposed operations and their input, output and error contracts.

3. **SQL:** a language for working with data; technically, Structured Query Language and its standardised and product-specific features.

4. **Concurrency:** work that overlaps in time; technically, the execution of operations whose interleaving can affect their observed results.

1.4 Questions produce answers, not new certainty

1.4.1 PLAIN — in simple words

1. A report is an answer calculated from records under a definition. It can be wrong even when every source record was stored correctly. The mistake may be the question, the selection of records or the calculation.

2. "How busy was the shop?" is not yet a precise question. It could mean the number of orders, the number of people served, the number of items sold or the total recorded order value. A single large order makes those measures behave differently.

3. The useful habit is to finish the sentence: "This number counts or measures , *from* , during , *excluding* ."

1.4.2 PLAIN — a picture in your head

1. Think of a recipe made from measured ingredients. The ingredients can be correct while the recipe is wrong for the dish you intended. A spoonful of sugar and a spoonful of salt are both real measurements; swapping their role ruins the result.

2. **Where the comparison stops:** a data calculation often leaves its inputs unchanged. Several different reports can legitimately be made from the same records. The issue is whether each one matches its stated definition.

1.4.3 PLAIN — a worked example

1. Our second fictional order, O-1043, contains one notebook at INR 75.50 and two pens at INR 20.00 each. Its total is 75.50 + 40.00 = INR 115.50.

2. Across O-1042 and O-1043, we have two orders, four order lines and six physical units. Their total recorded order value is 171.00 + 115.50 = INR 286.50. The average recorded value per order is 286.50 / 2 = INR 143.25.

3. Dividing by four would produce INR 71.625 per line, which is a different measure. None of these calculations establishes recognised revenue, settled receipts or profit. We have not introduced the records or definitions needed for those claims.

1.4.4 PLAIN — what is really happening inside

1. A report chooses records, combines related records when necessary, performs calculations and formats the result. Each step needs a defined meaning. "Today" requires a time boundary. "Completed" requires a business status. "Customer" requires an identity policy.

2. A dashboard may store a previously calculated answer so it can show it quickly. In that case the displayed value also has an "as of" boundary. The presence of a current-looking screen does not tell you when its underlying calculation was last refreshed.

3. For our teaching system, every derived report will have a name, an input definition, an inclusion rule and a calculation. Where a value is cached or refreshed periodically, the report will also state that timing policy.

1.4.5 TECHNICAL — the engineer's version

1. A **query** is a specified request for data or a calculation over it. A **projection**, in the broader application-design sense used here, is a derived representation built for a particular use. Do not confuse that usage with relational algebra's narrower projection operator, which selects attributes.
2. The cardinality of an input is its number of rows. A join can change that cardinality. A line-level dataset is not an order-level dataset simply because every line carries an order identifier. The case study's four rows require two distinct order identifiers to answer the order-count question.
3. Ordering is also part of the contract. PostgreSQL does not promise a particular output order unless sorting is requested. A query that happens to return O-1042 before O-1043 in a small demonstration does not establish a permanent ordering rule. [S03]
4. A report specification should distinguish its logical definition from the plan used to execute it. We will defer optimisation until the question itself is stable.

1.4.6 WORDS — remember these

1. **Query:** a precise question; technically, an operation expressing a requested result over data.
2. **Derived data:** an answer made from other data; technically, a representation produced by a transformation or calculation over inputs.
3. **Grain:** what one record stands for; technically, the observation or business unit represented by each row.
4. **Cardinality:** how many; here, the number of rows in a dataset or intermediate result.

1.5 Changes, failures and the word "saved"

1.5.1 PLAIN — in simple words

1. "Saved" can mean several things. The screen may have accepted your typing. The application may have sent a request. The database may have accepted the change. The information may also have reached storage expected to survive a particular failure.
2. Those are different milestones. A useful system tells the user which outcome it knows rather than using one reassuring word for all of them.
3. A timeout is especially important. It means an expected response did not arrive within the allowed time. From that fact alone, the caller cannot always know whether the change happened.

1.5.2 PLAIN — a picture in your head

1. You post a signed form to an office. The office records it, then sends you a receipt. The receipt is lost on the way back. You have no receipt, but the office may already have accepted the form.

2. **Where the comparison stops:** software can attach a stable request identifier and expose a status lookup. A well-designed retry protocol can use those records to avoid treating every resend as a brand-new request. The analogy describes uncertainty, not an unavoidable absence of solutions.

1.5.3 PLAIN — a worked example

1. Imagine a controlled database operation that creates O-1042 and records its associated stock reduction. If an error interrupts the operation between those changes, our intended outcome is either that all participating changes are accepted or that none are.

2. Now consider a different sequence: the database accepts the complete operation, but the network response to Dev is lost. Dev sees a timeout. Retrying as a completely new order could record the sale twice. Immediately claiming that no order exists could also be wrong.

3. The next safe question is about the original operation's identity and status. A useful request identifier makes that question possible; its existence alone still does not prove that duplicate handling was implemented correctly.

1.5.4 PLAIN — what is really happening inside

1. A transaction groups participating database changes into a controlled unit. In the model used by PostgreSQL's transaction tutorial, a successful commit accepts the unit, while rollback cancels the transaction's changes. [S01]

2. That solves a specific problem. It does not by itself explain when a response reaches a screen, how a notification is delivered, or how an already-completed transaction should be reversed as a business matter.

3. Storage promises also have conditions. A database may allow a mode that acknowledges a transaction before the corresponding recovery information has been durably written. PostgreSQL documents that an asynchronous commit can therefore be lost after a crash even though it was acknowledged. [S04]

1.5.5 TECHNICAL — the engineer's version

1. Keep four questions separate. **Atomicity:** which changes take effect together? **Isolation:** how can overlapping operations observe or affect one another? **Durability:** which accepted changes survive which failures under which configuration? **Knowledge at the caller:** what has the caller actually learned from the messages it received?

2. These questions interact, but none is a replacement for the others. A local rollback demonstration is evidence about that demonstration. It is not proof of power-loss durability, multi-machine failover, or correctness under concurrent requests.

3. Our laboratory deliberately creates a table in memory, makes a temporary change and rolls it back. It verifies that the demonstrated rollback restores the queried row count. Because the database is in memory, the laboratory makes no claim that its data survives process exit or a machine failure.

4. Later chapters will add explicit request identity, bounded retries and external-effect handling. The important lesson now is to put the guarantee next to its boundary.

1.5.6 WORDS — remember these

1. **Transaction:** a controlled unit of database work; technically, operations managed together under an engine's atomicity and related semantics.
2. **Commit:** accepting the transaction; technically, the transaction-completion operation under the engine's documented configuration.
3. **Rollback:** cancelling uncommitted work; technically, discarding the transaction changes within the requested rollback scope.
4. **Durability:** surviving specified failures; technically, the persistence guarantee for accepted changes under stated assumptions.

1.6 The map of the whole book

1.6.1 PLAIN — in simple words

1. The rest of the book expands this one shop without treating scale as magic. We first learn what records mean. Then we organise them, ask questions, control changes, survive failures and distribute work. Finally we build useful analysis and operate the system responsibly.
2. This order matters. Making an ambiguous record travel faster does not make it clearer. Copying an incorrect calculation to more machines does not make it more correct.

1.6.2 PLAIN — a picture in your head

1. Think of learning to run a workshop. Before buying faster machines, you learn what you are making, how parts fit, how to measure the result and what makes it safe to use.
2. **Where the comparison stops:** data systems can be reorganised without moving physical walls, but changes still have compatibility and recovery costs. "Software can change" does not mean "software can change without consequences."

1.6.3 PLAIN — a worked example

1. For O-1042, the early chapters ask whether the quantity means units or boxes. The database chapters ask where the order and its lines belong. The query chapters ask how to find the order quickly. The transaction chapters ask what happens when a request repeats.
2. The recovery chapters ask how the order can be restored after a failure. The distributed chapters ask which copy a reader is seeing. The analytical chapters ask what the order contributes to a report. The operational chapters ask who may inspect it and what happens when it is no longer needed.
3. It is the same order throughout. New machinery earns its place by answering a new question.

1.6.4 PLAIN — what is really happening inside

1. Each major section will use the same six reading blocks. First comes the idea in ordinary words. Then comes a mental picture, a worked example and a closer account of the mechanism. The technical block adds formal names, implementation details and limits. The final block connects new terms to the explanation you have already read.

2. You can read the plain blocks first and return for the technical blocks. You should not have to invent missing definitions to make that route work.

1.6.5 TECHNICAL — the engineer's version

1. The planned sequence has eight parts and 52 chapters. Parts A and B establish semantics and relational foundations. Parts C, D and E develop query execution, concurrency and storage. Parts F, G and H address distribution, analysis and operations.

2. Claims will be labelled by their status: a definition used in the book, a rule selected for the fictional case study, a documented product behaviour, or an observed result from a specified experiment. Those categories are not interchangeable.

3. This complete edition contains all fifty-two chapters. The companion-lab guide distinguishes executable exercises from conceptual discussions. A chapter's presence does not mean that every system it describes has been implemented, simulated or independently reviewed. Consult the recorded test scope before making a stronger claim.

1.6.6 WORDS — remember these

1. **Model:** a deliberately limited representation; technically, a set of concepts and rules chosen to answer particular questions.

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

3. **Implementation detail:** how a particular system works; technically, behaviour that must not be assumed to apply to every conforming or competing system.

4. **Reproducible example:** an example another reader can check; technically, one supplied with sufficient inputs, procedure and environment information to repeat its stated observation.

1.97 Check your understanding

Question 1. A report shows four orders because it counted the four lines in O-1042 and O-1043. Which definition was lost?

Answer. It lost the row grain. Each row represented an order line, not a complete order. There are two distinct orders, four lines and six units in the example.

Question 2. A stock count passes a non-negative-integer rule. Has the number of notebooks on the shelf been verified?

Answer. No. The representation passed the stated rule. A suitable observation or reconciliation is still needed to support the physical claim.

Question 3. Dev sees a timeout after sending O-1042. What can he conclude?

Answer. He did not receive the expected response in time. The example does not give enough evidence to conclude either that the order was accepted or that it was not. He needs an outcome check tied to the original operation.

Question 4. The laboratory demonstrates rollback in memory. Does that demonstrate crash durability?

Answer. No. The experiment tests the stated rollback result in its own environment. It does not exercise durable storage or a power failure.

1.98 Common wrong ideas

"A stored record must be true." Storage preserves representations. It does not independently establish their correspondence with the world.

"A database removes the need to design rules." The engine enforces the rules and operations actually supplied, subject to its documented behaviour.

"A correct total means the whole process is correct." A total can be arithmetically correct while using the wrong records, units or business definition.

"A timeout means nothing happened." The response can be lost after an operation succeeds.

"A successful toy test proves production readiness." A test supplies evidence only for the claim, inputs, environment and failure conditions it actually exercised.

1.99 Chapter summary in 20 lines

1. A data system connects actions, records, rules and questions.
2. O-1042 contains two notebooks and one pen.
3. Its recorded order value is INR 171.00 under our stated assumptions.
4. One order, two lines and three units are different counts.
5. A record is a preserved statement, not automatic proof of an event.
6. Valid structure and accurate observation are different properties.
7. A discrepancy identifies disagreement before it identifies a cause.
8. An application implements a workflow and requests data operations.
9. A DBMS manages database operations under documented rules.
10. A format describes a representation, not an entire operational contract.
11. Overlapping changes need a concurrency policy.
12. A report needs an explicit definition of what it measures.
13. Derived answers inherit the assumptions and limits of their inputs.
14. Output order should be requested when it matters.
15. The word "saved" hides several possible milestones.
16. A timeout alone may leave the outcome unknown.
17. A transaction groups participating database changes.
18. Durability claims require stated failure and configuration boundaries.
19. An experiment demonstrates only what it actually exercises.
20. The next chapter makes the meaning of each record explicit.

2

CHAPTER

What a Record Really Means

2.0 What this chapter gives you

This chapter teaches you to look at a record and ask the questions that a screen full of neat columns can hide. What is being claimed? What does one row stand for? Which thing does an identifier refer to? Who supplied the information, and which moment does its time describe?

You will turn an ambiguous line of text into an explicit teaching record. You will also learn why keeping more fields is not the same as keeping better evidence, and why a correction needs a meaning as well as a new value.

The running example remains Mira's Corner. O-1042 is the order for two notebooks and one pen, totalling INR 171.00 under the simplified pricing rules established in Chapter 1. We will not quietly change those assumptions halfway through an example.

2.1 The claim inside a record

2.1.1 PLAIN — in simple words

1. Consider this line: `1042, notebook, 2, 75.50, 09:30`. It looks informative, but important parts of its meaning are missing. Is 1042 an order, a customer or a delivery? Does two mean two notebooks or two boxes? Is 75.50 the price of one item or the amount for the whole line? Which day does 09:30 belong to?

2. A record becomes useful when the reader can recover the intended claim without guessing. A useful sentence might be: "Order O-1042, line 1, records an agreed quantity of two individual notebooks at INR 75.50 per notebook."

3. That sentence is already doing design work. It separates the order, the line, the item, the quantity, the unit and the price basis.

2.1.2 PLAIN — a picture in your head

1. Imagine receiving a parcel with only the number "2" on its label. It might mean two items, a second attempt, a shelf number or a delivery priority. A better label does not change the parcel; it tells the next person how to treat it.

2. **Where the comparison stops:** a data field's meaning can depend on rules stored somewhere else, such as a schema or data dictionary. You do not need to repeat a long explanation in every row, but the explanation must be available and connected to the row's version.

2.1.3 PLAIN — a worked example

1. We can make the first line of O-1042 explicit:

Field	Example value	Meaning in this book
order_id	O-1042	The shop's identifier for this order
line_no	1	This line's number within the order
product_id	P-NOTE	The notebook product in our catalogue
quantity	2	Two individual saleable units
unit_price_minor	7550	Agreed price per unit in paise
currency	INR	The currency attached to the amount

2. For this case study, one rupee is represented by 100 paise. The line amount is $2 \times 7550 = 15100$ paise, displayed as INR 151.00. Storing `7550` without stating "paise per unit" would not preserve the full claim.

3. The record says what was agreed for the order line. It does not, on its own, say that two notebooks were handed over or that a payment completed. Those require their own definitions and evidence.

2.1.4 PLAIN — what is really happening inside

1. The producer and consumer of a record share a contract. That contract says how to read the fields, which values are allowed and what business meaning the combination has.

2. A field name can help, but names are not a complete contract. `amount` is vague. `unit_price_minor` is better, but it still needs a currency, a definition of the minor unit and a rule about when the price was agreed.

3. The data dictionary is where we gather these meanings. W3C's data guidance distinguishes structural metadata from descriptive information that helps people understand and use data. In our book, the dictionary supplies both where appropriate. [S06]

2.1.5 TECHNICAL — the engineer's version

1. Model the example as a predicate: `AgreedOrderLine(order_id, line_no, product_id, quantity, unit_price_minor, currency)`. Read that as a statement with named arguments, not as runnable code.

2. The predicate's interpretation contains facts the type system alone cannot express. For example, `quantity` counts saleable individual units in this exercise. A later product sold by mass would need a different quantity-and-unit policy; reusing the integer field without revisiting its meaning would be a modelling error.

3. Separate the **syntactic contract** from the **semantic contract**. Syntax tells a parser where a field begins and ends. Semantics tells an application what the value means and which operations are meaningful. A JSON number can parse correctly while being interpreted in the wrong unit. [S07]

4. For each field, document its domain, unit or scale, missing-value policy, source, mutability and version boundary. These are proposed design requirements for our case study, not properties acquired merely by choosing JSON or SQL.

2.1.6 WORDS — remember these

1. **Field:** one named part of a record; technically, an attribute position with an agreed interpretation.

2. **Domain:** the allowed kind of value; technically, the set of permitted values and associated meaning for an attribute.

3. **Syntax:** the shape of a representation; technically, the grammar used to recognise valid constructions.

4. **Semantics:** what a representation means; technically, the interpretation and permitted consequences assigned to it.

2.2 Fields, values and the contract around them

2.2.1 PLAIN — in simple words

1. A label and a value travel together, but they are not the whole story. The same text can mean different things under different rules. A date written as `03/04/2026` may be read as 3 April or 4 March. Neither reader has necessarily made an arithmetic mistake; they may be using different conventions.

2. A safe exchange tells the receiver which convention applies. When that information is missing, "pick the interpretation that looks likely" is not a reliable general rule.

3. The goal is to make the contract explicit before a guess turns into a stored fact.

2.2.2 PLAIN — a picture in your head

1. Picture a board game with pieces but no instructions. You can see the pieces clearly and still have no idea whether moving diagonally is allowed. A schema describes much of the board and the pieces; the full application contract also describes the permitted play.

2. **Where the comparison stops:** a schema may itself encode some rules, while others live in application logic or procedures. There is no universal split that makes every important rule belong in exactly one place.

2.2.3 PLAIN — a worked example

1. Here is the first order line as a JSON object. JSON is a text format for exchanging structured values. The braces enclose an object; each quoted name is followed by its value. The structure below is an original teaching example, not a complete production message specification. [S07]

```
{
  "record_type": "agreed_order_line",
  "schema_version": 1,
  "order_id": "O-1042",
  "line_no": 1,
  "product_id": "P-NOTE",
  "quantity": 2,
  "unit_price_minor": 7550,
  "currency": "INR"
}
```

2. Suppose another program sends the same field names but uses `75.50` to mean rupees. It has not followed our contract. Silently accepting the value and treating it as paise would produce a different price.

3. The receiver should reject or explicitly transform that message under a documented input contract. It should not decide the unit by whether a number contains a decimal point.

2.2.4 PLAIN — what is really happening inside

1. A receiver first recognises the representation, then checks its structure and values. In our example, it expects a supported record type and schema version. It checks required fields and rejects values outside the contract.

2. A version number is useful only when it points to an actual definition. Writing `schema_version: 1` does not make an undocumented format self-explanatory. Nor should a receiver automatically accept version 2 because it understood version 1.

3. When a producer changes a field's meaning, readers need a compatibility plan. Changing an amount from paise to rupees under the same name and version would break that agreement even if every message still parsed.

2.2.5 TECHNICAL — the engineer's version

1. Define four outcomes for an incoming teaching record: accepted without transformation; accepted through a documented transformation; rejected with a precise reason; or held for review because its meaning is unresolved. The review path must not quietly feed uncertain values into ordinary totals.

2. For `quantity`, our contract requires a positive integer, not just something a language can coerce to one. The included Python validator deliberately rejects booleans as quantities, even though Python's Boolean type has an integer relationship. This is our validator's domain policy, demonstrated by a test. [S22]

3. Unknown fields also need a policy. An extensible analytics feed may preserve them. A tightly bounded command may reject them. Our small validator rejects unexpected field names so a misspelling such as `unit_prcie_minor` does not silently disappear.

4. The contract should also define how invalid inputs are reported. "Bad data" is not as useful as "quantity must be a positive integer." The latter helps the sender repair the actual defect without exposing unrelated records.

2.2.6 WORDS — remember these

1. **Schema:** the declared structure; technically, a definition of fields, types, relationships and applicable structural rules.

2. **Schema version:** which contract applies; technically, an identifier for a defined revision of the schema and its interpretation.

3. **Parser:** a shape reader; technically, software that recognises a representation according to its grammar.

4. **Coercion:** automatic conversion; technically, a transformation between types that may preserve or change intended meaning.

2.3 One row means one what?

2.3.1 PLAIN — in simple words

1. Before counting rows, finish this sentence: "One row in this dataset represents one ___." This is the row's **grain**.

2. An order row and an order-line row are not interchangeable. A product row describes a catalogue item. A stock-count row describes an observation of an item at a location and time. All can mention the same product without representing the same kind of fact.

3. When different grains are mixed, a table can look tidy while its totals become misleading.

2.3.2 PLAIN — a picture in your head

1. Imagine counting a classroom from two lists. The attendance list has one line per pupil. The timetable has one line per lesson attended by a pupil. Counting timetable lines does not count pupils unless you deliberately remove the repetition.

2. **Where the comparison stops:** repetition is not always a defect. The timetable genuinely needs several records for the same pupil because each record stands for a different lesson. The design question is whether repetition matches the intended grain.

2.3.3 PLAIN — a worked example

1. Our order-line dataset contains these four rows:

Order	Line	Product	Units	Line amount, paise
O-1042	1	P-NOTE	2	15100
O-1042	2	P-PEN	1	2000
O-1043	1	P-NOTE	1	7550
O-1043	2	P-PEN	2	4000

2. There are four rows because the grain is one agreed order line. There are two orders because only O-1042 and O-1043 appear as order identifiers. There are six units because $2 + 1 + 1 + 2 = 6$.

3. Now imagine that each row also repeats its complete order total. Adding that repeated column would count each two-line order twice: $17100 + 17100 + 11550 + 11550 = 57300$ paise. The correct combined order value is 28650 paise.

4. The stored numbers need not be individually false for the calculation to be wrong. The error comes from aggregating at the wrong grain.

2.3.4 PLAIN — what is really happening inside

1. A query works with the rows it receives. If a combination operation produces several rows for one order, a later sum sees several rows. The engine does not automatically know that a repeated order total should count only once.

2. The repair is not a universal instruction to remove duplicates. Two different orders can legitimately have the same total. Deduplicating the amount itself could remove a real order from the calculation.

3. Instead, decide which identity defines the things being counted and at which grain each amount is meaningful. We will use separate order-level and line-level structures when the relational model is built in later chapters.

2.3.5 TECHNICAL — the engineer's version

1. The sample line identity is `(order_id, line_no)`. The quantity and agreed unit price describe that line. An order total describes the order, so its natural aggregation boundary is different.
2. A **functional dependency** says that one set of attributes determines another under the data model. For our case study, the order-line identifier determines the values for that accepted line version. It does not imply that product identity alone determines an agreed historical price.
3. A product's current catalogue price may change tomorrow. Replacing yesterday's agreed line price with today's catalogue price would change the meaning of the historical order. Whether to preserve a copied price, reference a versioned price record or use another design is a later modelling decision; the requirement is to retain the correct historical meaning.
4. A basic test can expose the grain mistake: compute totals from the four line amounts and compare them with totals from the two order headers. Agreement is a useful check in this fixture, not a guarantee that every other report is correct.

2.3.6 WORDS — remember these

1. **Grain:** what one row represents; technically, the unit of observation or business fact represented at the dataset's chosen level.
2. **Aggregation:** combining several records into an answer; technically, a calculation over a defined group of inputs.
3. **Functional dependency:** one part determines another; technically, a constraint under which equal determining attributes imply equal dependent attributes.
4. **Historical value:** a value tied to an earlier event or agreement; technically, information whose interpretation must not be replaced by an unrelated current-state value.

2.4 Identity without guesswork

2.4.1 PLAIN — in simple words

1. An identifier lets us refer to a particular thing without redescribing it every time. It answers "which one?" It does not automatically answer "is this genuine?" or "is this the same real-world person?"
2. Two orders can contain the same items and amounts and still be different orders. One order can be delivered twice as a message and still represent only one order. Matching visible values does not settle the distinction.
3. We need to define the thing being identified and the scope within which its identifier is unique.

2.4.2 PLAIN — a picture in your head

1. A seat number is useful inside a particular theatre and performance. "Seat 12" alone may not identify where you should sit. You may also need the row, venue and performance.
2. **Where the comparison stops:** a software identifier can be designed to have a broader scope. Even then, a broad identifier does not authenticate the person presenting it or prove that two separately created records describe different real-world entities.

2.4.3 PLAIN — a worked example

1. At Mira's single shop, `O-1042` identifies our first order. Later a second branch starts its own sequence and also creates `O-1042`. If both identifiers were only promised to be unique within their own branch, the collision is a failure of the combined interpretation, not necessarily a broken local sequence.

2. One solution is to identify the order by the pair `(branch_id, order_id)`. Another is to introduce an identifier allocated across the whole system. We will compare designs later; for now, either requires an explicit scope.

3. For the current one-shop fixture, line 1 of O-1042 and line 1 of O-1043 are different lines. `line_no` alone is not their full identity.

2.4.4 PLAIN — what is really happening inside

1. The system can enforce a uniqueness rule over a chosen set of stored values. That rule is valuable, but its meaning depends on the columns chosen.

2. If every incoming message receives a fresh database identifier, the database may contain perfectly unique rows for the same repeated business event. The uniqueness rule succeeded at row identity while the business deduplication rule was never supplied.

3. Conversely, collapsing all records with the same product, quantity and amount could merge two genuine purchases. Duplicate detection requires a definition of "same event," not simply a desire for fewer rows.

2.4.5 TECHNICAL — the engineer's version

1. Distinguish **row identity**, **business identity** and **delivery identity**. Row identity selects a stored record. Business identity selects the real or administrative occurrence represented. Delivery identity selects a particular transmission or attempt. Our design may intentionally map several deliveries to one business event.

2. In PostgreSQL, a primary key can span several columns and requires unique, non-null values for the key. That supports the composite line identity used in our model. It does not infer whether two different keys refer to the same underlying purchase. [S02]

3. A deduplication decision also needs a conflict path. Two deliveries claiming the same business identifier but carrying different quantities should not automatically be treated as identical replays. The system must apply a documented rule or retain the conflict for review.

4. For now, the safe mental model is that an identifier is a reference under a contract. Its format, scope, allocation and lifecycle all belong in that contract.

2.4.6 WORDS — remember these

1. **Identifier:** a reference to a particular thing; technically, a value or tuple interpreted within a defined namespace and lifecycle.

2. **Composite key:** an identifier made from several parts; technically, a key consisting of more than one attribute.

3. **Namespace:** where a name has its meaning; technically, the scope within which identifiers are interpreted and uniqueness is defined.

4. **Deduplication:** recognising repeated representations of the same thing; technically, a policy-driven process whose identity assumptions and conflict handling must be explicit.

2.5 Source, event time and recording time

■ 2.5.1 PLAIN — in simple words

1. A record has a history. Somebody or something produced it, possibly from earlier records. Knowing that history can help you assess what it supports and how to reproduce it.

2. Time is part of that context. The time an event happened may differ from the time its record reached the system. An offline till can record a sale at the counter and upload it later.

3. Calling both moments simply `time` makes later questions unnecessarily difficult.

■ 2.5.2 PLAIN — a picture in your head

1. A letter has a date written inside and a date stamped when it reaches an office. The first describes what the sender says about timing; the second describes an observed step in delivery. You may need both to answer a question fairly.

2. **Where the comparison stops:** neither timestamp guarantees a perfectly accurate clock. The date inside may also describe a plan rather than an occurrence. The field's meaning and the clock's limitations still matter.

■ 2.5.3 PLAIN — a worked example

1. For our fictional order, assume the till records an occurrence time of `2026-09-01T09:30:00+05:30`. The receiving system records arrival at `2026-09-01T09:37:00+05:30`.

2. Under the example's shared, accurate-clock assumption, the difference is seven minutes. A report asking when the order occurred uses the first timestamp. A report asking how long the feed was delayed compares the two.

3. A third report might ask what the central office knew at 09:35. At that point the order had occurred in the story but had not yet arrived in the central dataset. Rebuilding that historical view requires more than the occurrence time alone.

■ 2.5.4 PLAIN — what is really happening inside

1. The receiver can preserve an event-time field supplied by the source and create a separate recording-time field itself. It can also record the source identifier and import batch so a suspicious record can be traced.

2. This is a practical form of **provenance**: information about how a record or result came to exist. W3C's PROV model distinguishes entities, activities and agents, and represents relationships such as derivation. Our small shop metadata borrows that distinction without claiming to implement the complete PROV standard. [S05]

3. A source label is a starting point for investigation, not a certificate of truth. A wrongly configured till can send consistently wrong times with a perfectly consistent source identifier.

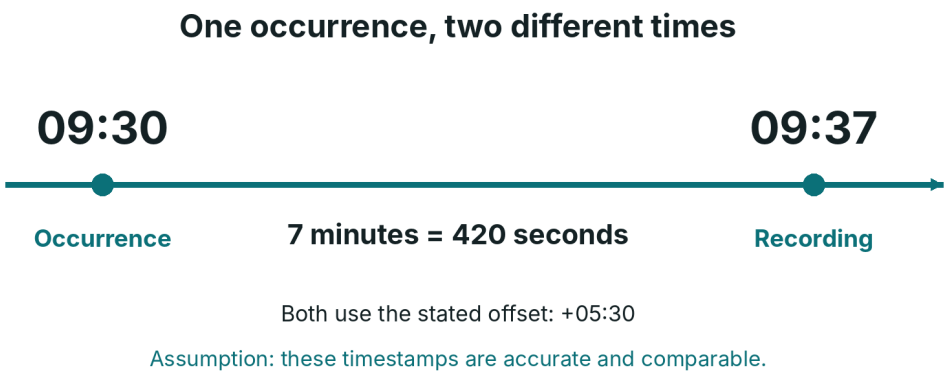

Figure 2.2: Figure 2.1. One occurrence, two relevant times. The timeline assumes the clocks are accurate and comparable; the record itself does not prove that assumption.

▪ 2.5.5 TECHNICAL — the engineer's version

1. Use explicit field semantics such as `occurred_at`, `recorded_at` and `source_system`. Their exact names are a case-study convention. `recorded_at` should state which receiving boundary it describes: initial receipt, validation completion or database commit are not necessarily the same moment.

2. Distinguishing event time from recording time introduces the need for multiple temporal perspectives. Two timestamp columns alone do not implement a full bitemporal database or preserve every revision needed for historical queries.

3. The numeric offset in our timestamp identifies a relationship to UTC for that representation. RFC 3339 defines a widely used Internet timestamp syntax with such offsets. It does not make a source clock accurate, and a numeric offset is not the same thing as a named time zone with rules. [S09]

4. For a derived report, provenance should connect the output to its input set and transformation version. An output label such as "sales report" is not enough to reconstruct which records were included or which rule produced the result.

▪ 2.5.6 WORDS — remember these

1. **Event time:** when the described occurrence happened; technically, the timestamp assigned to the event under its source's semantics.

2. **Recording time:** when a defined system boundary recorded the information; technically, a timestamp whose acquisition point must be specified.

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

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

2.6 Corrections and the limits of evidence

2.6.1 PLAIN — in simple words

1. Suppose a quantity was entered incorrectly. Replacing the value can make the current record more useful, but it may erase the explanation for why yesterday's report differed from today's.
2. A correction should say what claim changed and why the change was accepted. It should not pretend that the corrected record was always what the system knew.
3. This does not mean every system must retain every old value forever. Keeping history has a purpose and a lifecycle. The requirement is to choose the policy deliberately and describe its limits honestly.

2.6.2 PLAIN — a picture in your head

1. On a paper form, a correction might cross out one value, write the replacement and add a note. The visible change helps a later reader understand the discrepancy.
2. **Where the comparison stops:** digital records can have access controls, version histories and multiple derived copies. Preserving an old value in one place does not ensure that every report or export has been corrected, and indefinite preservation can itself be inappropriate.

2.6.3 PLAIN — a worked example

1. This is a separate correction exercise, not a change to the canonical O-1042 fixture. Imagine a draft order D-2001 with quantity three, later verified as quantity two. At a unit price of 7550 paise, the draft line amount was 22650 paise and the corrected amount is 15100 paise.
2. The difference is 7550 paise. If a report already included the draft amount, simply changing the current row does not tell us whether that report refreshed. We need to identify which outputs depend on the changed record.
3. A useful correction note might record the affected draft, old and new quantities, the reason "entry corrected against the retained source slip," the approving role, and the time of the decision. The note describes evidence used in this hypothetical example; it does not prove the source slip was itself accurate.

2.6.4 PLAIN — what is really happening inside

1. The system applies a chosen correction policy. It might update a draft in place while keeping an authorised revision history. It might append a correcting event. It might prevent direct editing after a particular business transition and require a separate reversal workflow.
2. Which policy is appropriate depends on what the record means. Correcting a spelling mistake in a draft label is not necessarily the same operation as reversing an accepted commercial obligation.
3. Derived outputs need their own response: refresh, correction notice, versioned replacement or an explicit statement that a previous export cannot be recalled. A deletion from the current screen does not tell us what happened to every retained copy.

2.6.5 TECHNICAL — the engineer's version

1. Separate **current-state correctness**, **historical explainability** and **retention policy**. A design may improve one while damaging another. For example, overwriting a value simplifies the current row but can destroy the information needed to explain an earlier output.
2. The case-study correction model will record the identity of the subject, the scope of the change, the asserted reason, the permitted actor and the affected revision. It will avoid treating an unverified reason code as established causation.
3. Provenance relationships can connect a corrected output with its predecessor and the activity that produced it. The PROV model provides concepts for such relationships; it does not dictate our business authorisation or retention policy. [S05]
4. The acceptance test must match the claim. A test that verifies the latest row says two does not also prove that historical reports are reproducible, permissions are correct or all external copies have changed. Those require separate observations.

2.6.6 WORDS — remember these

1. **Revision:** a particular state after a change; technically, an identifiable version of a record or artefact.
2. **Correction:** a change to repair a stated defect; technically, an authorised transition whose scope, evidence and consequences should be recorded.
3. **Audit trail:** information used to explain actions; technically, a record of relevant operations and context, with its own integrity and retention requirements.
4. **Retention:** how long and why information is kept; technically, a policy governing continued storage and eventual disposition for defined record classes.

2.97 A record-design exercise

You receive this export from a fictional third-party till:

```
id,item,qty,amount,time
1042,Notebook,2,151,09:30
1042,Pen,1,20,09:30
```

Task. Write down what can be read directly and what remains unknown. Do not fill the gaps from the familiar numbers in our running story; in this exercise the sender has not supplied that context.

Worked answer. The file contains two data rows with the same `id`, different item labels, quantities two and one, amounts 151 and 20, and matching time strings. That describes the representation, not its full business interpretation.

We do not yet know what `id` identifies, whether the amount is per unit or per line, which currency and scale apply, what units the quantities count, which date and offset accompany the time, or whether the records describe requests, accepted orders or completed handovers. We also lack a source record identifier and a duplicate-delivery policy.

A suitable repair is to obtain the sender's contract, then transform the records under that contract while retaining appropriate import provenance and rejected-input evidence. It is not to guess INR because the values happen to resemble our previous example.

Second task. Design a field dictionary for an agreed order line.

Worked answer. State the line grain and composite identity first. Define product reference, quantity unit, agreed unit price, currency, schema version and missing-field rules. Define source and temporal fields according to the actual workflow. A field is not required merely because it is fashionable; it is required when the stated use needs it.

2.98 Common wrong ideas

"Clear field names eliminate the need for a contract." Names help, but they do not fully define units, timing, scope, missingness or permitted changes.

"A unique database ID proves there are no duplicate business events." Separate stored rows can refer to the same event.

"Repeated values should always be removed." Repetition can be legitimate at the chosen grain. Two different orders can have the same total.

"A timestamp tells us everything about time." It needs a defined meaning and a clock context. Occurrence, receipt and commit are different boundaries.

"An audit record proves the recorded explanation." It preserves an asserted explanation and relevant context. The explanation still needs suitable evidence.

2.99 Chapter summary in 20 lines

1. A useful record makes an interpretable claim.
2. Values without units and context can be ambiguous.
3. Field names support a contract but do not replace it.
4. Syntax describes form; semantics describes meaning.
5. A data dictionary connects values to their intended interpretation.
6. A schema version must identify a real definition.
7. Parsing successfully does not prove domain validity.
8. A transformation should be explicit rather than silently guessed.
9. The row grain states what one record represents.
10. Counting lines is not the same as counting orders.
11. Aggregating repeated order totals can multiply the answer.
12. Historical prices must retain their historical meaning.
13. Identifiers operate within defined scopes.
14. Row identity and business-event identity can differ.
15. A repeated delivery and a repeated purchase require different treatment.
16. Event time and recording time answer different questions.
17. Provenance helps explain where information came from.
18. Provenance does not automatically certify truth.
19. Corrections need meaning, authority and an explicit history policy.
20. The next chapter explains the values that fields can carry.

3

CHAPTER

Numbers, Text, Time and Missing Values

3.0 What this chapter gives you

A database cannot make a field meaningful if the application has chosen the wrong kind of value. This chapter explains the everyday decisions hidden inside "just store it": whether something is a count or a label, whether a fraction must be exact, what a character means, which kind of time we are recording, and how to preserve the difference between zero and not known.

You will calculate integer limits, inspect a floating-point surprise, compare two representations of the same visible letter, read an offset timestamp and reason about SQL NULL. The included laboratory repeats the numerical and textual examples with synthetic inputs in an isolated environment.

Code blocks are optional on the first reading. Each one is accompanied by an explanation of what it demonstrates. A printed result is not a promise that every language or database implements the same behaviour.

3.1 Choose a type before choosing a format

3.1.1 PLAIN — in simple words

1. A **type** describes the kind of value a field can carry and the operations that make sense for it. A number of notebooks is a count. An order identifier is a label. Both may contain digits, but adding one to them has different meanings.

2. For the count, adding one can mean another notebook. For the label `0017`, converting it to the number 17 may throw away a distinction the sending system needs. A field should not become numeric merely because its current examples contain only digits.

3. Start by asking what the value represents. Then choose the representation and the rules that preserve that meaning.

3.1.2 PLAIN — a picture in your head

1. Think of drawers marked "quantities," "names" and "dates." Putting a name in the quantity drawer does not make arithmetic on the name useful. Labels help prevent inappropriate operations before they happen.

2. **Where the comparison stops:** programming languages differ in how strongly they enforce types and how readily they convert values. The drawer analogy describes the purpose of a type, not a guarantee that every language locks the drawers.

3.1.3 PLAIN — a worked example

Value	Meaning in an example	Suitable starting model
12	Counted notebooks on a shelf	Non-negative integer count
0017	Identifier supplied as a four-character label	Text with a documented identifier rule
7550	Unit price in paise for our fixture	Integer amount plus currency and scale
2026-09-01	A calendar date	Date, not a made-up midnight instant
09:30 at offset +05:30	A local clock reading with an offset	Needs a date to identify the example's instant
Not yet counted	Absence of an observation	Missing-value state, not the number zero

1. The same visible digits can participate in different models. A product named "100" is still a product label. A genuine measured quantity may require fractions rather than a whole-number type. The meaning decides the direction of the design.

3.1.4 PLAIN — what is really happening inside

1. A program reads bytes and interprets them as values. It may convert text to a number, validate the range and store a representation chosen by its engine. Each conversion is a place where information can be preserved, rejected or lost.

2. An import that turns `0017` into 17 cannot reconstruct the original four-character spelling from 17 alone without another rule. If the field's contract treats the spelling as significant, the conversion has lost information.

3. Conversely, storing every value as text postpones rather than solves the problem. A later calculation still needs to know whether "12" is a quantity, whether "12.0" is allowed and whether an empty string means missing or invalid.

3.1.5 TECHNICAL — the engineer's version

1. Separate the **domain type** from the **storage type**. `OrderId`, `StockCount` and `UnitPriceInPaise` are domain concepts. `text` and `integer` are possible storage types. Several domain concepts can share a storage type while having different valid operations.

2. For our fixture, `StockCount` permits zero, while an agreed order line's `quantity` must be positive. Both are integer-valued, but their domains differ. A type alone may therefore need a constraint. PostgreSQL documents column and table constraints as a way to express additional restrictions. [S02]

3. Conversions should have explicit failure behaviour. Reject a quantity of `two`, distinguish an unsupported format from an out-of-range value, and do not truncate a fractional quantity to make it fit a whole-unit contract.

4. SQLite's ordinary dynamic typing is a product-specific reminder not to infer all runtime behaviour from a declared type name. The laboratory uses explicit validation for its teaching contract rather than claiming that every engine will enforce identical types. [S21]

3.1.6 WORDS — remember these

1. **Type:** the kind of value; technically, a classification that governs representation and permitted or defined operations.
2. **Domain type:** the business meaning of the value; technically, a model-specific type whose rules may exceed the underlying storage type.
3. **Storage type:** the engine's representation category; technically, a supported type or storage class with documented operations and limits.
4. **Lossy conversion:** a change that forgets information; technically, a mapping from which the original value cannot always be reconstructed.

3.2 Integers and the units they count

3.2.1 PLAIN — in simple words

1. An integer is a whole number: zero, one, two, and their negative counterparts. Integers are useful for counting indivisible units and for representing amounts in a deliberately chosen smallest unit.
2. Whole does not mean unlimited. A fixed-size representation has only so many available patterns. You must know both what each step counts and how large the value can become.
3. For our order prices, one step counts one paisa. For notebook stock, one step counts one notebook. They should not be added to each other merely because both are integers.

3.2.2 PLAIN — a picture in your head

1. Imagine a mechanical counter with three decimal wheels. It has room for 000 through 999. A fourth digit cannot appear without a design change or some defined response to exceeding the limit.
2. **Where the comparison stops:** computers use binary patterns and different signed representations. Some languages can allocate more space for larger integers; some fixed-size operations reject an excessive value, and some have other specified overflow behaviour. Always name the actual environment.

3.2.3 PLAIN — a worked example

1. A bit has two possible values. With eight bits there are $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 256$ patterns. An unsigned interpretation can use them for 0 through 255: 256 distinct integers, not 255.
2. A 32-bit signed two's-complement interpretation has the range -2,147,483,648 through 2,147,483,647. PostgreSQL's four-byte `integer` type has that range and rejects values outside its range. [S10]
3. If a positive price amount uses that maximum as a count of paise, it represents INR 21,474,836.47. The calculation is 2,147,483,647 / 100. Changing the unit changes the represented amount without changing the bit limit.
4. For O-1042, the calculation remains comfortably within this example range:

```
2 × 7550 + 1 × 2000 = 17100 paise
17100 paise / 100 = INR 171.00
```

3.2.4 PLAIN — what is really happening inside

1. The representation stores a pattern interpreted under a rule. "Signed" means the chosen interpretation includes negative values. The unit comes from the surrounding contract, not from a special "money" property in each bit.

2. Multiplication and addition can produce results larger than their inputs. Choosing a type that holds one unit price is not enough if the same type must also hold the total for many items or a long reporting period.

3. The review question is therefore about the maximum intermediate result as well as the maximum stored input. In a toy fixture, the answer may be simple. In a real workload, it needs a stated bound and behaviour when that bound is exceeded.

3.2.5 TECHNICAL — the engineer's version

1. For an unsigned n-bit representation, the range is 0 through 2^n - 1. For an n-bit signed two's-complement representation, it is -2^(n-1) through 2^(n-1) - 1. These expressions count finite bit patterns; they do not claim that all language integer types use a fixed n.

2. Python's integers in the laboratory can grow beyond a fixed 32-bit range, subject to available resources. That means a passing arithmetic example in Python does not test whether a PostgreSQL `integer` column would accept the same enormous result. Our explicit range demonstration calculates the bound rather than claiming to exercise a PostgreSQL server. [S22]

3. The unit should be carried by naming, documentation and, where practical, domain-specific types. Converting a minor-unit amount to display text should not change the stored value. Combining currencies is outside the fixture's permitted operations; an amount tagged INR is not interchangeable with the same integer tagged another currency.

4. Integer minor units are not a universal answer to all price calculations. Proportions and allocations can produce fractions of the chosen unit. The next section introduces the rounding policy needed at that boundary.

3.2.6 WORDS — remember these

1. **Integer:** a whole number; technically, a value in the set of signed whole numbers, represented with implementation-specific limits.

2. **Range:** the smallest through largest supported values; technically, the domain bounds of a representation or declared type.

3. **Overflow:** exceeding a representation's capacity; technically, an operation whose mathematical result lies outside the representable range.

4. **Scale:** what one stored step represents; here, the fixed relationship between an integer amount and its displayed unit.

3.3 Decimals, fractions and rounding

3.3.1 PLAIN — in simple words

1. Some fractions fit a number system neatly and others do not. In decimal notation, one half is 0.5, but one third continues as 0.333… without ending. Binary arithmetic has a different set of neatly fitting fractions.

2. This is why a computer may show a tiny surprise when adding familiar decimals with a binary floating-point type. The machine is operating on nearby representable values, not on an infinitely precise version of every number you typed.

3. The solution is not "never use floating point." It is to choose a representation and an error or rounding policy suitable for the task.

3.3.2 PLAIN — a picture in your head

1. A ruler marked only in whole millimetres cannot directly express every possible distance. A point halfway between two marks needs an approximation or a more detailed measuring method.

2. **Where the comparison stops:** floating-point spacing changes with magnitude, unlike the evenly spaced marks on that ruler. The analogy explains limited representation, not the exact layout of floating-point numbers.

3.3.3 PLAIN — a worked example

1. In the Python environment used by the supplied laboratory:

```
print(0.1 + 0.2)
```

2. The displayed result is:

```
0.30000000000000004
```

3. Python's floating-point tutorial explains why common decimal fractions are approximated in binary and why display formatting can hide or reveal that approximation. This is a representation issue, not evidence that addition is randomly unreliable. [S12]

4. Now construct decimal values directly from decimal text:

```
from decimal import Decimal

print(Decimal("0.1") + Decimal("0.2"))
```

5. The result in the supplied example is `0.3`. Constructing the values from strings is deliberate: it avoids first introducing a binary floating-point approximation. Decimal arithmetic still has a precision context and rounding rules; it does not make every fraction finitely representable. [S13]

3.3.4 PLAIN — what is really happening inside

1. The chosen numeric representation determines which values are exact and how operations produce rounded results. Formatting determines what part of the result is shown. These are different layers.

2. Displaying a total with two decimal places does not prove that the underlying calculation used an appropriate representation or rounding policy. It can merely hide a discrepancy.

3. Consider dividing 100 paise equally among three recipients. Each mathematical share is 33 and one third paise. Under our whole-paisa output rule, three displayed shares of 33 paise add to only 99 paise. One paisa remains to allocate. An explicit rule might give the extra paisa to the first recipient in a defined ordering, producing 34, 33 and 33. That is a selected allocation policy, not an arithmetic law.

3.3.5 TECHNICAL — the engineer's version

1. Distinguish **representation error**, **measurement error** and **business rounding**. Representation error comes from storing or computing with an approximation. Measurement error concerns how closely a measured value reflects the relevant quantity. Business rounding is a deliberate policy for mapping an amount to an allowed unit. Treating all three as "rounding" conceals different causes.

2. PostgreSQL provides exact `numeric/decimal` types as well as floating-point types. Declared precision and scale affect what is stored. A field's type choice should be tied to its required calculations and boundaries rather than copied from an unrelated system. [S10]

3. A robust calculation specification names the input units, intermediate precision, rounding rule, rounding stage and reconciliation rule. Rounding each line and then summing is not always the same operation as summing exact intermediates and rounding once.

4. The laboratory's `allocate_minor_units(100, 3)` example uses integer division and remainder to produce `[34, 33, 33]`. It verifies that the shares sum to 100 and differ by at most one. The rule selects recipients by list position; it does not claim to be the correct policy for every allocation problem.

3.3.6 WORDS — remember these

1. **Floating point:** a number representation with a movable scale; technically, a finite significand and exponent representation with defined rounding behaviour.

2. **Decimal arithmetic:** arithmetic based on decimal representation; technically, operations over decimal values under a precision and rounding context.

3. **Quantisation:** fitting values to allowed steps; technically, mapping values to a discrete set of representable or permitted outputs.

4. **Rounding policy:** the chosen way to handle non-exact results; technically, the rule and stage used to select an allowed output value.

3.4 Text, Unicode and bytes

3.4.1 PLAIN — in simple words

1. A computer stores a representation of text, not tiny printed letters. A text system needs an agreement about which abstract characters are represented and how to turn that sequence into bytes.

2. Unicode provides a shared character framework. UTF-8 is one way to encode Unicode text as bytes. They are related concepts, not two names for the same layer. A Unicode scalar value takes from one to four bytes in UTF-8. [S14]

3. The number of bytes is therefore not generally the number of visible characters. That distinction matters when storing names, limiting input length or cutting a string into pieces.

3.4.2 PLAIN — a picture in your head

1. Imagine a catalogue assigning a number to each symbol, and a packing rule for sending those numbers in small envelopes. The catalogue is one layer; the packing rule is another. Some symbols need more envelope space than others.

2. **Where the comparison stops:** a visible character can be formed from several Unicode code points. The reader's "one character" is not always one catalogue entry, and rendering may combine or shape entries according to the writing system.

3.4.3 PLAIN — a worked example

1. The Latin letter `A` is one code point, U+0041. The rupee sign ₹ is one code point, U+20B9. In UTF-8 their byte sequences differ in length:

Text	Code point	UTF-8 bytes in hexadecimal	Byte count
A	U+0041	41	1
₹	U+20B9	E2 82 B9	3

2. The supplied laboratory checks these sequences directly. Hexadecimal, or base 16, is just a compact way to write the byte values; it does not change the text.

3. Now compare the visible letter "é." It can be represented by U+00E9, or by U+0065 followed by U+0301, a combining acute accent. The first sequence has one code point and the second has two. Unicode defines canonical equivalence and normalisation forms for cases such as this. [S15]

3.4.4 PLAIN — what is really happening inside

1. A decoder turns valid encoded bytes into a text representation. A renderer draws the resulting text using fonts and shaping rules. These are not the same operation: being able to decode a character does not guarantee that a particular font draws it well.

2. A program comparing code-point sequences can find our two forms of "é" unequal even when they look alike. Applying an appropriate normalisation policy can make canonically equivalent forms comparable. It does not translate a name, decide whether two people are the same, or make all similar-looking characters identical.

3. When a user asks for "20 characters," the product should decide what that limit is for. A storage byte limit, a code-point count and a user-visible editing limit are different requirements. Unicode's text-segmentation guidance defines grapheme-cluster boundaries relevant to user-perceived characters. [S16]

3.4.5 TECHNICAL — the engineer's version

1. The laboratory uses Python strings and UTF-8 encoding. `len("₹")` is 1 in that environment, while `len("₹".encode("utf-8"))` is 3. Neither result alone defines a universal user-interface character-count policy.

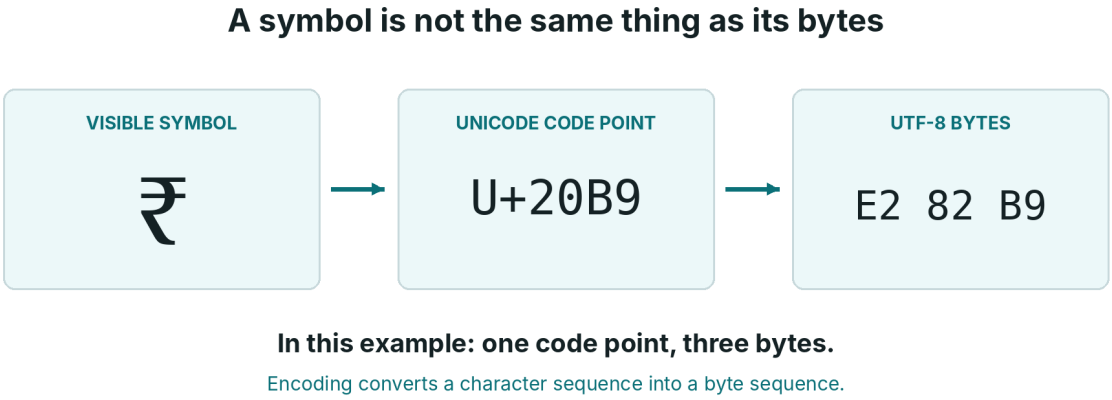

Figure 3.3: Figure 3.1. The rupee sign passes through three different descriptions: visible symbol, Unicode code point and UTF-8 byte sequence.

2. For the two acute-e forms, Python reports code-point counts of one and two. Their UTF-8 lengths are two and three bytes. Normalising each with NFC makes the two resulting strings equal in the demonstrated case. The equality before normalisation is false; the equality after it is true. [S15]

3. Normalisation is not the same operation as case folding, language-sensitive sorting or identity matching. A system that needs all of them should specify their order and purpose rather than hide them behind a vague "clean text" step.

4. At an import boundary, invalid encoding needs a deliberate policy. Silent replacement can make a file displayable while losing the original distinction that an identifier depended on. The laboratory uses strict decoding behaviour for its bounded example and verifies that an invalid UTF-8 byte sequence is rejected.

■ 3.4.6 WORDS — remember these

1. **Code point:** an assigned position in the Unicode code space; technically, a numeric value used in the character model, not a guarantee of one visible glyph.

2. **Encoding:** the mapping into bytes; technically, a defined representation of a character sequence for storage or interchange.

3. **Grapheme cluster:** a unit closer to what a reader perceives as a character; technically, a text-segmentation unit defined by applicable Unicode rules and possible tailoring.

4. **Normalisation:** a specified way to standardise equivalent text sequences; technically, conversion to a Unicode normalisation form, not general identity verification.

3.5 Dates, instants and durations

■ 3.5.1 PLAIN — in simple words

1. Not every value involving time is the same kind of thing. A birthday is a calendar date. A completed sale has an occurrence instant. "The shop opens at 09:00" is a local schedule. "The upload took seven minutes" is a duration.

2. Treating all four as an arbitrary text field creates ambiguity. Treating all four as midnight in a time zone can create a different ambiguity: a birthday was not necessarily meant to identify an instant at all.

3. Choose the temporal meaning before choosing the storage representation.

3.5.2 PLAIN — a picture in your head

1. A calendar, a wall clock and a stopwatch answer different questions. A calendar tells you which date. A wall clock gives a local time of day. A stopwatch measures elapsed time.

2. **Where the comparison stops:** computers can combine these concepts and convert between representations, but a conversion cannot recover context that was never supplied. A bare 09:30 does not identify one instant in history.

3.5.3 PLAIN — a worked example

1. Read this timestamp from left to right:

```
2026-09-01T09:30:00+05:30
```

2. It states the date 1 September 2026, local clock time 09:30:00, and a numeric offset five hours and thirty minutes ahead of UTC. For this representation, subtracting that offset gives:

```
2026-09-01T04:00:00Z
```

3. The two strings describe the same instant. The `Z` form uses UTC. The syntax is within the timestamp format described by RFC 3339. [S09]

4. Now compare occurrence at 09:30 and recording at 09:37 with the same date and offset. Under the example's comparable-clock assumption, the interval is 420 seconds. The laboratory verifies the conversion and subtraction for these inputs.

3.5.4 PLAIN — what is really happening inside

1. The program interprets the date, clock fields and offset, then can compare the resulting instants. A named time zone adds rules associated with a place; a numeric offset only describes an offset for the representation in front of you.

2. For a historical event, an instant representation may be sufficient for ordering and duration calculations, with additional source context retained when needed. For a recurring local opening time, "09:00 in the shop's local time zone" is a different requirement from "the same UTC time forever."

3. Two source clocks can disagree. Extra decimal places on a timestamp do not establish that the clock was accurate to that precision. A latency calculation between independent systems therefore needs assumptions about the clocks or another measurement method.

3.5.5 TECHNICAL — the engineer's version

1. PostgreSQL distinguishes `date`, `timestamp without time zone` and `timestamp with time zone`. For the latter, input is normalised to an instant and output is displayed using the session's time-zone setting; the original named zone is not retained in that value. The type's name should not be read as a promise to preserve all original zone context. [S11]

2. Use a date for a date-only concept. Use an instant-capable representation for an occurrence when an actual instant is required. Preserve a named zone separately when the application needs that original context or future local scheduling rules. A local recurring schedule is not fully represented by one past offset.

3. Duration arithmetic also has a domain boundary. "Add 24 elapsed hours" and "same local time tomorrow" need not be interchangeable in every zone and calendar situation. Our 420-second example avoids those complications by using two explicit offsets on the same date.

4. The Python laboratory checks the stated examples using its standard datetime facilities. It is not a complete RFC 3339 conformance test, a time-zone database certification or a clock-synchronisation experiment.

3.5.6 WORDS — remember these

1. **Date:** a calendar position; technically, a calendar day without necessarily specifying a time of day or an instant.

2. **Instant:** one point on a timeline; technically, a temporal position distinguishable from its local display representation.

3. **UTC offset:** the difference from UTC in a representation; technically, a signed hours-and-minutes offset, not a complete named-zone rule set.

4. **Duration:** elapsed amount of time; technically, a measured or specified interval whose units and arithmetic semantics must be stated.

3.6 Missing is not zero

3.6.1 PLAIN — in simple words

1. Zero is a value. "We have not counted yet" is a lack of a value. Replacing the second with the first changes the claim from "not known" to "known to be none."

2. An empty text value, a field that was omitted, an explicit missing marker and the word "unknown" are also different representations. A system needs a policy for them rather than assuming they all mean the same thing.

3. The policy should match the purpose. Sometimes a missing value should reject a record. Sometimes it should be preserved so the record remains honest about what is not known.

3.6.2 PLAIN — a picture in your head

1. Imagine four envelopes labelled "refund amount." One contains a slip saying zero. One contains a slip saying 100. One is empty. One has a note saying the envelope was never requested.

2. You cannot add the empty envelope as zero without making an extra assumption. **Where the comparison stops:** SQL NULL is a specific database marker with defined expression behaviour. It does not automatically preserve every possible reason for absence. A separate status may be needed when the reason matters.

3.6.3 PLAIN — a worked example

1. Suppose four fictional observations contain values 100, 200, missing and 0. The sum of the known values is 300. There are four rows, but only three known values. The mean of the known values is 300 / 3 = 100.

2. Replacing missing with zero would produce an average of 300 / 4 = 75. That is not another spelling of the same answer. It is a result under a different assumption.

3. In the laboratory's SQLite example, the query is:

```
SELECT
    COUNT(*) AS row_count,
    COUNT(value) AS known_count,
    AVG(value) AS mean_known
FROM observations;
```

4. For the supplied fixture, the result is `(4, 3, 100.0)`. The same distinction between counting rows and counting non-null values is documented for PostgreSQL's aggregate functions. [S19]

3.6.4 PLAIN — what is really happening inside

1. In ordinary SQL comparisons, a NULL operand generally produces an unknown result rather than a true equality or inequality. Asking `value = NULL` is not the way to find missing values; use `value IS NULL`. PostgreSQL documents this distinction explicitly. [S17]

2. Think of an unknown sealed-box quantity. You cannot say that it equals five, but you also cannot conclude that it does not equal five. The comparison is unknown. A filter keeps rows whose condition is true, so those unknown comparison results do not behave like true matches.

3. This also explains why changing `value > 0` to `NOT (value > 0)` does not automatically include missing values. Negating unknown still does not establish a known true condition. The missing case needs to be handled according to the question being asked. [S18]

3.6.5 TECHNICAL — the engineer's version

1. The relevant three-valued logic has true, false and unknown. Here are selected operations; "unknown" is shown as a word rather than confused with a stored text value.

Expression	Result
NOT unknown	unknown
true AND unknown	unknown
false AND unknown	false
true OR unknown	true
false OR unknown	unknown

2. These operations are consistent with PostgreSQL's documented logical tables. They do not imply that evaluation order is a safe substitute for controlling side effects in expressions. [S18]

3. A check constraint and a missing-value rule are also distinct. In PostgreSQL, `CHECK (quantity > 0)` alone does not reject NULL: a check is satisfied by true or null. Add `NOT NULL` when absence is forbidden. [S02]

4. JSON has a `null` literal, while an absent object member is structurally different. Neither representation carries a universal business explanation for missingness. Our line validator requires all its fields; it does not use null as a substitute for an agreed quantity. [S07]

5. For reporting, publish the denominator. "Mean among three known observations, one missing" is more informative than showing 100 with no population or missingness context. The arithmetic cannot decide whether the missing case should be excluded, estimated or treated as an error; that is part of the analytical contract.

3.6.6 WORDS — remember these

1. **NULL:** a specific SQL missing/unknown marker; technically, a marker with defined behaviour in comparisons, logic, constraints and aggregates.

2. **Empty string:** text containing no characters; technically, a zero-length string, distinct from SQL NULL in the engines discussed here.

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

4. **Missingness policy:** what absence means and how it is handled; technically, the domain rules for rejecting, preserving, classifying or analysing unavailable values.

3.97 Practice, then inspect the evidence

Exercise 1: the identifier. A source contract defines `0017` as a four-character order label. An import turns it into the integer 17. What was lost?

Worked answer. The original representation's leading zeros were lost. Whether that changes business identity depends on the source contract, but this contract explicitly required the four-character label. The receiver should not perform that conversion silently.

Exercise 2: the allocation. Split 100 paise among three recipients using the book's rule: integer base shares, then give one extra paisa to the earliest recipients until the remainder is exhausted.

Worked answer. Integer division gives a base share of 33 and a remainder of one. The shares are 34, 33 and 33. They sum to 100. A different recipient ordering may allocate the extra paisa differently, so the ordering must be part of the policy.

Exercise 3: the timestamp. Do `2026-09-01T09:30:00+05:30` and `2026-09-01T04:00:00Z` identify different instants?

Worked answer. No. The first is five and a half hours ahead of UTC in its local representation. Subtracting the offset yields the second.

Exercise 4: the average. Four rows contain 100, 200, NULL and 0. Explain both 100 and 75 without calling them equivalent.

Worked answer. The mean among three known values is 100. Replacing NULL with zero produces a mean of 75 over four values. The latter requires an additional assumption that the missing value should count as zero. The source rows alone do not justify it.

Exercise 5: the text. Two strings look like "é," but one has one code point and the other two. Does this prove a corruption?

Worked answer. No. They may be the canonically equivalent forms demonstrated in the chapter. Inspect their code points and apply the relevant text policy rather than judging identity only by appearance.

3.98 Common wrong ideas

"Digits mean the field should be numeric." Identifiers can contain digits while remaining labels with significant spelling and scope.

"Integers cannot have numerical limits." Fixed-size representations have finite ranges. A language with growing integers does not remove a database column's limit.

"Showing two decimals makes a calculation correct." Display formatting does not define representation, intermediate precision or rounding policy.

"Decimal arithmetic makes every fraction exact." Some fractions still require rounding under a finite precision context.

"One visible character is one byte." Text has distinct byte, code-point and user-perceived segmentation layers.

"Store every time as UTC and all temporal questions disappear." Dates, instants, local schedules and original zone context remain different requirements.

"Unknown means zero or false." Missingness and three-valued logic must be handled according to the actual domain and query.

3.99 Chapter summary in 20 lines

1. Choose a value's meaning before choosing its representation.
2. An identifier containing digits is not automatically a quantity.
3. Domain types can be more specific than storage types.
4. A conversion can lose information while appearing convenient.
5. Integers count whole steps under a stated unit.
6. Fixed-size integer representations have finite ranges.
7. Intermediate calculations need range analysis too.
8. Minor-unit amounts still need currency and scale context.
9. Binary floating point cannot represent every familiar decimal exactly.
10. Decimal arithmetic also has precision and rounding boundaries.
11. Formatting is not a substitute for a calculation policy.
12. Allocations need a rule for indivisible remainders.
13. Unicode and UTF-8 describe different layers of text representation.
14. Bytes, code points and grapheme clusters are different counting units.
15. Normalisation is not general identity verification.
16. A calendar date is not the same concept as an instant.
17. A numeric UTC offset is not a complete named-zone rule set.
18. Missing, zero and empty text must not be silently equated.
19. SQL NULL changes comparison, logical and aggregate behaviour.
20. Clear types preserve meaning before a system ever becomes large.

4

CHAPTER

Measurements, Units and Uncertainty

4.0 What this chapter gives you

1. You will turn "we measured it" into a statement that names the object, quantity, unit, method and time.
2. You will convert units without accidentally changing the quantity, and use units to catch impossible calculations.
3. You will distinguish a repeatable instrument from an accurate result and a detailed display from genuine knowledge.
4. You will work through an average, a sample standard deviation and the uncertainty of an average without skipping the meaning of the arithmetic.
5. You will explain why missing observations and timeouts can change what a report is allowed to claim.
6. You will carry uncertainty through a calculation and state the assumptions behind the resulting interval.

A new fixture, not a change to the orders

Mira's Corner now weighs parcels and records how long work takes. These are new synthetic measurement exercises. They do not change O-1042, O-1043 or the notebook discrepancy introduced earlier. The shop still has no evidence establishing the cause of that discrepancy.

Our main measurement fixture is **M-BOX-01**: five repeated indications for the same sealed test box, in grams: **998, 1000, 1001, 999, 1002**. These are repeated measurements of one box, not the masses of five different boxes. An instrument comparison, a gross-minus-tare exercise and a request-timing exercise will be introduced separately and labelled when used.

4.1 What exactly was measured?

4.1.1 PLAIN — in simple words

1. A number does not tell you what happened until you know what it measures. "The parcel is 1,000" could describe grams, a price, a product code or an item count.
2. Even "the parcel weighs 1,000 grams" leaves a question: does that include the box, tape and packing material, or only the goods inside?
3. Decide what you are trying to measure before deciding where to store the answer. Otherwise two careful people can report different numbers while measuring different things.
4. Keep the difference between a direct reading and a calculated result. Subtracting the packaging mass produces a new result from two inputs; it is not another direct reading of the same display.

4.1.2 PLAIN — a picture in your head

1. Imagine two people measuring a journey. One starts a stopwatch when the driver leaves home. The other starts when the vehicle joins the main road. Their stopwatches can both work perfectly and still disagree.
2. The missing piece is an agreed starting line. A useful measurement definition supplies both the starting and finishing lines, even when the quantity is not time.
3. **Where this comparison breaks:** some quantities, such as a parcel's mass, are not journeys. Their equivalent boundaries concern the object, its condition and the procedure. Agreement about boundaries does not remove instrument error.

4.1.3 PLAIN — a worked example

1. In a separate teaching exercise, the scale indicates **1,052 g** for a packed parcel and **52 g** for its empty packaging under the stated procedure.
2. Gross mass includes the packaging. Tare mass is the packaging mass being subtracted. The estimated net mass is therefore **1,052 - 52 = 1,000 g**.
3. A report that compares 1,052 g from one branch with 1,000 g from another without checking gross versus net has mixed definitions, not discovered a heavier product.
4. The input record below makes one observation's meaning visible. Its identifiers are invented, and the method code points to our own teaching procedure, not a recognised certification.

Field	Example	What it prevents you from guessing
observation_id	M-GROSS-01	Which observation is being discussed
object_id	PARCEL-TEST-01	Which physical test object was measured
quantity_kind	gross_mass	Whether packaging is included
indicated_value	1052	What the instrument displayed
unit	g	The size of each numerical unit
method_version	PACKED-01	Which measurement procedure was followed
recorded_at	2026-09-02T10:00:00+05:30	When this example was recorded

5. This table is not a complete laboratory record. A real procedure might also require instrument identity, environmental conditions, calibration information and a separate measurement time.

4.1.4 PLAIN — what is really happening inside

1. A measurement process connects an object to an instrument, an instrument indication to a recorded value, and that value to a documented interpretation.
2. Software often sees only the last part. It cannot recover an omitted packaging rule by looking harder at the integer 1052.
3. The international metrology vocabulary calls the quantity intended to be measured the **measurand**. Naming it includes enough information about the quantity and object to distinguish the intended measurement. [S26]
4. For the shop, we propose a method register. Every observation refers to a supported method version. The register explains boundaries, units, permitted instruments and what to do when a reading is unavailable.

5. A changed method receives a new version. That lets a later report separate a genuine trend from a change in how the shop measured the work.

4.1.5 TECHNICAL — the engineer's version

1. Distinguish the physical quantity, the instrument indication, the stored representation and the estimator used in a report. A database column may contain any one of these; the column name alone is insufficient evidence.
2. A **measurement model** specifies how inputs produce a result. In our gross-minus-tare exercise, write `net = gross - tare`. This equation also exposes which inputs need compatible units and uncertainty information.
3. The observation's **grain** is one execution of the stated procedure on one object. The record is not automatically one object forever: repeated observations legitimately share the same object identifier.
4. Keep raw input where the purpose and retention policy justify it, but distinguish it from interpreted data. For example, the received string `"1052"` and the interpreted value `1052 g` are not the same representation or the same claim.
5. A practical schema separates a value from its status. `status = not_measured` with no value is different from `status = measured` with value zero. A type-valid number accompanied by the wrong method version should not silently enter a comparable series.
6. Provenance can describe the source entities, the activity that transformed them and the agent associated with that activity. W3C PROV provides a vocabulary for such relationships; our small method register is not a claim to implement all of PROV. [S05]
7. The honest version: more metadata does not repair a badly defined measurand. Start with a question that people can execute consistently, then preserve the information required to interpret the answer.

4.1.6 WORDS — remember these

1. **Measurand:** the quantity you intend to measure; technically, the quantity defined as the target of a measurement, with the necessary object and condition information.
2. **Indication:** what the instrument shows; technically, the quantity value supplied by a measuring instrument or system.
3. **Gross mass:** goods plus the included packaging; technically, mass under a definition that includes the stated container and packing materials.
4. **Tare mass:** the packaging amount taken away; technically, the measured or assigned container mass used in a net-mass calculation.

4.2 Units and dimensional checks

4.2.1 PLAIN — in simple words

1. A unit says how big one step in a number is. One kilogram and one gram are different steps for describing the same kind of quantity.
2. Converting units changes the number, not the object. A mass written as 1.25 kg is written as 1,250 g after conversion.

3. Units are also a way to check reasoning. Adding a mass to a duration does not become sensible just because a spreadsheet accepts both as numbers.
4. Two values can share a physical unit and still mean different things. A gram of packaging and a gram of goods have compatible dimensions, but whether they should be added depends on the question.

4.2.2 PLAIN — a picture in your head

1. Think of the same distance drawn on two maps. One map uses centimetres on the page for kilometres outside. Another uses a different scale. The road has not moved.
2. A conversion factor is the rule connecting the scales. Forget that rule and identical-looking numbers describe different distances.
3. **Where this comparison breaks:** many unit conversions are simple multiplication, but not all are. Temperature scales can include an offset, and a currency exchange rate is not a fixed physical conversion factor.

4.2.3 PLAIN — a worked example

1. Convert the parcel mass: **1.25 kg × 1,000 g/kg = 1,250 g**. The kilograms cancel in the written calculation, leaving grams.
2. Now convert an area. A rectangular label is 20 cm by 30 cm, so its area is **600 cm²**.
3. One centimetre is 0.01 m. Therefore one square centimetre is **0.01 m × 0.01 m = 0.0001 m²**.
4. The label area is consequently **600 × 0.0001 = 0.06 m²**. Dividing by 100 instead of 10,000 would give the wrong area.
5. Finally, a synthetic request takes 400 ms. Since 1 ms is 0.001 s, its duration is **0.4 s**. Recording the number 400 under a column labelled seconds creates a thousandfold error.

Calculation	Resulting unit	Interpretation
grams + grams	grams	A sum of compatible masses
grams / seconds	g/s	A mass flow rate, if the model supports it
requests / seconds	requests/s	A throughput measure under a stated request definition
centimetres × centimetres	cm^2	An area, not a length
order IDs + order IDs	None defined here	Arithmetic on these labels has no business meaning

4.2.4 PLAIN — what is really happening inside

1. The computer usually stores a number separately from the unit's meaning. It will happily calculate 400 + 1 even when the first number means milliseconds and the second means kilograms.
2. A safe boundary checks a quantity's kind and unit before combining it with another quantity. It converts accepted inputs into a chosen internal unit, while preserving enough metadata to explain that conversion.
3. SI prefixes specify decimal multipliers. For example, kilo is 10^3, milli is 10^{-3} and micro is 10^{-6}. Their letter case matters: `m` and `M` are not interchangeable prefix symbols. [S27]
4. The shop's `quantity` field for O-1042 still counts individual notebooks or pens. It must not quietly become a mass field when the shop starts weighing parcels. Those are different contracts.

4.2.5 TECHNICAL — the engineer's version

1. A **quantity equation** expresses a relationship independently of a chosen unit system. `area = length × width` remains meaningful whether lengths are expressed in metres or centimetres.
2. A **numerical value equation** may include conversion factors tied to the units actually used. `area_m2 = length_cm * width_cm / 10000` is one such implementation for our rectangular-label model.
3. Dimensional analysis can reject inconsistent operations, but it is not a complete semantic checker. Counts of orders and counts of lines are both dimensionless physical counts; their business grains remain different.
4. Consider a result labelled "average request duration." Dividing total duration by request count gives a time per request. Dividing by successful-request count instead gives a different denominator unless only successful durations were included.
5. For a linear unit conversion `y = c × x`, an uncertainty expressed in the same measurement sense scales by `abs(c)` as well. Converting 1,000 g with standard uncertainty 2 g produces 1.000 kg with standard uncertainty 0.002 kg; it does not improve the measurement. [S31]
6. Store conversion rules deliberately. A measurement service might allow `g` and `kg` as input for mass, convert both to grams, and reject `ms` in that field. Rejecting the wrong dimension is better than applying a guessed conversion.
7. Standard, convention or implementation detail: the SI prefix multipliers are standardised; our decision to store parcel mass in grams is a design convention; the exact validation code is our implementation.

4.2.6 WORDS — remember these

1. **Unit:** the agreed size of a numerical step; technically, a reference quantity used to express values of quantities of the same kind.
2. **Conversion factor:** the multiplier between compatible units; technically, the ratio used to transform numerical values while preserving the represented quantity.
3. **Dimension:** the physical kind of a quantity; technically, its expression in powers of the chosen base quantities, such as length or time.
4. **Dimensional analysis:** checking a calculation through its units; technically, checking compatibility of dimensions across an equation while recognising that semantics require further checks.

4.3 Precision versus accuracy

4.3.1 PLAIN — in simple words

1. A scale can give nearly the same reading every time and still be consistently off target. Repeatability and correctness are not the same achievement.
2. A display with three decimal places can also be misleading. The extra digits show how finely the instrument reports a result, not how close the result is to the quantity being measured.
3. In everyday words, precision concerns how closely repeated measurements agree under stated conditions. Accuracy concerns closeness to the true value, which is often not directly available to us. [S28] [S29]
4. That is why a comparison needs a suitable reference and information about that reference's own uncertainty. One unverified scale is not automatically the truth against which another should be judged.

4.3.2 PLAIN — a picture in your head

1. Imagine throwing paper balls into a bin. A tight cluster on the floor beside the bin is repeatable but displaced. A scattered set around the bin has a different problem.
2. The cluster's spread suggests one kind of improvement; its displacement suggests another. "Throw more times" does not necessarily move a consistently displaced cluster into the bin.
3. **Where this comparison breaks:** a bin gives a visible target. In measurement the true quantity is generally not known exactly. The reference procedure and its uncertainty must be part of the comparison.

4.3.3 PLAIN — a worked example

1. For a separate synthetic instrument check, assign a reference value of **1,000 g**, with reference uncertainty set aside temporarily so we can inspect the arithmetic.
2. Scale A reads **1008, 1008, 1009, 1007, 1008 g**. Its average is **1,008 g**, which is 8 g above that assigned reference.
3. Scale B produces our M-BOX-01 readings: **998, 1000, 1001, 999, 1002 g**. Their sum is 5,000 g and their average is **1,000 g**.
4. Scale A's readings are more tightly grouped, while Scale B's average is closer to the assigned reference in this small exercise. Neither conclusion alone establishes a complete instrument specification.
5. For M-BOX-01, subtract the mean from each reading: **-2, 0, 1, -1, 2 g**. Square those differences and add: **$4 + 0 + 1 + 1 + 4 = 10\ g^2$**.
6. Divide by **n - 1 = 4**, then take the square root. The sample standard deviation is **sqrt(10/4) ≈ 1.581 g**.
7. This spread describes the five readings under the stated conditions. It is not a certificate that the box's mass lies within 1.581 g of 1,000 g.

4.3.4 PLAIN — what is really happening inside

1. Repeated observations can vary because of repositioning, environmental changes, instrument behaviour and other effects. Some effects change between readings; others are shared by all of them.
2. Averaging may reduce the effect of independent, zero-centred variation. It does not automatically remove a shared offset, such as a scale that consistently reads too high.
3. A calibration establishes a relationship between indications and reference values with associated uncertainties. Adjusting the instrument changes it. These are connected activities, but not synonyms. [S30]
4. When storing an average, also retain the sample count, relevant spread and method information where justified. Without them, another reader cannot tell whether "1,000 g" came from one reading or a carefully defined repeated-measurement procedure.

4.3.5 TECHNICAL — the engineer's version

1. For observations `x1 … xn`, the sample mean is `xbar = sum(xi) / n`. The sample variance is `s² = sum((xi - xbar)²) / (n - 1)`, requiring at least two observations.
2. Under an independent, identically distributed model with a finite common variance, this sample variance estimates the variance of individual observations. Independence and stable conditions are assumptions to investigate, not consequences of collecting many rows.

3. Under the corresponding independent-observation model, the estimated standard uncertainty of the mean is `s / sqrt(n)`. For M-BOX-01 it is **1.58113883 / sqrt(5) ≈ 0.70710678 g**. [S32]
4. Do not divide an individual reading's uncertainty by sqrt(n) merely because there are n rows somewhere in the database. The reduction applies to the particular average under its statistical model, not to every future box.
5. If all readings share an uncertain offset, write each as `xi = m + b + ei`: the target mass m, shared offset b and varying component ei. Averaging reduces the independent part of the ei terms, while the shared b remains.
6. The VIM treats measurement accuracy as a qualitative concept, not a quantity assigned a numerical value. A statement such as "accuracy = 99.9%" needs an independently defined metric; it is not the formal definition of measurement accuracy. [S28]
7. **Resolution** concerns the smallest change that can be distinguished in the stated context. Quantisation to a whole gram can hide smaller changes, while an instrument displaying tenths of a gram can still have an error larger than one gram. [S33]
8. The honest version: five neat values do not establish long-term stability, independence, freedom from bias or performance across an instrument's entire range.

4.3.6 WORDS — remember these

1. **Precision:** repeated readings agreeing closely; technically, closeness among replicate indications or measured values under specified conditions.
2. **Accuracy:** closeness to what is actually being measured; technically, closeness of agreement between a measured value and a true value of the measurand, not a numerical quantity in VIM usage.
3. **Sample standard deviation:** the observed spread of a sample; technically, the square root of the sum of squared deviations from its mean divided by n - 1.
4. **Resolution:** the smallest distinguishable change; technically, the change in the measured quantity that causes a perceptible change in the corresponding indication under the stated conditions.
5. **Calibration:** comparing indications with suitable references; technically, establishing the specified relationship between reference values and indications with their associated uncertainties.

4.4 Rounding and significant figures

4.4.1 PLAIN — in simple words

1. Rounding shortens a representation. It does not improve a measurement and can discard information needed by a later calculation.
2. A rounding rule needs both a target precision and a way to handle values exactly halfway between two choices.
3. Do not let the display format silently become the stored evidence. A report may display one decimal place while the controlled calculation retains more digits.
4. Extra digits should not imply extra confidence. A computer can calculate a very long decimal from uncertain inputs; the input uncertainty does not disappear inside the arithmetic.

4.4.2 PLAIN — a picture in your head

1. Think of cutting a length of ribbon to the nearest marked centimetre. Once you cut it, the discarded part cannot be recovered from the new length alone.
2. Rounding at every intermediate step is like trimming the ribbon repeatedly before knowing the final use. The accumulated change may matter.
3. **Where this comparison breaks:** software can preserve both the original value and its rounded display. The loss occurs when a rounded value replaces the information required for later reasoning.

4.4.3 PLAIN — a worked example

1. Use exact decimal inputs for this exercise, not binary floating-point approximations. Round each to two decimal places.

Exact input	Half-even	Half-up
2.345	2.34	2.35
2.355	2.36	2.36
1.245	1.24	1.25

2. In half-even, a halfway case chooses the neighbour with an even final retained digit. In half-up, a halfway case is rounded away from zero. The examples above are positive. [S13]
3. Take two amounts of **1.245**. Rounding each half-up first gives **1.25 + 1.25 = 2.50**.
4. Adding first gives **1.245 + 1.245 = 2.490**; rounding that sum gives **2.49**. The difference is 0.01, entirely explained by the location of rounding.
5. A business or measurement procedure must choose which operation it intends. "Both used rounding" does not make the two pipelines equivalent.

```
from decimal import Decimal, ROUND_HALF_EVEN, ROUND_HALF_UP

value = Decimal("2.345")
step = Decimal("0.01")
print(value.quantize(step, rounding=ROUND_HALF_EVEN))  # 2.34
print(value.quantize(step, rounding=ROUND_HALF_UP))    # 2.35
```

4.4.4 PLAIN — what is really happening inside

1. Quantisation maps many possible inputs to a smaller set of reported values. Several nearby masses may all appear as 1,000 g on a whole-gram display.
2. A calculation cannot determine which of those original values occurred from the rounded reading alone. It needs more evidence or an explicitly stated model.
3. Significant figures are a reporting convention for indicating the digits being retained. The spelling `1000` by itself does not establish the uncertainty, measurement method or number of reliable digits.
4. Prefer an explicit result with its unit and uncertainty statement where those distinctions matter. A value written as `1.000 kg` without further information is still not a complete measurement report.

4.4.5 TECHNICAL — the engineer's version

1. In the Python decimal model, `quantize` produces a value with the exponent of the supplied operand, subject to the decimal context. Constructing `Decimal` from the string `"2.345"` avoids importing a prior binary floating-point approximation. [S13]

2. The precision of a decimal arithmetic context is not the physical precision of an instrument. One controls numerical representation and arithmetic; the other concerns repeated measurement behaviour.
3. Write the pipeline explicitly: `raw observation → checked unit → calculation → final rounding → display`. Record any mandatory intermediate rounding as part of the calculation's definition.
4. For an ideal rounding-to-nearest device with step q, the rounding error lies within a half-step interval under that model. Modelling it as uniformly distributed gives standard uncertainty **q / sqrt(12)**, because the half-width is q/2. [S34]
5. With q = 1 g, that component is about **0.288675 g**. This is an assumed quantisation model, not a measured total uncertainty for every one-gram scale.
6. Do not add that component again if the same effect has already been included in another uncertainty estimate. A longer uncertainty budget is not automatically a better one; correlated or duplicated components can misstate the result.
7. For this book, retain sufficient intermediate digits and round the final uncertainty and result together. The exact reporting rule belongs to the stated procedure; a blanket "always keep two decimals" is not a substitute.

4.4.6 WORDS — remember these

1. **Rounding:** reporting a nearby value on a chosen scale; technically, mapping a value to a specified representable set under a tie-breaking rule.
2. **Half-even:** choosing the even final digit at a tie; technically, rounding halfway cases to the nearest value whose retained least significant digit is even.
3. **Quantisation:** replacing a continuous range with steps; technically, mapping input values to discrete output levels.
4. **Significant figures:** the digits deliberately retained in reporting; technically, a numerical presentation convention that does not alone specify a full uncertainty model.

4.5 Sampling and missing observations

4.5.1 PLAIN — in simple words

1. The records you can see may not represent everything you want to describe. Busy periods, broken devices and failed requests can be missing for reasons connected to the result.
2. A report of successful operations is not automatically a report of all attempted operations. The excluded cases may be exactly the cases that most need attention.
3. More rows can make an answer more repeatable while leaving its selection problem untouched. A million readings from the wrong population still describe the wrong population.
4. Always name what was eligible, what was observed and what was excluded before interpreting the average.

4.5.2 PLAIN — a picture in your head

1. Suppose Mira asks only customers who completed a purchase whether checkout was easy. People who abandoned a long queue are absent from the answers.
2. The survey may accurately describe the people who answered. The problem arises when the headline claims to describe everyone who tried to buy something.

3. **Where this comparison breaks:** missing telemetry is not always a voluntary decision like leaving a queue. It can arise from an instrument fault, a filtering rule or a storage failure. The cause must be investigated rather than assumed.

4.5.3 PLAIN — a worked example

1. Introduce a separate synthetic timing fixture: **20 requests** are attempted. Twelve complete successfully, with completed-request durations summing to **24 seconds**. Eight reach a **10-second timeout** without an observed completion.
2. The average among successful completions is **24 / 12 = 2 seconds**. That is a legitimate descriptive result for those twelve observations.
3. It is not the average completion time of all twenty requests. The eight missing completion times cannot be replaced by zero.
4. If each timed-out request eventually completes, and the timing definition confirms it had not completed within those ten seconds, its completion duration is at least ten seconds. Under those assumptions, the all-request average completion duration is at least **(24 + 8 × 10) / 20 = 5.2 seconds**.
5. Some requests might never complete. In that case a finite mean completion duration for all twenty is not established at all.
6. The report should therefore separate **20 attempts**, **12 observed successes**, **8 timeouts** and **2 seconds mean among successes**, rather than compressing them into "average request time: 2 seconds."

4.5.4 PLAIN — what is really happening inside

1. A collection pipeline decides which events become records. A sensor may sample once a minute. An application may log only successful responses. A report may remove rows with missing values.
2. Each choice changes the set of observations available to the calculation. A database cannot infer all the missing events simply because it can count the rows that arrived.
3. Distinguish a missing value from a missing row. A row marked `timed_out` preserves evidence of an attempt. A pipeline that drops the whole attempt hides even the denominator.
4. In our shop's reports, inclusion and exclusion counts will travel with the result. That is a design rule for this case study, not a claim that every analytical product supplies those counts automatically.

4.5.5 TECHNICAL — the engineer's version

1. Separate the **target population**, the **sampling frame**, the selected sample and the actually observed sample. The first is what the question concerns; the others describe how evidence was acquired.
2. Missing observations can depend on the measurement itself. Slow operations are especially likely to be absent from a dataset containing only successes observed before a deadline. Treating that dataset as a random sample of all durations is an additional assumption, and a poor one for this fixture.
3. The unknown completion durations in the example are **right-censored** at ten seconds under the stated observation protocol: we have a lower boundary, not an exact completion value. The toy lower-bound calculation does not fit a survival model.

4. A client's observed waiting time is a different metric. If it deliberately stops waiting at ten seconds, its recorded wait can be ten seconds even though server-side completion is later or never observed. State which clock and endpoint the metric describes.
5. Weighting also changes a question. Giving every branch equal weight estimates an average of branch results; weighting by each branch's request count estimates a request-level result under compatible definitions. Neither choice is universally correct.
6. SQL aggregate functions commonly distinguish missing values from observed zero. Our earlier SQLite fixture illustrates the practical consequence: omitting an unknown from a mean and treating it as zero give different answers. Reuse that lesson rather than hiding the exclusion behind a clean chart. [S19] [S24]
7. A useful report contract contains numerator, denominator, time window, time basis, missing-value treatment and exclusions. Later Chapter 46 will develop selection bias and analytical interpretation further.

4.5.6 WORDS — remember these

1. **Target population:** everything the question is meant to describe; technically, the defined set of units about which an inference or descriptive claim is intended.
2. **Sampling frame:** the set from which observations can be selected; technically, the operational list or mechanism used to reach candidate sample units.
3. **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.
4. **Censored observation:** a boundary rather than an exact value; technically, partial information restricting a value to a range, such as an unobserved completion time known to exceed a deadline.

4.6 Carrying uncertainty into a report

4.6.1 PLAIN — in simple words

1. An uncertainty statement describes the spread of values reasonably associated with a measurement under the available information. It is not an admission that the measurement was careless. [S35]
2. Different sources of uncertainty need to be considered together. Repeating a reading may reduce one component while leaving another unchanged.
3. The meaning of "plus or minus" must be stated. A standard uncertainty, an expanded uncertainty and a guaranteed physical limit are different statements.
4. An honest report contains both a result and the boundary of what the evidence supports. A narrow interval produced from unjustified assumptions is not stronger evidence than a wider, well-supported one.

4.6.2 PLAIN — a picture in your head

1. Imagine locating a shop on a map using two clues. One clue has a small uncertainty about the street position; another has an uncertainty about where the whole map was aligned.
2. Repeating the first clue does not automatically fix the map's alignment. Some uncertainty is local, and some is shared.

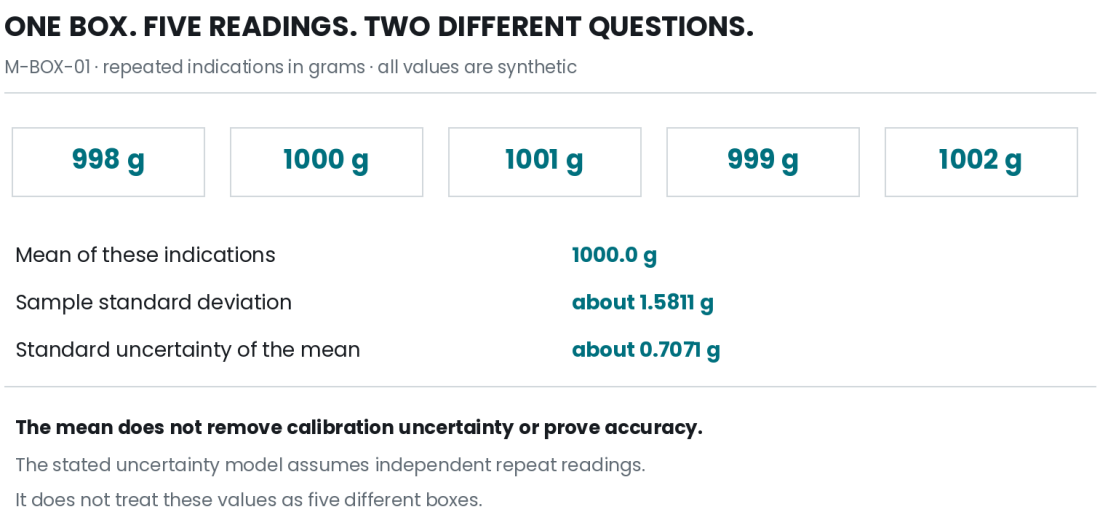

Figure 4.4: Figure 4.1. Keep the object, procedure, readings and uncertainty model attached to the result. The arithmetic is only one stage of the measurement record.

3. **Where this comparison breaks:** uncertainties are not simply distances that should always be added. Their combination depends on the mathematical model and on how the contributing effects are related.

4.6.3 PLAIN — a worked example

1. Return to M-BOX-01. The mean indication is **1,000 g**. Under the repeated-independent-observation model, the standard uncertainty of the mean from those readings is **0.70710678 g**.
2. For this calculation, assume a separate calibration contribution with **standard uncertainty 1.5 g**, uncorrelated with the first component. Assume any required correction has already been applied and that the two components cover the effects included in this toy budget.
3. Square, add, then take the square root:

```
u_repeatability = 0.70710678 g
u_calibration   = 1.5 g

u_combined = sqrt(0.70710678^2 + 1.5^2)
           = sqrt(0.5 + 2.25)
           = sqrt(2.75)
           = 1.65831240 g
```

4. If the reporting procedure uses coverage factor **k = 2**, the expanded uncertainty is **U = 2 × 1.65831240 = 3.31662479 g**.
5. One rounded report is **mass estimate 1,000.0 g; expanded uncertainty 3.3 g; k = 2**, with the observation procedure and the assumptions stated. This is an illustrative uncertainty calculation, not a real calibration certificate.
6. The association of k = 2 with approximately 95% coverage needs an appropriate approximately normal model and a reliable uncertainty estimate. The multiplier alone does not establish that coverage. [S36]

4.6.4 PLAIN — what is really happening inside

1. The report is applying a measurement model to input estimates and their uncertainty information. That information can come from statistical analysis or from other justified evidence.
2. A Type A evaluation uses a statistical analysis of observations. A Type B evaluation uses other available information, such as a calibration report or a specified distribution. These labels describe evaluation methods, not "good" versus "bad" uncertainty. [S32] [S34]
3. The model must account for shared effects. When subtracting a tare reading from a gross reading made on the same scale, some offset effects may be shared and may partly cancel. They should not automatically be treated as independent.
4. The software should keep enough of the uncertainty budget to explain the published result. Storing only `plus_minus = 3.3` discards the unit, coverage factor, components and assumptions that make the number meaningful.

4.6.5 TECHNICAL — the engineer's version

1. For a result `y = f(x1, …, xn)`, first-order uncertainty propagation uses sensitivities of f to the input values, their variances and their covariances. A sensitivity is how much the result changes for a small change in an input. [S31]
2. For an uncorrelated sum `y = a1*x1 + a2*x2`, the combined variance is `a1²*u1² + a2²*u2²`. Taking the square root gives the combined standard uncertainty. This is not the same operation as adding worst-case bounds.
3. For `net = gross - tare`, the general two-input expression is **u_net² = u_gross² + u_tare² - 2*cov(gross, tare)**. Ignoring the covariance assumes something about the relationship between the readings.
4. In a separate illustrative budget, standard uncertainties of 3 g and 4 g produce 5 g for the difference only when the covariance term is zero. A shared uncertain offset does not satisfy that independence assumption merely because two rows were recorded.
5. Strong nonlinearity, awkward distributions, dependent inputs or small-sample effects can require more careful methods than a first-order approximation. This chapter introduces the model; it does not certify every use of its shortcut.
6. A machine-readable report could include `estimate`, `unit`, `standard_uncertainty`, `coverage_factor`, `expanded_uncertainty`, `method_version`, `input_observation_ids` and `assumptions`. Include only fields justified by the actual method, rather than filling unknown quantities with invented zeros.
7. The final quality test is interpretability: can another reader reproduce the arithmetic and distinguish what was observed from what was assumed? A long decimal without that distinction is merely a precise-looking string.

4.6.6 WORDS — remember these

1. **Standard uncertainty:** uncertainty expressed like a standard deviation; technically, measurement uncertainty expressed as a standard deviation under the stated model.
2. **Combined standard uncertainty:** the uncertainty after input effects are combined; technically, the standard uncertainty of the result obtained from its input uncertainty model, including relevant dependence.
3. **Expanded uncertainty:** a stated multiple of standard uncertainty; technically, U = k × u_combined, accompanied by the coverage factor and justified interpretation.

4. **Covariance:** how two quantities vary together; technically, a measure of joint variation that contributes to propagation when input estimates are dependent.

5. **Uncertainty budget:** the explanation behind the uncertainty figure; technically, a statement of contributing uncertainty components, their evaluation and their combination in the measurement model.

4.97 Practice and worked answers

First predict the result

1. An export contains `mass = 1.25` without a unit. Can you safely compare it with a record containing `mass_g = 1250`?
2. A rectangular label is 25 cm by 40 cm. Calculate its area in square metres and show why a single division by 100 is insufficient.
3. Use M-BOX-01 to calculate the mean, sample standard deviation and standard uncertainty of the mean under the stated independent-observation model.
4. Two exact decimal inputs are 1.245 and 1.245. Compare half-up rounding to two decimal places before and after addition.
5. In the request-timing fixture, what does the 2-second average establish, and what does it leave unknown?
6. Combine independent standard uncertainties of 0.70710678 g and 1.5 g, then apply k = 2. Name one assumption that can invalidate a simple interpretation of the resulting interval.

Worked answers

1. No. The first record's unit and quantity definition are missing. It might be kilograms, grams or something else. A plausible match is not permission to rewrite the source. Obtain the contract or quarantine the ambiguous record.
2. The area is 25 × 40 = 1,000 cm^2. Each square centimetre is 0.0001 m^2, so the area is 0.1 m^2. Both length dimensions change scale; therefore the factor is squared.
3. The mean is 1,000 g. Squared deviations sum to 10 g^2, so the sample variance is 10/4 = 2.5 g^2 and s ≈ 1.58113883 g. Dividing by sqrt(5) gives about 0.70710678 g for the average's repeatability component, not its complete uncertainty.
4. Before addition: 1.25 + 1.25 = 2.50. After addition: 2.490 rounds to 2.49. The procedure must specify where rounding occurs.
5. It describes the twelve observed successful completions. It does not establish the mean completion time of all twenty attempts. Under the stated eventual-completion assumptions the latter has a lower bound of 5.2 seconds; without eventual completion a finite all-request mean is not established.
6. The combined value is sqrt(2.75) ≈ 1.65831240 g, and the expanded value is approximately 3.31662479 g. Unmodelled shared error, double-counted components or an unjustified distributional assumption can invalidate the intended interpretation.

4.98 Common wrong ideas

1. **Wrong:** every extra decimal place is extra accuracy. **Right:** displayed resolution and closeness to the measured quantity are different properties.
2. **Wrong:** a stable instrument must be correct. **Right:** it can repeat a shared offset consistently.

3. **Wrong:** an average of five readings is the same thing as a typical value for five products. **Right:** repeated measurements of one object and measurements across different objects answer different questions.
4. **Wrong:** units are decorative labels added at the end. **Right:** they determine interpretation and can expose invalid calculations before a report is produced.
5. **Wrong:** rounding each input and rounding the final result are interchangeable. **Right:** the order can change the answer.
6. **Wrong:** missing means zero. **Right:** replacing absence with zero introduces a substantive assumption.
7. **Wrong:** a low mean among successes proves the service is fast for everyone. **Right:** excluded timeouts can change the conclusion.
8. **Wrong:** all uncertainty components should be added as independent squares. **Right:** the model must account for dependence and avoid counting the same effect twice.
9. **Wrong:** $k = 2$ always guarantees 95% coverage. **Right:** that interpretation depends on the model and reliability of the uncertainty estimate.
10. **Wrong:** a report is complete once its arithmetic is correct. **Right:** it also needs the quantity definition, population, method, time boundary and limits of inference.

4.99 Chapter summary in 20 lines

1. Name the quantity intended to be measured before choosing a database field.
2. The measurand includes the object and conditions needed to distinguish the intended question.
3. Gross mass, tare mass and net mass have different definitions.
4. A recorded indication is not automatically a verified physical truth.
5. Units travel with values and determine meaningful calculations.
6. Converting a squared unit requires squaring its conversion factor.
7. Compatible dimensions do not guarantee compatible business meanings.
8. Precision concerns agreement among repeated observations under stated conditions.
9. Accuracy concerns closeness to the true value and is not itself a numerical percentage in VIM usage.
10. A detailed display does not prove a small measurement error.
11. The M-BOX-01 readings have mean 1,000 g and sample standard deviation about 1.581 g.
12. Under the independent-observation model, their mean's repeatability uncertainty is about 0.707 g.
13. Averaging does not automatically remove shared offsets.
14. Rounding needs a scale, a tie rule and a defined place in the calculation.
15. Preserve missing observations rather than silently converting them to zero.
16. Report denominators and exclusions before generalising an average.
17. Uncertainty components need a stated model and a check for dependence.
18. The example's independent components combine to about 1.658 g.
19. Multiplying by $k = 2$ gives about 3.317 g, with coverage interpretation conditional on assumptions.
20. A useful measurement report lets the next reader separate observations, calculations and assumptions.

5

CHAPTER

Identity, Keys and the Problem of Duplicates

5.0 What this chapter gives you

1. You will distinguish a thing, its record, its display name and the identifier used to refer to it.
2. You will choose a key by examining its scope and lifetime, not merely whether its sample values look different.
3. You will explain why generating an identifier and enforcing uniqueness are separate jobs.
4. You will separate a repeated delivery of one event from a second genuine event with similar contents.
5. You will design a reviewable matching process that can preserve an uncertain outcome instead of pretending every pair is a match or a non-match.
6. You will preserve enough history to explain and reverse a mistaken identity merge without quietly rewriting unrelated records.

Keep the meanings separate

This chapter concerns **record identity**: which product, order, source record or event a reference names. It is not a complete treatment of proving who a person is or deciding what they may access. Those are distinct problems. The existence of a database key is not authentication, and knowing that key is not authorisation.

The established shop fixtures remain unchanged. O-1042 still has two order lines; its first line is identified by `(O-1042, 1)`. Multi-branch and matching examples below are new synthetic exercises. They do not assert that existing customers have been merged or that the shop has already deployed a multi-tenant service.

5.1 Names and identifiers

5.1.1 PLAIN — in simple words

1. A name helps a person recognise something. An identifier helps a system refer to the intended thing under an agreed rule. Sometimes one value does both jobs, but the jobs remain different.
2. Two products can both be called “Blue Notebook.” A product can also change its display name without becoming a different product in the shop’s catalogue.
3. The system therefore needs a rule for what stays the same through a change. A product identifier might remain stable while its label, description or supplier changes.
4. That stability is a design decision. Software cannot discover it merely by noticing that two strings look similar.

5.1.2 PLAIN — a picture in your head

1. Imagine a room of people wearing name badges. Two badges say "Dev." Calling out the name does not identify one person unambiguously.
2. Now give each visitor a numbered ticket for this event. The ticket can identify a particular admission record even when several visitors share a name.
3. **Where this comparison breaks:** the ticket identifies an admission in its issuing system, not necessarily one person for life. A person can have two tickets, and a stolen ticket does not prove the holder's identity.

5.1.3 PLAIN — a worked example

1. In our original catalogue, **P-NOTE** identifies the notebook product used by O-1042. Its agreed unit price in that order is 7,550 paise.
2. Suppose a new display label is proposed: "Ruled Notebook." Under this teaching policy the product identity remains P-NOTE. The historical order still points to the same product and retains its own agreed price.
3. Now introduce a distinct catalogue item, **P-NOTE-A5**, whose display label also contains "Notebook." Similar words do not justify replacing P-NOTE with P-NOTE-A5 in old orders.
4. Finally, the supplier's catalogue might call our P-NOTE **NB-17**. That creates a mapping between identifier systems; it does not make `NB-17` universally meaningful.

Reference	What it names in this example	What it does not establish
P-NOTE	One product in our catalogue	A physical individual notebook
O-1042	One order in our shop fixture	Payment settlement or delivery by itself
O-1042, line 1	One agreed order line	Every line containing the same product
NB-17 under supplier SUP-A	A supplier's product reference	The meaning of NB-17 at another supplier

5.1.4 PLAIN — what is really happening inside

1. A reference points into an identifier space. The receiving system looks up that reference and interprets the returned record under the correct contract.
2. When a human-readable label changes, a stable identifier can keep references connected. When an identifier changes, the system needs a deliberate migration or mapping policy.
3. A record ID should also have a lifecycle rule. Can it be reused after deletion? Can it be imported from another system? Can it survive a merge? Those choices affect old links, delayed messages and historical reports.
4. For our teaching catalogue, retired product identifiers will not be reassigned to different products. That is an explicit policy for the story, not a universal property of the word "identifier."

5.1.5 TECHNICAL — the engineer's version

1. Distinguish an **entity** from its representations. A product can have a current catalogue record, an export row and historical order-line references. These records do not all have the same grain or identifier.

2. A **key** identifies a record under the model's uniqueness rules. A primary key is the selected identifying key of a table; other candidate keys can be enforced independently. PostgreSQL primary keys require uniqueness and non-null values. [S02]
3. A displayed name is not a candidate key unless the domain actually guarantees uniqueness and the schema enforces the intended rule. A current dataset containing no duplicate names is only an observation about that dataset.
4. Consider mutability. An email address may change, be shared or later be reassigned. Whether it is suitable for a particular key depends on the domain contract and lifecycle; a convenient login field is not automatically a stable entity identity.
5. Store external identifiers with their authority or namespace. A useful mapping grain is one external identifier under one issuing namespace, linked to an internal record under a documented validity policy.
6. The honest version: adding a generated key does not solve every identity problem. It guarantees neither that only one record represents each real-world entity nor that an existing reference was attached to the correct entity.

5.1.6 WORDS — remember these

1. **Entity:** the thing the model is about; technically, a distinguishable object, concept or occurrence represented within a domain model.
2. **Identifier:** a value used to refer to something; technically, a designation interpreted within an issuing namespace and its lifecycle rules.
3. **Display name:** the label shown to a person; technically, a presentation attribute that need not be unique or stable.
4. **Primary key:** the table's chosen identifying key; technically, a selected set of columns whose combined values uniquely identify each row and cannot be null under the relevant database rules.

5.2 Natural and generated keys

5.2.1 PLAIN — in simple words

1. Some identifiers already exist in the work being modelled. Others are created by the new system solely to name its records.
2. An existing catalogue code can be useful when its issuing organisation maintains it consistently. A generated number can be useful when ordinary attributes are mutable or ambiguous.
3. Neither option is automatically correct. Ask who issues the value, whether it can change, where it must be unique and what happens when records arrive from elsewhere.
4. A machine that generates new numbers is not the same thing as a rule that refuses duplicate stored numbers. Good designs examine both.

5.2.2 PLAIN — a picture in your head

1. Imagine organising books using the number already printed by the publisher, or attaching your library's own accession label to each copy.
2. The publisher's number and the library label can both be useful because they name things at different levels. One edition can have many physical copies.

3. **Where this comparison breaks:** database key terminology depends on the model. A value that looks "natural" in one system may have been generated by another. The important question is what it identifies and what rules govern it.

5.2.3 PLAIN — a worked example

1. In a new supplier-import exercise, SUP-A supplies product code `017`, while SUP-B also supplies product code `017`.
2. Treating `017` alone as a globally unique product key incorrectly collides two issuing systems.
3. The pair `(SUP-A, 017)` differs from `(SUP-B, 017)`. An internal product record can have a separate generated ID while preserving either mapping as evidence.
4. A second import of `(SUP-A, 017)` might be an update to the same supplier reference, not a new product. Its meaning depends on that supplier's versioning and reuse policy.
5. A generated internal ID therefore solves the naming of the receiving row, while the supplier mapping solves a different problem: recognising which source reference the row represents.

```
-- PostgreSQL 17 illustration; not an executed PostgreSQL lab.
CREATE TABLE supplier_reference_example (
    reference_id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    supplier_id text NOT NULL,
    supplier_code text NOT NULL,
    UNIQUE (supplier_id, supplier_code)
);
```

6. Read the two rules separately: the primary key identifies the receiving row, while the unique pair prevents two current rows from claiming the same supplier reference in this deliberately narrow model.

5.2.4 PLAIN — what is really happening inside

1. A sequence-like generator allocates values. A uniqueness constraint checks whether the relevant values conflict with accepted rows. The generator and the constraint can be configured or bypassed differently.
2. PostgreSQL documents that an identity column is not, by itself, a uniqueness guarantee. The underlying sequence can be reset, and explicit values can be inserted under permitted rules. A primary-key or unique constraint is still needed for uniqueness. [S37]
3. Another approach uses a large identifier format designed for distributed generation, such as a UUID. This reduces the need for every issuer to ask one shared counter, but it still needs correct generation and handling. [S38]
4. Choosing a large identifier does not remove the need to reject reused IDs, preserve namespaces or distinguish duplicate source records from duplicate real-world entities.

5.2.5 TECHNICAL — the engineer's version

1. A **natural key** is derived from domain attributes or identifiers already meaningful in the modelled work. A **surrogate key** is introduced to identify a record without carrying the same business meaning. These are modelling categories, not quality rankings.
2. A natural key may be long, composite or mutable. A surrogate key can provide a compact reference, but it can also permit duplicate business records unless relevant alternate uniqueness rules remain enforced.

3. In UUID version 4, 128 total bits include fixed version and variant fields, leaving **122 random bits** in the specified layout. The collision model therefore has **2^122** possible random values, not 2^128. [S38]
4. Under independent uniform generation, the approximate probability of at least one collision among n IDs is `n*(n-1)/(2*N)` when this probability is small and N is the number of possible values. This is the birthday approximation, derived by counting possible pairs.
5. With n = 1,000,000,000 and N = 2^122, the approximation is about **9.4 × 10^-20**. This is an arithmetic illustration under the stated random model, not an observed failure rate of a deployed system.
6. Broken randomness, copied state, accidental identifier reuse and incorrect imports are outside that ideal collision calculation. Extremely small random-collision probability does not prove correct implementation.
7. Different UUID versions encode different information. A time-ordered identifier is not automatically a globally authoritative event order or an authentication token. Interpret the actual version rather than assuming every UUID has the same meaning. [S38]
8. Query patterns and storage costs matter too, but only after identity requirements are defined. Later index and storage chapters will examine the costs of different key layouts without turning them into universal "always use this type" rules.

5.2.6 WORDS — remember these

1. **Natural key:** a key drawn from the work's own attributes; technically, an identifying set of domain attributes rather than a record-only identifier introduced by the receiving model.
2. **Surrogate key:** an extra identifier created for the record; technically, a key whose principal role is row identity rather than existing business meaning.
3. **Unique constraint:** a rule against repeated key values; technically, database enforcement that the relevant value combination cannot appear in conflicting rows under its null and comparison rules.
4. **UUID:** a standardised large identifier; technically, a 128-bit identifier whose version and variant determine the interpretation and generation rules.

5.3 Scope and composite identity

5.3.1 PLAIN — in simple words

1. "Number 17" is incomplete when two branches each issue their own number 17. The missing information is the scope in which the number was issued.
2. A composite identity uses several fields together. The combination identifies the intended record even when none of the fields is unique alone.
3. An order-line number is a familiar example. Line 1 appears in many orders. The order identifier plus the line number names the intended line.
4. Do not lose the scope while exporting, joining or constructing an address. Keeping it in the database but dropping it in the next message reintroduces the ambiguity.

5.3.2 PLAIN — a picture in your head

1. Two apartment buildings can both contain flat 17. A courier needs the building and the flat, not just the number painted on the door.

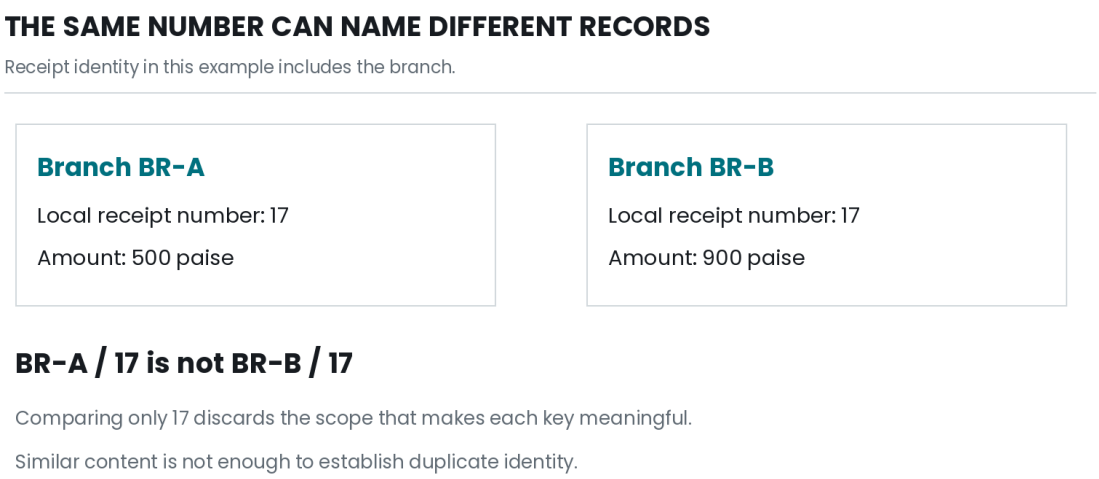

Figure 5.5: Figure 5.1. The same local number can be valid in two namespaces. Keep the full identifier together through lookups, messages and joins.

2. The full address works because its pieces belong together. Sorting the pieces independently and pairing them again can create an address that was never supplied.
3. **Where this comparison breaks:** a database scope need not be a physical place. It might be a tenant, issuing authority, dataset version or parent record.

5.3.3 PLAIN — a worked example

1. Our established order lines include **(O-1042, 1)** and **(O-1043, 1)**. They share a line number but not an identity.
2. Now use a separate branch-local receipt example: **(BR-A, 17)** and **(BR-B, 17)**. Both are legitimate under the policy that each branch numbers its own receipts.
3. A lookup using only `local_receipt_no = 17` may find two rows. Selecting the first one is not a valid resolution rule.
4. A lookup using both fields expresses the complete key: `branch_id = 'BR-A' AND local_receipt_no = 17`.
5. The same issue appears in joins. Joining receipt details to headers only by local receipt number can pair one branch's details with another branch's header.

Branch	Local receipt	Amount in this separate fixture
BR-A	17	500 paise
BR-B	17	900 paise

6. The two amounts are invented for this exercise and do not replace either canonical order total. The example exists to reveal the missing key component.

5.3.4 PLAIN — what is really happening inside

1. A database can enforce a rule over a column combination. The combination is checked as a whole; the individual columns need not each be globally unique. [S02]

2. The application must then carry the same combination through its workflows. A user-interface link that drops the branch component undermines the model even when the database schema is correct.
3. The record's parent relationship may also be enforced by a foreign key. That checks the relevant stored relationship; it does not decide whether the current caller is allowed to read the parent or child.
4. The design therefore has two independent questions: **which record is this?** and **may this caller perform this action on it?** A complete key answers only the first.

5.3.5 TECHNICAL — the engineer's version

1. A composite primary key such as `(branch_id, local_receipt_no)` identifies a row by the ordered set of its component meanings. Referencing tables should preserve the relevant scope in their foreign-key relationship.
2. If a local number is unique only within a branch, enforcing a foreign key on the number alone does not express the intended relationship. Where a database rejects that incomplete reference, the error is useful feedback about the model.
3. The order-line grain remains `(order_id, line_no)` in our original single-shop fixture. A future multi-tenant design might need a tenant component or a separately guaranteed global order ID. That is a future schema decision, not a silent reinterpretation of existing examples.
4. Avoid ambiguous string concatenation. The pairs `('AB', 'C')` and `('A', 'BC')` both become `ABC` when joined without boundaries. Use a structured representation or a documented escaping and length rule.
5. Nullability is also part of identity. A complete identifying key cannot rely on an absent parent component. Different databases have distinct null and uniqueness semantics, so state the actual constraints rather than relying on a label such as "unique." [S02]
6. A scope check is not a substitute for authorisation. In a tenant-scoped service, the server must connect the requested scope to the authenticated caller's allowed actions. Accepting a user-supplied tenant ID without that connection does not enforce isolation.
7. Preserve key components when generating reports and cache entries. A cache keyed only by a branch-local number can return a perfectly valid record belonging to the wrong branch.
8. The honest version: "the ID is unique" is an incomplete claim until you state **where**, **for what kind of record**, **over which lifetime**, and **under which enforcement rule**.

5.3.6 WORDS — remember these

1. **Namespace:** the context in which an identifier has meaning; technically, an issuing or interpretive scope within which names are assigned and resolved.

2. **Composite key:** several fields identifying a row together; technically, a key consisting of more than one attribute.

3. **Foreign key:** a checked reference to another stored key; technically, a constraint requiring referencing values to correspond to an eligible referenced key, subject to the database's rules.

4. **Tenant scope:** the customer boundary in a shared service; technically, the domain boundary used to associate records and permitted operations with a tenant, requiring access control beyond identification.

5.4 Duplicate delivery versus repeated events

5.4.1 PLAIN — in simple words

1. Receiving a message twice does not necessarily mean the underlying action happened twice. A sender may repeat delivery because the first acknowledgement was lost.
2. The reverse also matters: two real actions can have identical-looking contents. A customer can buy one pen now and another pen a minute later.
3. The system needs to distinguish the identity of an event from its contents and from the identity of one delivery attempt.
4. "Delete every repeated-looking row" is therefore not a safe general duplicate policy. It can remove genuine work or preserve a harmful replay, depending on what it compares.

5.4.2 PLAIN — a picture in your head

1. Mira sends a photocopy of one signed dispatch note because the recipient says the first copy did not arrive. Two envelopes now carry evidence about one dispatch.
2. On another day she sends two separate dispatch notes for two separate parcels containing the same product. Similar descriptions now concern two events.
3. **Where this comparison breaks:** software can preserve an explicit source-and-event identifier and compare a replay against recorded content. A paper description may not contain those machine-checkable fields.

5.4.3 PLAIN — a worked example

1. Introduce four synthetic messages from source `/miras-corner/till-a`. None is a new statement about the already established stock discrepancy.

Delivery	Event ID	Meaningful payload	Our proposed treatment
D-A	E-700	One pen issued	First accepted delivery of E-700
D-B	E-700	One pen issued	Replay of E-700; do not apply again
D-C	E-701	One pen issued	Distinct event; apply under its own rules
D-D	E-700	Three pens issued	Identifier conflict; investigate

2. D-A and D-B are two deliveries describing one event. D-C describes a second event even though its quantity matches.
3. D-D reuses E-700 with different meaningful content. Silently treating it as an ordinary replay would hide an inconsistency; applying it would reuse one identity for conflicting claims.
4. Our toy consumer will therefore distinguish **accepted**, **duplicate**, and **conflict**. The conflict is rejected rather than resolved by choosing whichever value arrived last.
5. The standard CloudEvents model similarly defines event uniqueness through the combination of `source` and `id`. Its specification permits a resent duplicate to retain that identity. [S39]

5.4.4 PLAIN — what is really happening inside

1. The receiver checks whether it has already accepted the source-and-ID pair within the relevant retention and processing boundary.
2. If the pair is new, it validates the event and applies the intended update. If the pair is already present with the same agreed content, it returns the previous logical outcome without repeating the update.
3. If the pair has conflicting content, the receiver needs an explicit failure path. A producer bug, import mistake or hostile input must not become an unnoticed change to the original event.
4. Recording "seen" and applying the update are themselves related changes. In a persistent consumer they need appropriate coordination. Otherwise a failure between them can either lose an update or apply it twice.
5. The companion lab keeps these operations in a single in-memory routine. That illustrates the contract; it does not test a durable multi-process consumer.

5.4.5 TECHNICAL — the engineer's version

1. An **idempotent** operation produces the same intended state effect when repeated under its declared identity and conditions. Idempotency is a property of an operation's semantics, not a magic attribute attached to a request header.
2. Distinguish four possible identifiers: the business object ID, the command/request ID, the accepted event ID and the delivery-attempt ID. A retry can preserve some while changing others. Their relationship must be documented.
3. CloudEvents 1.0.2 requires the `source` and `id` combination to identify distinct events. Its wire-level `specversion` value remains `1.0`; the patch release number is not written as `1.0.2` in that attribute. [S39]
4. A uniqueness rule on `(source, event_id)` prevents storing two rows under the same identity, but does not alone prove that the first update and its deduplication record were committed together.
5. Compare meaningful content under a stable schema. Raw JSON text can differ in insignificant spacing or member order. Conversely, a comparison that drops a meaningful field can falsely classify conflicting events as equivalent. JSON representation and application semantics are separate boundaries. [S07]
6. A hash can accelerate a comparison or bind a retained representation, but it must be interpreted with its algorithm and canonicalisation rules. The lab compares a fixed tuple of validated domain fields rather than presenting an arbitrary JSON hash as semantic truth.
7. A deduplication record also has a lifetime. If it expires before a legitimate retry arrives, the consumer may treat the retry as new unless another durable business rule prevents the repeated effect. The guarantee must state its retention and recovery assumptions.
8. "Exactly once" is therefore incomplete without an effect boundary: once in a table, once in a projection, once in an email service or once in the physical world are different claims.

5.4.6 WORDS — remember these

1. **Event identity:** the name of one accepted occurrence or change; technically, the identifier combination that distinguishes an event under the producer's contract.

2. **Delivery attempt:** one effort to transport a message; technically, a transmission attempt that may repeat the same logical event.

3. **Idempotency:** repeating the operation without repeating its intended effect; technically, stability of the declared effect under repeated application within the operation's identity and failure model.
4. **Deduplication:** recognising a repeated identity; technically, detecting and handling duplicate deliveries or records under an explicitly defined equivalence and retention policy.
5. **Identity conflict:** one ID attached to incompatible content; technically, a violation of the mapping from an identifier to the immutable or otherwise controlled claim it is supposed to name.

5.5 Matching imperfect records

5.5.1 PLAIN — in simple words

1. Sometimes two systems describe the same thing without sharing an identifier. Matching then becomes a reasoning problem rather than a simple key lookup.
2. Similar spelling is evidence, not certainty. Two supplier records can look alike and describe different products; one product can also appear under several spellings.
3. A useful matching process can say **same**, **different**, or **not enough evidence**. Forcing every uncertain pair into one of the first two categories hides the uncertainty instead of solving it.
4. The consequences matter. An incorrect suggestion is different from an automatic merge that changes orders, permissions or historical reports.

5.5.2 PLAIN — a picture in your head

1. Imagine sorting two boxes of old catalogue cards. Some cards share an exact issuing code; others share only an abbreviated name and a partial description.
2. You can place uncertain pairs beside each other for review without gluing them into one card. That preserves the option to inspect more evidence.
3. **Where this comparison breaks:** a software merge may redirect thousands of relationships instantly. A convenient-looking shortcut can have effects far beyond the two rows visible on screen.

5.5.3 PLAIN — a worked example

1. In an independent synthetic matching exercise, consider records **SRC-A-17**, labelled "Blue ruled notebook," and **SRC-B-88**, labelled "Blue notebook." Their names suggest a candidate pair, not a completed identity decision.
2. A checked common manufacturer code would be different evidence from name similarity, but even that requires understanding the code's issuing scope and reuse rules.
3. Suppose a labelled evaluation set contains **10 genuinely matching pairs** and **90 genuinely different pairs**. A candidate rule accepts 8 of the matches and incorrectly accepts 3 different pairs.
4. It has therefore accepted **11 pairs**, of which 8 are true matches. Its precision on this set is **8/11 ≈ 72.7%**.
5. It found **8 of the 10 true matches**, so its recall is **80%**. Its false-positive rate among the different pairs is **3/90 ≈ 3.33%**.
6. These are different denominators. None of the figures means there is a 72.7% probability that any particular new pair is the same product.

Label in the fixture	Accepted as a match	Not accepted as a match
Same product	8	2

Label in the fixture	Accepted as a match	Not accepted as a match
Different products	3	87

7. The matrix is invented to teach the arithmetic. It is not a performance claim for a real KedByte matching service.

5.5.4 PLAIN — what is really happening inside

1. A matching workflow first prepares comparable features, then proposes candidates, evaluates evidence and applies a decision policy.
2. Normalising spaces or case can make comparison useful, but it also changes a representation. Preserve the original source value when needed to explain the decision, subject to the purpose and retention policy.
3. Unicode normalisation can make certain equivalent text sequences comparable. It does not prove that two named entities are the same, or that visually similar characters are safe to treat as identical. [S15]
4. A human-review queue should show why a pair was proposed and what remains uncertain. "The system says 0.92" is not enough when the score's meaning and validation are unexplained.
5. For our teaching design, a candidate match has no authority to rewrite source records. Only an explicitly recorded decision can create a mapping, and that mapping remains reviewable.

5.5.5 TECHNICAL — the engineer's version

1. **Entity resolution** attempts to identify records that refer to the same underlying entity. It is different from deduplicating identical source event IDs: one compares uncertain referents, while the other can enforce a known producer identity contract.
2. Candidate generation, sometimes called blocking, reduces the number of pairs examined. It can improve computational efficiency while excluding some true matches. Evaluate that stage rather than measuring only the final scorer.
3. Precision is `true_positives / (true_positives + false_positives)`. Recall is `true_positives / (true_positives + false_negatives)`. Define how zero denominators are reported instead of quietly inventing a perfect score.
4. Performance measured on a labelled sample depends on the sample's construction, label quality and relationship to the deployment population. A hand-selected easy test set does not establish production reliability.
5. A threshold selects a trade-off between missed matches and false matches. The appropriate policy depends on consequences, review capacity and available evidence. A score is not automatically a calibrated probability.
6. Separate matching from permissions. Linking two product records should not accidentally merge ownership rules. Linking person records is even more consequential and requires a separately designed access and decision process; this toy exercise does not implement one.
7. Preserve provenance for each accepted mapping: source record IDs, rule version, evidence references, decision outcome and the applicable time or version boundary. W3C PROV's distinctions between entities, activities and agents help describe this reasoning without confusing a source fact with a derived decision. [S05]

8. The honest version: an algorithm that compares strings can be perfectly implemented and still make wrong identity decisions. Correct computation and justified inference are separate requirements.

5.5.6 WORDS — remember these

1. **Entity resolution:** deciding which records concern the same thing; technically, inference over records that may lack a shared reliable identifier.
2. **Candidate match:** a pair worth investigating; technically, a proposed association that has not yet satisfied the final decision policy.
3. **Precision, matching:** the share of accepted matches that are correct; technically, true positives divided by all positive predictions, not measurement precision from Chapter 4.
4. **Recall:** the share of true matches found; technically, true positives divided by the actual positive cases under the evaluation labels.
5. **False positive:** a match accepted when the entities differ; technically, a positive decision for a negatively labelled case in the stated evaluation.

5.6 Merge, split and identity history

5.6.1 PLAIN — in simple words

1. A merge says that records should now be treated as referring to one entity under a decision policy. It does not make every original value true or erase why the records differed.
2. A split reverses or revises that association. To do it well, the system needs to know which source records and relationships belonged to the disputed decision.
3. "Delete the duplicate row" is often too crude. It can discard the evidence needed to repair a wrong merge or explain an old report.
4. At the same time, keeping every detail forever is not the solution. Retain the information justified by the purpose and define what becomes impossible when particular evidence is removed.

5.6.2 PLAIN — a picture in your head

1. Imagine attaching a removable note to two catalogue folders: "For current purchasing, treat these as one product group; decision M-01." The folders still show where their original information came from.
2. If the decision proves wrong, you can remove the association and identify the work performed while it was active.
3. **Where this comparison breaks:** undoing an association does not automatically undo purchases, exports or messages already produced from it. Those downstream effects need their own review and correction.

5.6.3 PLAIN — a worked example

1. Use a separate synthetic identity-history fixture with three source records: **SRC-A-17**, **SRC-B-88** and **SRC-B-89**. No real customer information is present.
2. Decision **M-01** associates SRC-A-17 and SRC-B-88 with canonical product group **CAN-10**. SRC-B-89 remains in **CAN-11**.

3. A later review finds that SRC-B-88 describes a different pack size. Decision **M-02** supersedes the earlier association for that source record and places it in **CAN-12**.
4. The original source IDs remain intact. A report can now distinguish "grouping under M-01" from "grouping under M-02" rather than quietly pretending the earlier interpretation never existed.
5. The corrected grouping does not automatically recalculate every previously exported report. The system must identify which outputs depended on M-01 and decide what correction or notification is appropriate.

Decision version	SRC-A-17	SRC-B-88	SRC-B-89
M-01	CAN-10	CAN-10	CAN-11
M-02	CAN-10	CAN-12	CAN-11

6. A small reversible mapping is therefore more informative than overwriting SRC-B-88's identity and losing its source history.

5.6.4 PLAIN — what is really happening inside

1. The source record and the current identity interpretation are separate layers. A mapping connects them under a decision version.
2. Updating the mapping changes what a current query resolves, but it does not change the original observation's meaning. A historical query may intentionally use the older mapping version.
3. A safe change checks dependent records, invalidates or rebuilds affected derived views, and records the decision's basis. Where the evidence is insufficient, the association can remain unresolved.
4. The process must also prevent impossible mappings, such as a chain that loops back to itself or one source assigned to two current canonical records when the domain permits only one.
5. These are proposed invariants for our exercise. The presence of a `canonical_id` column does not enforce them without the necessary constraints and change procedure.

5.6.5 TECHNICAL — the engineer's version

1. One modelling option is a versioned **crosswalk** from `(source_system, source_record_id)` to an internal canonical ID, accompanied by a decision identifier and status. It preserves both source identity and the receiving system's interpretation.
2. Specify whether one source may map to several canonical entities. A bundled product might require a decomposition relationship rather than an ordinary identity mapping. Do not force a many-part relationship into a false "same thing" assertion.
3. Distinguish **effective time** from **recording time** when needed. A decision can be recorded today while correcting a grouping used for an earlier period. Adding two timestamp columns alone does not implement a complete bitemporal system.
4. Review reference redirection. A merged ID might become an alias rather than vanish; a split can make an old alias ambiguous. Returning an explicit unresolved or retired state can be safer than guessing a destination.
5. Rebuildable projections should depend on named mapping versions or retain sufficient lineage to determine which outputs need reconsideration. A current dashboard and an archived decision report need not use the same grouping policy.

6. Retention changes the evidence available for reversal. If original attributes or mapping history are removed, a later repair may no longer be reproducible. Document that limitation rather than promising perfect reversibility forever.
7. The companion lab demonstrates two immutable mapping snapshots and verifies that changing the current grouping does not mutate the earlier snapshot. It does not implement a complete production merge workflow, cascading database migration or person-identity system.
8. The lesson is architectural: identity decisions deserve explicit state, provenance and failure handling. They are not harmless string replacements.

5.6.6 WORDS — remember these

1. **Canonical ID:** the receiving system's chosen reference; technically, an identifier used for the entity representation treated as the current reference under a stated mapping policy.
2. **Crosswalk:** a mapping between identifier systems; technically, a relation connecting scoped source identifiers with identifiers in another model.
3. **Merge decision:** an explicit association of records; technically, a versioned decision to treat selected representations as referring to one entity under stated evidence and rules.
4. **Split:** revising an association into distinct entities; technically, a correction of mapping relationships whose downstream effects may require separate repair.
5. **Identity history:** the record of how associations changed; technically, retained versions or events describing identifier mappings and their applicable decision boundaries.

5.97 Practice and worked answers

First predict the result

1. Two catalogue rows both have display name "Notebook." Does that prove one is a duplicate? What evidence is missing?
2. SUP-A and SUP-B both issue code `017`. Propose a source-reference key and explain why a generated receiving-row ID solves a different problem.
3. A receipt join uses only local receipt number 17 and omits branch. Describe a possible wrong result using the two-branch fixture.
4. E-700 arrives twice with the same validated content; E-701 arrives once with identical business fields. How many logical events should our proposed consumer apply?
5. Recalculate precision, recall and false-positive rate for 8 true matches accepted, 2 true matches missed, 3 different pairs accepted and 87 different pairs rejected.
6. After M-02 separates SRC-B-88 from CAN-10, which statements can be made about the current grouping, the old grouping and reports previously exported under M-01?

Worked answers

1. No. A shared display name does not establish shared identity. Check product definitions, issuing identifiers, pack sizes, source provenance and the domain's equivalence rule. Keep an uncertain result when evidence is insufficient.

2. Use `(supplier_id, supplier_code)` under the stated supplier-code lifecycle. The generated receiving ID identifies the receiving row; the scoped pair recognises the source reference. Neither alone proves that two supplier references describe the same product.
3. BR-A's detail could be paired with BR-B's header, or a report could multiply the joined rows. Include the branch scope in the relationship and verify the resulting grain.
4. Apply E-700 once and E-701 once: two logical events. The second delivery of E-700 is a replay, while E-701 has a distinct producer identity. A changed payload under E-700 is a conflict, not a third ordinary event.
5. Precision is $8/(8+3) \approx 72.7\%$; recall is $8/(8+2) = 80\%$; false-positive rate is $3/(3+87) \approx 3.33\%$. Each uses a different denominator. These are fixture-level evaluation results, not individual-match probabilities.
6. The current mapping places SRC-B-88 in CAN-12. The retained M-01 snapshot still records the former association. Previously exported reports require a separate dependency and correction review; they are not automatically repaired by changing the current mapping.

5.98 Common wrong ideas

1. **Wrong:** equal names mean the same entity. **Right:** names can be shared, changed or incomplete.
2. **Wrong:** a generated key prevents duplicate business records. **Right:** it can give two duplicate business records two different IDs.
3. **Wrong:** a UUID cannot collide. **Right:** random-collision models assign a small probability under assumptions; implementation and reuse failures remain separate risks.
4. **Wrong:** an identity column is automatically unique. **Right:** generation and uniqueness enforcement are separate in PostgreSQL's documented model.
5. **Wrong:** an ID needs no context. **Right:** scope and lifecycle determine where its uniqueness claim applies.
6. **Wrong:** knowing the full key grants access. **Right:** identification and authorisation are separate decisions.
7. **Wrong:** identical contents always indicate a replay. **Right:** two real events can have identical business fields and distinct identities.
8. **Wrong:** a matching score is automatically a probability. **Right:** its interpretation requires calibration and validation appropriate to the intended population.
9. **Wrong:** merging records makes the source facts agree. **Right:** a merge is a decision about interpretation; contradictory source claims can remain.
10. **Wrong:** undoing a merge repairs every consequence. **Right:** downstream reports, exports and actions need their own correction process.

5.99 Chapter summary in 20 lines

1. An entity, its record, its display name and its identifier are different things.
2. Stable references depend on an explicit identity and lifecycle policy.
3. A primary key identifies a row within the relevant table's rules.
4. Natural keys come from domain attributes; surrogate keys are introduced for record identity.
5. Generated keys do not replace relevant business uniqueness constraints.
6. PostgreSQL identity columns need separate primary-key or unique enforcement for uniqueness.
7. UUID version 4 has 122 random bits within its 128-bit layout.

8. Collision arithmetic is conditional on correct independent uniform generation.
9. A namespace supplies the context required to interpret a local identifier.
10. A composite key keeps all necessary identifying components together.
11. Dropping scope in a join, cache key or message can select the wrong record.
12. Identification never substitutes for authorisation.
13. One event can have several delivery attempts.
14. Two distinct events can carry the same business contents.
15. Our consumer distinguishes accepted events, same-content duplicates and identity conflicts.
16. Deduplication guarantees depend on coordinated updates and retention boundaries.
17. Entity resolution evaluates uncertain references rather than merely comparing keys.
18. Matching precision and recall describe different denominators and are not measurement precision.
19. Versioned mappings preserve the distinction between source records and identity decisions.
20. A merge or split needs provenance, explicit invariants and review of downstream effects.

6

CHAPTER

From Events to State: Keeping History

6.0 What this chapter gives you

1. You will distinguish a record of something happening from a value describing the current position.
2. You will reconstruct a notebook stock figure without treating a physical count as a movement of goods.
3. You will read a state-transition rule and explain why some requests must be rejected.
4. You will separate recorded order, physical time, delivery order and causal order.
5. You will understand what a snapshot saves, what a projection computes and what neither proves.
6. You will correct an earlier record without pretending that an external action has been undone.
7. You will describe the limits of reconstruction when records, interpretation rules or supporting evidence are no longer available.

The stock stream in this chapter is a synthetic continuation of Mira's Corner. The order `O-1042` still contains two notebooks and one pen, for INR 171.00. No price or quantity from that order is changed. A later count and an explicitly authorised stock adjustment are new teaching events. The separate order-state and reversal exercises use different identifiers so their histories cannot be mistaken for the original order.

6.1 An event versus a current value

6.1.1 PLAIN — in simple words

1. A current value answers a question such as "how many notebooks does our stock record say we have now?" An event answers a different question: "what did we record as happening?"
2. "Ten notebooks" is a position. "Two notebooks were issued against this order" is a change that helps explain a position.
3. The current position alone does not tell us how we arrived there. Ten could result from twelve minus two, eight plus two, or a much longer history.
4. Keeping a history can make an answer explainable. It does not automatically make every entry in that history true. A mistaken count remains a mistaken count even when stored carefully. [S05]
5. We must also distinguish an observation from a movement. Looking at nine notebooks on a shelf does not itself remove one notebook from a record that expected ten.
6. A useful system can show both: "expected stock: ten" and "last observed count: nine". It can then show the unresolved difference without inventing a reason for it.

6.1.2 PLAIN — a picture in your head

1. Imagine a water tank with a level indicator and a notebook beside it. The indicator says how much water appears to be there. The notebook records deliveries, withdrawals and inspections.
2. A delivery entry explains why the expected level rose. A withdrawal explains why it fell. A person reading the indicator makes an observation, not a delivery or withdrawal.
3. If the indicator and the notebook disagree, writing "leak" in the book is not a diagnosis. A leak is one possible explanation, but so are a bad reading and a missed entry.
4. **Where this comparison breaks:** goods are often counted as separate units, while water is measured continuously. Our notebook example also assumes that "one notebook" means one item of the same product and stock location, not a pack, a reservation or an item in another branch.
5. The analogy helps separate movement from observation. It does not tell us which stock policy the shop should adopt after a discrepancy.

6.1.3 PLAIN — a worked example

1. The stream identifier is `STOCK-P-NOTE`. It describes the expected physical stock of product `P-NOTE` at the one shop location in this example. It is not an "available to promise" quantity with reservations subtracted.
2. Here are the first three entries. Sequence means accepted position in this one stream.

Sequence	Recorded event	Effect on expected stock	Position afterwards
1	Opening balance: 12 notebooks	Establish 12	12
2	Goods issued: 2, for O-1042 line 1	Subtract 2	10
3	Physical count observed: 9	No movement	10

3. After sequence 3, the latest observed count is nine, the expected position is ten, and the discrepancy is `9 - 10 = -1` notebook.
4. We know that the two figures differ. We do not know from these three entries whether an item was lost, miscounted, issued without an entry, or assigned to the wrong location.
5. In this teaching continuation, Mira reviews the discrepancy and explicitly authorises an adjustment of minus one. That decision becomes sequence 4 and refers to observation 3.
6. The expected position becomes nine because of the authorised adjustment, not because the software silently treated every observation as a movement.
7. A later receipt of five notebooks, with reference `R-500`, becomes sequence 5. The expected position becomes fourteen.

```
opening balance                          12
issue for O-1042 line 1                   -2
count observation of 9                     0  (no movement)
authorised adjustment linked to count 3   -1
receipt R-500                            +5
                                         --
expected position after sequence 5        14
```

A COUNT OBSERVATION DOES NOT MOVE GOODS

STOCK-P-NOTE · one location · expected physical stock

1	Opening balance: 12	12
2	Issue 2 for O-1042 line 1	10
3	Observe 9: no movement	10
4	Authorise adjustment of -1	9
5	Receive 5 under R-500	14

Sequence 4 is an explicit new decision linked to observation 3.
The discrepancy does not, by itself, identify a cause.

Figure 6.6: Figure 6.1. Expected stock changes only under the declared movement and adjustment rules; the observation is separate evidence.

■ 6.1.4 PLAIN — what is really happening inside

1. The system reads the event type before deciding how an entry affects the position. A number without its meaning is not enough.
2. An opening entry creates the initial position. An issue reduces it. A receipt increases it. A count observation stores separate evidence. An authorised adjustment applies an explicit change.
3. The calculation starts from a stated boundary and applies the appropriate rule to each accepted entry. The answer depends on both the entries and those rules.
4. The program also tracks the last sequence processed. "Fourteen notebooks through sequence 5" is a stronger description than an unexplained "fourteen". It tells us the boundary of included history.
5. A person may still dispute the history. Linking the adjustment to the observation allows a reviewer to follow the decision; it does not prove that the observation was accurate or that every required approval occurred outside this teaching example.
6. The honest version: a record can preserve what the system accepted and why it produced an answer. It cannot travel back in time and directly inspect the shelf.

■ 6.1.5 TECHNICAL — the engineer's version

1. A **state** is the information a model holds at a particular logical boundary. An **event** records an occurrence relevant to that model. A transition function can be written `next_state = apply(previous_state, event)`.
2. A projection folds a sequence of events into a derived representation. For an ordered sequence `e1 ... en`, the state is `apply(...apply(apply(initial, e1), e2)..., en)`. That notation is repeated application, not a requirement to hold the entire history in memory. [S40]
3. **Event sourcing** is a particular architectural choice in which an event history is the authoritative record used to reconstruct application state. A system having an audit table is not, for that reason

alone, event sourced. The relationship between the event store and any current-state tables must be specified. [S40]

4. Our stock projection has an explicit schema: expected quantity, latest observation and its sequence, processed sequence, and stream identifier. The teaching implementation rejects unknown event types instead of guessing that every numeric payload is a delta.
5. Opening balance and subsequent movements are different operations. Replaying an opening-balance event in the middle of an established stream would replace rather than explain the position; our lab rejects that malformed sequence.
6. An invariant in this bounded model is that expected physical stock must not become negative. That is a chosen rule for this fixture, not a universal database rule or a claim that every inventory application must reject negative book stock.
7. Retain provenance for important derived statements: input identifiers, transformation identity and the boundary of included input. W3C PROV provides a vocabulary for entities, activities and derivations, rather than an automatic guarantee of truth. [S05]
8. The lab tests the transition rules in memory. It does not implement a durable event store, multi-process concurrency, crash recovery or production access control. Those require additional mechanisms introduced later in the book.

6.1.6 WORDS — remember these

1. **State:** the position the model currently holds — a representation at a defined logical boundary.
2. **Event:** a record of something relevant happening — an occurrence interpreted under a stated event schema.
3. **Projection:** an answer built from other records — a derived representation produced by applying transformation rules to source data.
4. **Event sourcing:** keeping the accepted event history as the principal record — an architecture that reconstructs state from authoritative events.
5. **Observation:** something recorded as seen or measured — evidence that need not itself authorise or cause a state-changing business action.
6. **Stock adjustment:** an explicit change to the expected stock record — a domain operation distinct from a goods movement or a count observation.

6.2 State transitions

6.2.1 PLAIN — in simple words

1. Not every requested change makes sense from every starting point. A draft order can be confirmed. An order already fulfilled cannot simply become an untouched draft again.
2. A state-transition rule says which change is allowed, from which starting state, and what the resulting state will be.
3. A request is not the same as an accepted event. “Please cancel” describes an intention. “Cancellation accepted” says that the system checked a request and accepted a change.
4. A rejection is also useful information, but it must not be mistaken for the successful action the requester wanted.

5. Two people can act on the same old view. The system needs to notice that one person's accepted change may make the other person's request stale.
6. These rules are choices made for the work being modelled. A database cannot invent the shop's cancellation policy merely by knowing what a table is. [S02]

6.2.2 PLAIN — a picture in your head

1. Picture a parcel moving through labelled trays: "preparing", "ready to send", and "handed to carrier". Moving it between trays is allowed only through specified steps.
2. A clerk who still sees an old list saying "preparing" must not erase the carrier handover just by placing a cancellation slip in the first tray.
3. The clerk needs the current position and a rule for what a cancellation means at that position. It might be a rejection or a different process, such as a return request.
4. **Where this comparison breaks:** physical trays encourage us to imagine one visible location. In software, several screens can display copies with different ages. A page that looks current is not proof that no accepted change has happened since it loaded.
5. Nor does a tray enforce the business rule by itself. We still need an authority that checks and records the move.

6.2.3 PLAIN — a worked example

1. Use a separate fictional order, `P-3001`. These rules do not assert anything new about the fulfilment status of `O-1042`.
2. We choose four states: `DRAFT`, `CONFIRMED`, `FULFILLED` and `CANCELLED`.

Current state	Request	Result in this teaching policy
DRAFT	Confirm	CONFIRMED
DRAFT	Cancel	CANCELLED
CONFIRMED	Fulfil	FULFILLED
CONFIRMED	Cancel	CANCELLED
FULFILLED	Cancel	Reject; a different returns process would be needed
CANCELLED	Fulfil	Reject

3. Mira and Dev both load `P-3001` at version 2, in state `CONFIRMED`. Mira requests fulfilment and Dev requests cancellation.
4. Suppose the system accepts Mira's request first. It records fulfilment and advances the version from 2 to 3.
5. Dev's request says that it was based on version 2. The system rejects it as stale rather than silently overwriting version 3.
6. On refresh, the current state is `FULFILLED`. Under our chosen policy, cancellation is not an allowed transition from that state.
7. Retrying does not mean repeatedly forcing the same desired result. It means reading the new state and re-evaluating whether the requested action is still valid.

6.2.4 PLAIN — what is really happening inside

1. The request includes an identity for the object being changed and the version on which the requester based the decision.

2. The system checks the current version and the transition rule together with accepting the new state. Separating the check from the acceptance leaves a gap in which another change could win.
3. A rejected request must not increase the version as though a successful transition occurred. Nor should a failed attempt send a success notification.
4. We can preserve a record of the failed attempt where appropriate, but it belongs to a different kind of record from the accepted order event.
5. The version number is not a timestamp. Version 3 means a position after version 2 in this object's accepted history; it does not mean three seconds have passed.
6. The honest version: checking versions in a single Python function demonstrates the idea. It does not make a shared production system safe unless the storage operation enforces the check under real concurrent access.

6.2.5 TECHNICAL — the engineer's version

1. A finite state machine specifies a set of states and permitted transitions. A command is evaluated against a state; a successful command may produce one or more domain events. Keep command identity, delivery identity and event identity distinct. [S39] [S40]
2. Optimistic concurrency attaches an expected version to a write. The storage boundary must condition acceptance on the expected version still matching. In a relational system, related checks and updates belong in an appropriate transaction with suitable constraints or locking. A transaction boundary alone does not choose the isolation behaviour needed for every multi-row invariant. [S01] [S02]
3. The lab's `transition_order(state, version, expected_version, action)` is a pure educational check. It returns a new state and version or raises a validation error. It never contacts a payment provider or changes the actual shop.
4. A repeated request after a lost reply creates a second problem: the client may not know that its first request succeeded. Request-level idempotency can let it retrieve the earlier outcome, but only within the stated identity and retention policy. Expected-version checking by itself is not a complete idempotency design. [S40]
5. Domain validation is different from authorisation. "Cancellation is allowed from CONFIRMED" does not mean any caller may cancel any confirmed order. The production operation must also check the caller's authority at the appropriate boundary.
6. An accepted transition can have effects outside one database. An order-state transaction does not atomically recall a parcel already handed to a courier. Separate external effects, retries and compensations must be modelled explicitly. [S40]
7. Test forbidden edges as well as allowed ones. Include unknown states, unknown actions, mismatched versions and a rejected action that leaves the original state unchanged. These are assertions about this chosen transition graph, not exhaustive proof of a complete commerce system.

6.2.6 WORDS — remember these

1. **Command:** a request to do something — an intention evaluated against current state and applicable rules.

2. **State transition:** an allowed move between positions — a specified change from one model state to another.

3. **Expected version:** the history position a request relies on — a concurrency precondition used to detect stale writes.
4. **Stale request:** a request based on an older position — an operation whose assumed version no longer matches the accepted state.
5. **Invariant:** a rule that must remain true — a property the chosen model preserves across accepted operations.
6. **Authorisation:** permission to perform an action — an access decision separate from whether the action is a valid domain transition.

6.3 Ordering and causality

6.3.1 PLAIN — in simple words

1. "Which came first?" can ask several different questions. Which event happened first? Which was written first? Which message arrived first? Which action depended on another?
2. These orders can disagree. A slow connection can deliver an earlier event after a later one.
3. Two devices can also have clocks that disagree. A printed time is a useful observation, but it is not automatically a trustworthy instruction about causal order.
4. If one action sends a message and another action receives that same message, the send comes before that receive in the communication relationship. The receive cannot be the cause of the send it receives. [S41]
5. An accepted sequence number can define the replay order for one stock stream. It does not automatically tell us the order of every action in every shop.
6. Before sorting a history, name the question the sort is supposed to answer. Sorting by the easiest available column may produce a tidy but misleading story.

6.3.2 PLAIN — a picture in your head

1. Imagine two clerks posting letters. Each writes the time shown on their own desk clock. One clock is five minutes fast.
2. A reply depends on the original letter arriving, even if the timestamps make the reply look earlier. The communication relationship is evidence that the clock labels alone are not the whole story.
3. Now consider two unrelated letters sent to different people. Sorting them by envelope colour gives an order, but tells us nothing about whether one caused the other.
4. **Where this comparison breaks:** software can exchange enormous numbers of messages with retries, relays and duplicates. Establishing that a receive corresponds to a particular send requires identifiers and protocol evidence, not a human guess from similar wording.
5. The analogy separates an ordering convention from a causal claim. It does not provide a universal clock.

6.3.3 PLAIN — a worked example

1. Our stock projection has processed sequences 1 and 2 and holds expected stock ten. It next receives sequence 4, the authorised adjustment.
2. The strict replay rule in this lab requires sequence 3 next. It rejects sequence 4 as a gap rather than pretending that the preceding history is complete.

3. A real ingestion service might buffer 4 until 3 arrives or fetch the missing entry. Our small lab does neither: rejection makes the missing prerequisite visible.
4. Once 3 is available, applying 3, 4 and 5 in sequence gives expected positions ten, nine and fourteen.
5. Here is a separate logical-clock exercise. A logical counter is a number used to preserve communication order; it is not a time of day.

```
Process A starts with counter 0.
A performs an event: counter becomes 1.
A sends a message: counter becomes 2; message carries 2.

Process B already has counter 5.
B receives that message: counter becomes max(5, 2) + 1 = 6.
B performs its next event: counter becomes 7.
```

6. The send is labelled 2 and its receive 6, preserving their order. Those four steps between 2 and 6 are not four seconds, and they do not count four intervening events in the whole system.
7. An unrelated process could also have an event labelled 6. Equal logical values do not prove the events happened at the same physical instant.

6.3.4 PLAIN — what is really happening inside

1. The stock stream uses one explicit sequence for interpretation. An arrival with a missing predecessor is held outside the accepted projection boundary in our lab.
2. An event timestamp can still be stored for reporting. It must not silently replace the sequence rule when clocks disagree or old events arrive late.
3. A logical clock advances locally and incorporates values carried by messages. That gives it information about communication dependencies without requiring perfectly synchronised wall clocks. [S41]
4. The clock provides a useful guarantee in one direction: if an event causally precedes another under the model, the first receives a smaller logical value. A smaller value alone does not establish a causal link. [S41]
5. A system may break ties using a process identifier to produce a deterministic total order. That tie-break is a chosen order, not a newly discovered fact about the physical world.
6. The honest version: the identifiers and clocks only describe the events the system models. A phone call or a human action outside the recorded communication can matter without appearing in that model.

6.3.5 TECHNICAL — the engineer's version

1. Lamport's happened-before relation includes local process order, message send before its receive, and transitive consequences. It is a partial order: some distinct events are not ordered by it. Logical clocks can satisfy `a -> b` implies `C(a) < C(b)`; the reverse implication is not generally valid. [S41]
2. The ordinary scalar-clock update is local increment for an event and, at a receive, `C = max(C, received_C) + 1`. A deterministic process-ID tie-break can extend clock order to a total order without proving additional causal relationships. [S41]
3. Distinguish **event time**, **recording time**, **delivery time** and **stream sequence**. Chapter 3 introduced representations of instants; this chapter explains why a correctly formatted timestamp still does not define every required processing order. [S09] [S40]
4. Our projector requires an exact next sequence within an exact stream. It rejects duplicates, gaps and unexpected stream identifiers. Duplicate transport delivery is handled before this strict fold; silently applying it twice would corrupt state. [S39] [S40]

5. Ingesting several streams does not create a globally agreed sequence just because each stream is internally ordered. Cross-stream invariants need a separate coordination model, addressed in the later parts on transactions and distributed systems.
6. Integer deltas can commute in an unconstrained arithmetic sum, but admission rules may not. Starting at zero, receiving one and then issuing one passes a non-negative-stock rule; issuing before receiving fails it. Final arithmetic equality does not erase the intermediate rule violation.
7. The lab demonstrates that difference with a small sequence and checks that a gap raises an error. It does not simulate network consensus or promise a global real-time order.

6.3.6 WORDS — remember these

1. **Stream sequence:** a record's position in one accepted history — a scoped ordering value used for interpretation or concurrency checks.
2. **Causal order:** one event preceding another through a dependency — the order induced by the model's local actions and communications.
3. **Logical clock:** a counter that helps preserve event order — a timestamping mechanism that need not represent physical time.
4. **Partial order:** an order that does not compare every pair — a relation under which some distinct events remain unordered.
5. **Sequence gap:** an expected predecessor is missing — a discontinuity that prevents this strict projector from claiming a complete prefix.
6. **Commutative operation:** an operation whose result is unchanged by swapping order — a property that does not automatically extend to validation rules or external effects.

6.4 Snapshots and projections

6.4.1 PLAIN — in simple words

1. Re-reading a long history from the very beginning can cost time. A snapshot is a saved position at a known point in that history.
2. To continue, we load the snapshot and read only the later events. That works only when the snapshot belongs to the correct stream and interpretation rules.
3. A snapshot without its boundary is like an answer without the question. We need to know which entries it already includes so we do not skip or repeat them.
4. A projection is the answer produced for a particular purpose. The same accepted events can support a stock balance, a daily movement report and a list of unresolved count discrepancies.
5. Those answers need not update at the same instant. A report can lag behind the accepted history, so it should make its processed boundary visible where that matters. [S40]
6. A snapshot and a backup are not interchangeable words. A snapshot used to accelerate one calculation may be stored on the same machine and fail with the rest of that machine.

6.4.2 PLAIN — a picture in your head

1. Imagine reading a long account book and writing a subtotal at the bottom of each page. Tomorrow you can begin from a checked subtotal rather than adding every old line again.

2. “Subtotal after page 12” is useful. “Subtotal: 10” with no page number is dangerous because you do not know where to resume.
3. A second clerk might make a different summary from the same pages, such as the number of deliveries rather than the number of items. Both are projections of the same records for different questions.
4. **Where this comparison breaks:** a software interpretation can change between versions. A subtotal made by a buggy program can be reproduced exactly and still be wrong. Its existence is not a certificate of correctness.
5. Nor is a subtotal a replacement for the original lines when a reviewer needs the details the subtotal discarded.

6.4.3 PLAIN — a worked example

1. Save a snapshot of `STOCK-P-NOTE` immediately after sequence 2. It says expected quantity ten, no count observation yet, and last processed sequence two.

```
{
  ”stream”: ”STOCK-P-NOTE”,
  ”projection_version”: ”stock-v1”,
  ”last_sequence”: 2,
  ”expected_quantity”: 10,
  ”last_observation”: null
}
```

2. Resume with sequence 3. The expected position stays ten and the latest observation becomes nine.
3. Apply sequence 4. The authorised adjustment reduces expected stock to nine. Apply sequence 5 and it rises to fourteen.
4. Replaying all five entries from the beginning also gives fourteen, with the same last observation and sequence boundary. The companion lab compares the full resulting state, not just one coincidentally equal number.
5. Now deliberately label the snapshot `STOCK-P-PEN` while keeping notebook events. The lab rejects it. A plausible number in a snapshot for the wrong product is not a valid starting point.
6. Deliberately change the projection version to an unknown value. The lab rejects that too; it does not assume an unfamiliar interpretation is compatible.

6.4.4 PLAIN — what is really happening inside

1. The snapshot contains both the derived state and metadata that says how to use it. Together they define a starting point for replay.
2. The projector accepts only events beyond that boundary, in the required order. Reapplying sequence 2 would issue the same two notebooks twice if the duplicate were not rejected.
3. A snapshot made while new entries are arriving must correspond to one coherent point. Copying the balance after event 5 but the checkpoint after event 2 would produce a mixed record that no valid replay created.
4. A report’s checkpoint similarly needs to agree with the result it describes. A crash between updating a report and updating its checkpoint can cause a retry problem unless those writes are coordinated. [S40]
5. Rebuilding a projection should calculate data, not casually repeat physical actions. Reading an old “goods issued” event must not send a second instruction to issue the goods again.

6. The honest version: successful replay proves agreement between two calculations under the tested rules and inputs. It does not prove that the input history includes every real-world event.

6.4.5 TECHNICAL — the engineer's version

1. A snapshot is an optimisation associated with a stream boundary and a projection schema or version. It is not necessarily the authoritative history and can often be discarded and rebuilt when the required source history is retained. That rebuilding condition matters. [S40]
2. Deterministic replay requires that interpretation not depend on uncontrolled current values. Calling "today's price", the current exchange rate or a random generator inside the fold can change a past result. Store the relevant historical input or define a versioned, reproducible rule where appropriate. [S05] [S40]
3. Our `stock-v1` projection uses integers, explicit event types and a stated sequence. A snapshot serialises expected quantity, observation information and last sequence. The lab validates matching stream and version before continuing.
4. A materialised projection may be updated asynchronously. A checkpoint such as "processed through sequence 5" is a statement about incorporated input, not a wall-clock guarantee that every source anywhere has been observed. [S40]
5. Persisting a projection and its consumer position requires an appropriate atomicity or idempotency strategy. In a single database, both can sometimes be committed in one transaction; across external systems that assumption must not be made without evidence. [S01] [S25] [S40]
6. Side effects deserve their own boundary. Sending an email, dispatching a parcel or charging a payment cannot safely be treated as an ordinary pure replay step. Recording delivery intent and handling retries are later topics; this chapter's fold deliberately performs none of those actions. [S40]
7. Test full replay against snapshot replay, wrong-stream snapshots, unknown versions and a missing first suffix event. Those checks expose specific errors. They do not replace tests of persistent storage, concurrent snapshot creation or recovery after process failure.

6.4.6 WORDS — remember these

1. **Snapshot:** a saved position at a known boundary — a serialised state with enough metadata to resume a compatible interpretation.
2. **Checkpoint:** how far processing has reached — a recorded boundary of incorporated source input.
3. **Deterministic replay:** the same retained inputs and rules give the same state — reconstruction without uncontrolled dependencies on current external values.
4. **Materialised projection:** an answer stored for later reading — a persisted derived view rather than a query result recomputed on every request.
5. **Projection version:** which interpretation produced the answer — an identifier for the relevant transformation or state representation.
6. **Side effect:** an action beyond calculating the returned value — an external change that must not accidentally be repeated during reconstruction.

6.5 Corrections and reversal events

6.5.1 PLAIN — in simple words

1. Keeping history does not mean pretending mistakes never happen. It means deciding how a mistake will be corrected and how the correction will be explained.
2. Overwriting an old entry can remove evidence of what the system previously believed. Adding a correction can preserve both the earlier record and the later repair.
3. A reversal often applies an opposite effect to an earlier entry. That arithmetic can repair a recorded position, but it does not necessarily undo anything outside the record.
4. If a parcel left the shop, writing an opposite quantity does not teleport it back. The physical return and the record of that return are separate matters.
5. A correction needs a target. "Subtract one" without saying which mistake it repairs can make today's total look right while leaving tomorrow's reviewer unable to explain it.
6. Different kinds of records need different correction policies. We should not impose a complicated event system on every small spreadsheet, or erase important history merely because editing a cell is easy. [S40]

6.5.2 PLAIN — a picture in your head

1. Imagine a teacher marking an exercise. The first mark was written beside the wrong answer. A later note says which mark was mistaken and gives the corrected result.
2. Crossing out the first mark without a trace makes it harder to explain why the student saw a different score yesterday. Keeping both marks without indicating which one now applies is also confusing.
3. The correction therefore has two jobs: make the current answer right and preserve an understandable relationship to the earlier answer.
4. **Where this comparison breaks:** reversing an accounting entry or correcting an inventory issue can affect several related records, permissions and external processes. A note in the margin alone does not enforce any of those relationships.
5. The analogy is about explanation, not a universal instruction to retain every detail forever.

6.5.3 PLAIN — a worked example

1. Use a separate teaching stream, `R-EX-01`. Do not apply these entries to `O-1042` or to the five-event notebook stream above.
2. This stream opens at twelve notebooks. In the scenario, two notebooks were actually issued, but the operator mistakenly recorded an issue of three.

Sequence	Entry	Recorded effect	Expected position
1	Opening balance	Establish 12	12
2	Erroneous issue record	-3	9
3	Reversal referring to entry 2	+3	12
4	Correct replacement issue	-2	10

3. The combined correction is `+3 - 2 = +1`. It takes the mistaken recorded position from nine to ten.
4. The physical issue remains two. We are repairing its representation, not asserting that three notebooks were physically returned and then two issued again.

5. A report of accepted entries can show all four events. A report of the corrected issue should explain the reversal relationship so it does not count two unrelated real-world issues.
6. A second reversal of entry 2 would add another three and corrupt the result. Our teaching lab rejects a second reversal of the same target.
7. A reversal referring to an unknown or non-reversible entry is also rejected. The word "reversal" is not permission to add any convenient number.

6.5.4 PLAIN — what is really happening inside

1. The system locates the target event and checks that the chosen correction policy allows it to be reversed.
2. It derives the reversal's effect from that target instead of trusting a caller's arbitrary positive quantity. In this example, reversing minus three produces plus three.
3. It records the link between the reversal and the original. A later reader can inspect why the extra positive entry exists.
4. It tracks whether the target was already reversed. Otherwise, a retry could apply the repair twice.
5. A replacement entry expresses the corrected fact. Reversal and replacement may need to be accepted together when a temporarily reversed-only position would violate the intended operation.
6. The honest version: our arithmetic example explains a correction model. Production acceptance must also address concurrent requests, transaction boundaries, permissions and the availability of evidence supporting the correction. [S01] [S40]

6.5.5 TECHNICAL — the engineer's version

1. A compensating event changes subsequent interpretation without mutating the accepted earlier event. Compensation is domain-specific: not every effect has a simple mathematical inverse, and an inverse in a ledger does not imply reversal of an external action. [S40]
2. The lab's reversal fixture permits reversal only of a prior issue in the same fixture and only once. It computes the inverse of the recorded delta and links to the target sequence. It deliberately does not expose a general "reverse anything" API.
3. The pure implementation can check these rules sequentially. In persistent storage, "not already reversed" must be enforced under concurrent access, for example through an appropriate unique relationship and transaction design. A preliminary application query alone leaves a race. [S01] [S02]
4. Preserve the distinction between event validity at acceptance and later interpretation. An entry can be a faithful record of what the system accepted even after a later correction establishes that its business content was wrong.
5. Queries need a declared perspective. "What did we record by yesterday?" and "What is our corrected account of yesterday's activity using information available now?" can return different answers without either being a database error. Source and derivation boundaries must be explicit. [S05]
6. A schema change can also affect how old events are read. A versioned reader or an explicit transformation may be needed; it should not silently reinterpret an old quantity's unit or turn an optional field into a fabricated historical fact. [S05] [S40]
7. Do not assume a sum is a sufficient audit. A wrongly duplicated issue and an unrelated positive adjustment can cancel numerically. Reconciliation should inspect identities, links and expected relationships as well as totals.

6.5.6 WORDS — remember these

1. **Reversal:** an opposite recorded effect tied to an earlier entry — a constrained correction operation whose scope must be defined.
2. **Compensation:** a later action that addresses an earlier effect — a domain-specific response that need not restore the exact original world.
3. **Replacement entry:** the newly accepted corrected representation — a record distinguished from the earlier entry it supersedes or repairs.
4. **Correction target:** the particular entry being repaired — an explicit reference needed to validate and explain a correction.
5. **Reconciliation:** checking whether related records agree as expected — a comparison of identities, relationships and totals rather than a single balancing sum.

6.6 History with retention boundaries

6.6.1 PLAIN — in simple words

1. "We keep history" needs a boundary. Which records are kept, for how long, for which purpose, and under whose control?
2. An append-only rule means ordinary changes add records rather than editing earlier ones. It does not, by itself, mean that every detail must be retained forever.
3. A history can contain unnecessary personal details or sensitive text. Adding more copies can increase the work of protecting and eventually removing those details.
4. Reconstruction depends on what remains available. If old source events are removed, a retained snapshot may preserve a position while losing the ability to explain every earlier movement.
5. That can be an intentional choice. The honest report says what was retained and what can no longer be reconstructed, rather than promising both complete deletion and unlimited recovery from the same removed evidence.
6. This chapter describes engineering boundaries, not a legal retention schedule. Applicable obligations and permissions must be established separately for the actual system and jurisdiction. [S40]

6.6.2 PLAIN — a picture in your head

1. Imagine an archive that retains annual totals but disposes of some old working slips under its approved policy. A total can still be read, but the archivist cannot honestly promise to produce every disposed slip.
2. A catalogue saying a slip once existed is not the slip itself. A checksum saying a retained file has not changed is not proof that the file originally contained a correct observation.
3. The archive might keep different categories for different periods. That requires an inventory and a process, not a single switch labelled "history".
4. **Where this comparison breaks:** digital records can be copied into logs, exports, backups, caches and analytical systems. Removing a main entry does not automatically remove every copy or every derived identifier.
5. The analogy is useful only if we remember that the archive has many rooms, not one cupboard.

6.6.3 PLAIN — a worked example

1. Suppose the teaching stock system retains a checked snapshot through sequence 5: expected quantity fourteen, latest observation nine at sequence 3, with interpretation `stock-v1`.
2. Under a hypothetical retention decision, the detailed events 1–5 are no longer available to this reader. The snapshot is still present and later events remain readable.
3. The reader can continue a compatible calculation from that boundary, provided the snapshot and its metadata are trustworthy enough for the intended use.
4. The reader cannot replay the missing prefix from scratch or inspect the original wording of the authorised adjustment. That evidence is absent from the available package.
5. A remaining event identifier or hash may help compare a future recovered copy. It cannot recreate the deleted content by itself.
6. Write the limitation plainly: "Current state continues from snapshot through sequence 5; the underlying prefix is not available here. Full prefix reconstruction was not performed."
7. This is a hypothetical retention exercise. No source files, user records or published materials were deleted to demonstrate it.

6.6.4 PLAIN — what is really happening inside

1. The system separates data categories and their purposes. A minimal operational event need not repeat every detail of the customer profile that happened to be on screen when the action occurred.
2. It records what a projection actually needs. Where an event refers to another record, we must decide what happens if that referenced detail changes or is no longer retained.
3. A replay that depends on deleted external content may fail or produce a less detailed result. The system should not silently fill the gap with a current profile or a guessed value.
4. Backups and exported copies need their own handling. A restored old copy can reintroduce information that a later process removed unless restoration accounts for subsequent actions.
5. A retained audit statement can say that a deletion or expiry action was recorded. It should not overclaim that every possible copy on every system has been examined unless that was actually established.
6. The honest version: recordkeeping, reproducibility and minimisation can pull in different directions. Good design makes the trade-off explicit and tests the promises it actually makes. [S05] [S40]

6.6.5 TECHNICAL — the engineer's version

1. Event-store immutability is an application or storage property with a defined enforcement boundary. It is not an automatic retention policy and does not imply immunity to privileged changes, storage loss or deletion. [S40]
2. Avoid putting unnecessary sensitive payloads into long-lived events. One design keeps an event's minimal operational fact separate from mutable or separately governed attributes. That separation creates reference-resolution and replay questions that must be designed, not ignored. [S40]
3. A reconstruction claim should identify input availability, schema versions, transformation versions, snapshots and the scope of the resulting output. A provenance model can describe those relationships, while verification remains an additional activity. [S05]
4. If a projection was rebuilt from a retained snapshot rather than the full history, record that distinction. Equal current totals do not prove equality of all earlier event histories.

5. A hash-based integrity check detects disagreement with a trusted reference value under its stated assumptions. It does not establish that the source was true, restore missing content, or prove that no unexamined copies remain.
6. The lab tests a snapshot continuation and its limits as data structures. It does not implement retention enforcement, encryption-key destruction, backup expiry or a legal erasure workflow. Do not treat its successful tests as evidence that those unimplemented capabilities exist.
7. Operationally, inventory the stores, define category-specific decisions, record authorised changes, test restoration and disclose gaps. The later chapters on retention and operations will develop that workflow; it is not complete merely because an event log exists.

6.6.6 WORDS — remember these

1. **Append-only:** ordinary updates add rather than replace — a mutation policy whose enforcement and exceptions must be specified.
2. **Retention boundary:** which history remains available — a stated limit on kept records, detail, time or access.
3. **Reconstruction boundary:** how far an answer can be rebuilt — the scope supported by retained inputs, snapshots and interpretation rules.
4. **Data minimisation:** avoiding unnecessary stored detail — collecting and retaining only the information justified for the defined purpose.
5. **Integrity check:** a check that retained content agrees with a reference — evidence of consistency, not automatic evidence of real-world truth.
6. **Restoration:** bringing back a stored copy — a recovery operation that must account for the copy's age and subsequent required changes.

6.97 Exercises with worked answers

Exercise 1 | A count is not a movement

The stream opens at 12, records an issue of 2, and then records a count observation of 9. What are expected stock, latest observed count and discrepancy?

Worked answer. Expected stock is `12 - 2 = 10`. Latest observation is 9. Observed minus expected is `9 - 10 = -1`. No adjustment has yet been authorised, so changing the expected balance to 9 would add a decision not present in these three entries. The discrepancy does not identify its cause.

Exercise 2 | Continue from a snapshot

A compatible `STOCK-P-NOTE` snapshot through sequence 2 holds expected quantity 10. Sequence 3 observes 9, sequence 4 authorises an adjustment of -1 linked to 3, and sequence 5 receives 5. What is the result, and what must be checked before replay?

Worked answer. The observation leaves expected quantity at 10. Adjustment gives 9. Receipt gives 14. Check the stream identifier, projection version, snapshot state and last sequence. Require the suffix to begin at 3 and continue without gaps under this lab's strict rule. Record that the final position includes sequence 5.

■ Exercise 3 | Two requests, one old version

An order is CONFIRMED at version 2. Fulfilment is accepted first, producing FULFILLED at version 3. A cancellation request based on version 2 then arrives. Should the system blindly retry the state change?

Worked answer. No. Its precondition is stale. After reloading version 3, cancellation is forbidden by the teaching transition graph. A separate returns process could exist, but it is not implemented here and cannot be invented by relabelling the fulfilled order as cancelled.

■ Exercise 4 | Read the logical clock

A process has counter 5 and receives a message carrying counter 8. What is the new counter? Does an unrelated event with a smaller counter necessarily cause this event?

Worked answer. `max(5, 8) + 1 = 9`. No. Logical-clock order preserves causal precedence in one direction; a smaller scalar value alone does not establish a causal relationship. The counter is not a physical duration. [S41]

■ Exercise 5 | Repair a mistaken issue

In separate stream R-EX-01, the opening quantity is 12. An issue was incorrectly recorded as 3 when the scenario says 2 were actually issued. Show a reversal and replacement, and explain why a second reversal should fail.

Worked answer. Record -3, giving 9. Reverse that target with +3, returning the recorded position to 12. Record the correct replacement -2, giving 10. A second reversal would add another 3 despite referring to the same already repaired entry, so the fixture rejects it. These entries repair the record; they do not describe goods physically travelling back and forth.

■ Exercise 6 | State what missing history prevents

You retain a compatible snapshot through sequence 5 and all later events, but not events 1–5. Can you claim to have reconstructed the entire history from its original source events?

Worked answer. No. You can report continuation from the retained snapshot under stated assumptions. You cannot inspect or replay the missing prefix. The snapshot's metadata and any retained verification help characterise its basis; they do not replace the absent source details.

6.98 Common wrong ideas

1. **Wrong:** a current balance explains its own history. **Right:** several different histories can produce the same balance.
2. **Wrong:** observing a stock count is the same as authorising an adjustment. **Right:** evidence and state-changing decisions are distinct operations in this model.
3. **Wrong:** a cancellation request proves that cancellation occurred. **Right:** a request may fail validation, authorisation or its version precondition.
4. **Wrong:** sorting timestamps discovers every causal relationship. **Right:** clocks, delivery and dependencies require separate interpretation.
5. **Wrong:** a smaller logical counter proves causation. **Right:** causal precedence implies increasing counters, but the converse does not generally hold. [S41]
6. **Wrong:** a snapshot is useful without knowing where it belongs. **Right:** stream identity, interpretation version and sequence boundary are essential.

7. **Wrong:** replaying old events should repeat their physical effects. **Right:** rebuilding a projection should not resend parcels or repeat charges.
8. **Wrong:** a reversal automatically undoes the outside world. **Right:** an opposite recorded effect and an external compensation are different things.
9. **Wrong:** append-only means every payload must remain forever. **Right:** mutation policy and retention policy answer different questions.
10. **Wrong:** passing this in-memory lab proves a production event platform is reliable. **Right:** it checks bounded rules and fixtures, not unimplemented storage, security, concurrency or recovery.

6.99 Chapter summary in 20 lines

1. A state describes a position; an event records an occurrence relevant to a model.
2. A retained event is not automatically a true account of the physical world.
3. Our notebook stream starts at twelve and issues two for O-1042, giving ten.
4. Observing nine records a discrepancy; it does not itself authorise a movement.
5. An explicit minus-one adjustment and a receipt of five produce fourteen.
6. A projection applies declared interpretation rules to a defined input history.
7. Event sourcing makes an event history authoritative; an audit table alone does not establish that architecture.
8. A command requests a change and can be rejected rather than becoming an accepted event.
9. State-transition rules identify permitted moves from particular starting states.
10. An expected version helps detect requests based on stale state.
11. Physical time, recording time, delivery order and stream sequence are different concepts.
12. Logical clocks can preserve causal order without being clocks of physical time.
13. Smaller logical values do not, by themselves, prove causation.
14. A strict projector must not quietly claim completeness across a sequence gap.
15. A snapshot needs a stream, interpretation version and included-history boundary.
16. Rebuilding a projection must not casually repeat external side effects.
17. A reversal needs a valid target and a rule preventing duplicate application.
18. Correcting a record does not necessarily undo a real-world action.
19. Retention decisions can limit which earlier details remain reconstructible.
20. State the evidence boundary honestly, then move to Chapter 7: files, folders and data formats.

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

Accuracy

closeness to what is actually being measured; technically, closeness of agreement between a measured value and a true value of the measurand, not a numerical quantity in VIM usage.

Read in context: 4.3.6

Aggregation

combining several records into an answer; technically, a calculation over a defined group of inputs.

Read in context: 2.3.6

Append-only

ordinary updates add rather than replace — a mutation policy whose enforcement and exceptions must be specified.

Read in context: 6.6.6

Application

the program people use; technically, the software that implements a workflow and requests operations from other components.

Read in context: 1.1.6

Audit trail

information used to explain actions; technically, a record of relevant operations and context, with its own integrity and retention requirements.

Read in context: 2.6.6

Authorisation

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

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

Read in context: 6.2.6

C

Calibration

comparing indications with suitable references; technically, establishing the specified relationship between reference values and indications with their associated uncertainties.

Read in context: 4.3.6

Candidate match

a pair worth investigating; technically, a proposed association that has not yet satisfied the final decision policy.

Read in context: 5.5.6

Canonical ID

the receiving system's chosen reference; technically, an identifier used for the entity representation treated as the current reference under a stated mapping policy.

Read in context: 5.6.6

Cardinality

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

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

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

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

Read in context: 1.4.6

Causal order

one event preceding another through a dependency — the order induced by the model's local actions and communications.

Read in context: 6.3.6

Censored observation

a boundary rather than an exact value; technically, partial information restricting a value to a range, such as an unobserved completion time known to exceed a deadline.

completion time is only partly known — an observation limited by a timeout or stopping rule rather than a measured final duration.

Read in context: 4.5.6

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: 6.4.6

Code point

an assigned position in the Unicode code space; technically, a numeric value used in the character model, not a guarantee of one visible glyph.

Read in context: 3.4.6

Coercion

automatic conversion; technically, a transformation between types that may preserve or change intended meaning.

converting a value into another representation or type — an engine or application behaviour that must be deliberate when it changes interpretation.

Read in context: 2.2.6

Combined standard uncertainty

the uncertainty after input effects are combined; technically, the standard uncertainty of the result obtained from its input uncertainty model, including relevant dependence.

Read in context: 4.6.6

Command

a request to do something — an intention evaluated against current state and applicable rules.

Read in context: 6.2.6

Commit

accepting the transaction; technically, the transaction-completion operation under the engine's documented configuration.

accepting the transaction's database changes — successful completion under the engine's stated persistence and visibility semantics.

accept a transaction's database changes — the completion operation whose acknowledgement has engine- and configuration-specific guarantees.

Read in context: 1.5.6

Commutative operation

an operation whose result is unchanged by swapping order — a property that does not automatically extend to validation rules or external effects.

Read in context: 6.3.6

Compensation

a later action that addresses an earlier effect — a domain-specific response that need not restore the exact original world.

a new action addresses an earlier effect — an explicit corrective operation, not erasure of the original history.

a new action addressing an earlier completed action — an application-defined correction distinct from local transaction rollback.

Read in context: 6.5.6

Composite key

an identifier made from several parts; technically, a key consisting of more than one attribute.

several fields identifying a row together; technically, a key consisting of more than one attribute.

an identity made from several values together — a unique column combination whose scope is not supplied by any one component alone.

several fields identify one thing together — an identifying attribute set whose combined values are unique in its declared scope.

Read in context: 2.4.6, 5.3.6

Concurrency

work that overlaps in time; technically, the execution of operations whose interleaving can affect their observed results.

Read in context: 1.3.6

Contract

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

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

Read in context: 1.6.6

Conversion factor

the multiplier between compatible units; technically, the ratio used to transform numerical values while preserving the represented quantity.

Read in context: 4.2.6

Correction

a change to repair a stated defect; technically, an authorised transition whose scope, evidence and consequences should be recorded.

Read in context: 2.6.6

Correction target

the particular entry being repaired — an explicit reference needed to validate and explain a correction.

Read in context: 6.5.6

Covariance

how two quantities vary together; technically, a measure of joint variation that contributes to propagation when input estimates are dependent.

Read in context: 4.6.6

Crosswalk

a mapping between identifier systems; technically, a relation connecting scoped source identifiers with identifiers in another model.

Read in context: 5.6.6

D

Data minimisation

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

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

Read in context: 6.6.6

Database

the organised records; technically, a managed collection of data interpreted under a model and its rules.

Read in context: 1.1.6

Date

a calendar position; technically, a calendar day without necessarily specifying a time of day or an instant.

Read in context: 3.5.6

DBMS

the record-management machinery; technically, the engine providing database operations and its documented guarantees.

the database-management software — facilities for defining, accessing and administering databases.

Read in context: 1.1.6

Decimal arithmetic

arithmetic based on decimal representation; technically, operations over decimal values under a precision and rounding context.

Read in context: 3.3.6

Deduplication

recognising repeated representations of the same thing; technically, a policy-driven process whose identity assumptions and conflict handling must be explicit.

recognising a repeated identity; technically, detecting and handling duplicate deliveries or records under an explicitly defined equivalence and retention policy.

Read in context: 2.4.6, 5.4.6

Delivery attempt

one effort to transport a message; technically, a transmission attempt that may repeat the same logical event.

Read in context: 5.4.6

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: 3.6.6

Derived data

an answer made from other data; technically, a representation produced by a transformation or calculation over inputs.

Read in context: 1.4.6

Deterministic replay

the same retained inputs and rules give the same state — reconstruction without uncontrolled dependencies on current external values.

Read in context: 6.4.6

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: 4.2.6

Dimensional analysis

checking a calculation through its units; technically, checking compatibility of dimensions across an equation while recognising that semantics require further checks.

Read in context: 4.2.6

Display name

the label shown to a person; technically, a presentation attribute that need not be unique or stable.

Read in context: 5.1.6

Domain

the allowed kind of value; technically, the set of permitted values and associated meaning for an attribute.

the values and meaning permitted for a role — a business-level contract potentially narrower than a storage type.

Read in context: 2.1.6

Domain type

the business meaning of the value; technically, a model-specific type whose rules may exceed the underlying storage type.

Read in context: 3.1.6

Durability

surviving specified failures; technically, the persistence guarantee for accepted changes under stated assumptions.

the promised survival of committed work — persistence under defined failures, engine configuration and storage assumptions.

acknowledged work survives specified failures — persistence under a named configuration, recovery model and failure boundary.

Read in context: 1.5.6

Duration

elapsed amount of time; technically, a measured or specified interval whose units and arithmetic semantics must be stated.

Read in context: 3.5.6

E

Empty string

text containing no characters; technically, a zero-length string, distinct from SQL NULL in the engines discussed here.

Read in context: 3.6.6

Encoding

the mapping into bytes; technically, a defined representation of a character sequence for storage or interchange.

Read in context: 3.4.6

Entity

the thing the model is about; technically, a distinguishable object, concept or occurrence represented within a domain model.

Read in context: 5.1.6

Entity resolution

deciding which records concern the same thing; technically, inference over records that may lack a shared reliable identifier.

Read in context: 5.5.6

Event

a record of something relevant happening — an occurrence interpreted under a stated event schema.

Read in context: 6.1.6

Event identity

the name of one accepted occurrence or change; technically, the identifier combination that distinguishes an event under the producer's contract.

Read in context: 5.4.6

Event sourcing

keeping the accepted event history as the principal record — an architecture that reconstructs state from authoritative events.

Read in context: 6.1.6

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: 2.5.6

Evidence

what supports a claim; technically, information whose relevance and limitations must be assessed for that claim.

Read in context: 1.2.6

Expanded uncertainty

a stated multiple of standard uncertainty; technically, U = k × u_combined, accompanied by the coverage factor and justified interpretation.

Read in context: 4.6.6

Expected version

the history position a request relies on — a concurrency precondition used to detect stale writes.

Read in context: 6.2.6

F

False positive

a match accepted when the entities differ; technically, a positive decision for a negatively labelled case in the stated evaluation.

a possible match that is not actually present — a positive candidate result rejected by a later exact check.

Read in context: 5.5.6

Field

one named part of a record; technically, an attribute position with an agreed interpretation.

Read in context: 2.1.6

File format

how bytes are arranged; technically, the syntax and interpretation rules for a representation.

Read in context: 1.3.6

Floating point

a number representation with a movable scale; technically, a finite significand and exponent representation with defined rounding behaviour.

Read in context: 3.3.6

Foreign key

a checked reference to another stored key; technically, a constraint requiring referencing values to correspond to an eligible referenced key, subject to the database's rules.

a declared reference to another allowed key — a referential-integrity constraint linking child values to parent values.

a stored link required to match an eligible parent — a referential constraint on child values with defined missingness, actions and checking time.

Read in context: 5.3.6

Functional dependency

one part determines another; technically, a constraint under which equal determining attributes imply equal dependent attributes.

one set of fields fixes another under a rule — X → Y means equality on X implies equality on Y in every legal relation instance.

Read in context: 2.3.6

G

Grain

what one record stands for; technically, the observation or business unit represented by each row.

what one row represents; technically, the unit of observation or business fact represented at the dataset's chosen level.

Read in context: 1.4.6, 2.3.6

Grapheme cluster

a unit closer to what a reader perceives as a character; technically, a text-segmentation unit defined by applicable Unicode rules and possible tailoring.

Read in context: 3.4.6

Gross mass

goods plus the included packaging; technically, mass under a definition that includes the stated container and packing materials.

Read in context: 4.1.6

H

Half-even

choosing the even final digit at a tie; technically, rounding halfway cases to the nearest value whose retained least significant digit is even.

Read in context: 4.4.6

Historical value

a value tied to an earlier event or agreement; technically, information whose interpretation must not be replaced by an unrelated current-state value.

Read in context: 2.3.6

I

Idempotency

repeating the operation without repeating its intended effect; technically, stability of the declared effect under repeated application within the operation's identity and failure model.

Read in context: 5.4.6

Identifier

a reference to a particular thing; technically, a value or tuple interpreted within a defined namespace and lifecycle.

a value used to refer to something; technically, a designation interpreted within an issuing namespace and its lifecycle rules.

Read in context: 2.4.6, 5.1.6

Identity conflict

one ID attached to incompatible content; technically, a violation of the mapping from an identifier to the immutable or otherwise controlled claim it is supposed to name.

Read in context: 5.4.6

Identity history

the record of how associations changed; technically, retained versions or events describing identifier mappings and their applicable decision boundaries.

Read in context: 5.6.6

Implementation detail

how a particular system works; technically, behaviour that must not be assumed to apply to every conforming or competing system.

Read in context: 1.6.6

Indication

what the instrument shows; technically, the quantity value supplied by a measuring instrument or system.

Read in context: 4.1.6

Instant

one point on a timeline; technically, a temporal position distinguishable from its local display representation.

Read in context: 3.5.6

Integer

a whole number; technically, a value in the set of signed whole numbers, represented with implementation-specific limits.

Read in context: 3.2.6

Integrity check

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

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

Read in context: 6.6.6

Interface

how one component asks another to do something; technically, the exposed operations and their input, output and error contracts.

Read in context: 1.3.6

Invariant

a rule that must keep holding; technically, a predicate required of accepted states or permitted transitions.

a rule that must remain true — a property the chosen model preserves across accepted operations.

a condition the system is meant to preserve — a stated property that every accepted state transition must maintain.

a condition that must keep holding — a predicate preserved across the operations within its specified scope.

a rule every accepted state must preserve — a condition maintained by constraints and operation protocols across permitted transitions.

a rule accepted operations must preserve — a predicate over the relevant system state that the protocol is designed to maintain.

Read in context: 1.1.6, 6.2.6

L

Lineage

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

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

Read in context: 2.5.6

Logical clock

a counter that helps preserve event order — a timestamping mechanism that need not represent physical time.

Read in context: 6.3.6

Lossy conversion

a change that forgets information; technically, a mapping from which the original value cannot always be reconstructed.

Read in context: 3.1.6

M

Materialised projection

an answer stored for later reading — a persisted derived view rather than a query result recomputed on every request.

Read in context: 6.4.6

Measurand

the quantity you intend to measure; technically, the quantity defined as the target of a measurement, with the necessary object and condition information.

Read in context: 4.1.6

Merge decision

an explicit association of records; technically, a versioned decision to treat selected representations as referring to one entity under stated evidence and rules.

Read in context: 5.6.6

Missingness policy

what absence means and how it is handled; technically, the domain rules for rejecting, preserving, classifying or analysing unavailable values.

Read in context: 3.6.6

Model

a deliberately limited representation; technically, a set of concepts and rules chosen to answer particular questions.

Read in context: 1.6.6

N

Namespace

where a name has its meaning; technically, the scope within which identifiers are interpreted and uniqueness is defined.

the context in which an identifier has meaning; technically, an issuing or interpretive scope within which names are assigned and resolved.

a named container used to distinguish object names — in PostgreSQL, a schema that can contain tables and other database objects.

Read in context: 2.4.6, 5.3.6

Natural key

a key drawn from the work's own attributes; technically, an identifying set of domain attributes rather than a record-only identifier introduced by the receiving model.

Read in context: 5.2.6

Normalisation

a specified way to standardise equivalent text sequences; technically, conversion to a Unicode normalisation form, not general identity verification.

Read in context: 3.4.6

NULL

a specific SQL missing/unknown marker; technically, a marker with defined behaviour in comparisons, logic, constraints and aggregates.

Read in context: 3.6.6

O

Observation

something someone or something noticed; technically, an acquisition of information under a stated method and context.

something recorded as seen or measured — evidence that need not itself authorise or cause a state-changing business action.

Read in context: 1.2.6, 6.1.6

Overflow

exceeding a representation's capacity; technically, an operation whose mathematical result lies outside the representable range.

Read in context: 3.2.6

P

Parser

a shape reader; technically, software that recognises a representation according to its grammar.

Read in context: 2.2.6

Partial order

an order that does not compare every pair — a relation under which some distinct events remain unordered.

Read in context: 6.3.6

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: 4.3.6

Precision, matching

the share of accepted matches that are correct; technically, true positives divided by all positive predictions, not measurement precision from Chapter 4.

Read in context: 5.5.6

Primary key

the table's chosen identifying key; technically, a selected set of columns whose combined values uniquely identify each row and cannot be null under the relevant database rules.

the chosen required identifier for a table — the designated key used as a principal row identity in the current schema.

Read in context: 5.1.6

Projection

an answer built from other records — a derived representation produced by applying transformation rules to source data.

choose the fields needed — a relational or SQL operation producing selected expressions rather than every available attribute.

Read in context: 6.1.6

Projection version

which interpretation produced the answer — an identifier for the relevant transformation or state representation.

Read in context: 6.4.6

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: 2.5.6

Q

Quantisation

fitting values to allowed steps; technically, mapping values to a discrete set of representable or permitted outputs.

replacing a continuous range with steps; technically, mapping input values to discrete output levels.

Read in context: 3.3.6, 4.4.6

Query

a precise question; technically, an operation expressing a requested result over data.

a request for a result from data — an expression evaluated under a data model and its language semantics.

Read in context: 1.4.6

R

Range

the smallest through largest supported values; technically, the domain bounds of a representation or declared type.

Read in context: 3.2.6

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: 5.5.6

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: 1.2.6, 6.5.6

Reconstruction boundary

how far an answer can be rebuilt — the scope supported by retained inputs, snapshots and interpretation rules.

Read in context: 6.6.6

Recording time

when a defined system boundary recorded the information; technically, a timestamp whose acquisition point must be specified.

Read in context: 2.5.6

Replacement entry

the newly accepted corrected representation — a record distinguished from the earlier entry it supersedes or repairs.

Read in context: 6.5.6

Reproducible example

an example another reader can check; technically, one supplied with sufficient inputs, procedure and environment information to repeat its stated observation.

Read in context: 1.6.6

Resolution

the smallest distinguishable change; technically, the change in the measured quantity that causes a perceptible change in the corresponding indication under the stated conditions.

Read in context: 4.3.6

Restoration

bringing back a stored copy — a recovery operation that must account for the copy's age and subsequent required changes.

Read in context: 6.6.6

Retention

how long and why information is kept; technically, a policy governing continued storage and eventual disposition for defined record classes.

Read in context: 2.6.6

Retention boundary

which history remains available — a stated limit on kept records, detail, time or access.

Read in context: 6.6.6

Reversal

an opposite recorded effect tied to an earlier entry — a constrained correction operation whose scope must be defined.

Read in context: 6.5.6

Revision

a particular state after a change; technically, an identifiable version of a record or artefact.

Read in context: 2.6.6

Rollback

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

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

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

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

Read in context: 1.5.6

Rounding

reporting a nearby value on a chosen scale; technically, mapping a value to a specified representable set under a tie-breaking rule.

Read in context: 4.4.6

Rounding policy

the chosen way to handle non-exact results; technically, the rule and stage used to select an allowed output value.

Read in context: 3.3.6

S

Sample standard deviation

the observed spread of a sample; technically, the square root of the sum of squared deviations from its mean divided by n - 1.

Read in context: 4.3.6

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: 4.5.6

Scale

what one stored step represents; here, the fixed relationship between an integer amount and its displayed unit.

Read in context: 3.2.6

Schema

the declared structure; technically, a definition of fields, types, relationships and applicable structural rules.

Read in context: 2.2.6

Schema version

which contract applies; technically, an identifier for a defined revision of the schema and its interpretation.

the named edition of a record contract — an identifier for the supported structure and meaning, distinct from a software release.

Read in context: 2.2.6

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: 4.5.6

Semantics

what a representation means; technically, the interpretation and permitted consequences assigned to it.

Read in context: 2.1.6

Sequence gap

an expected predecessor is missing — a discontinuity that prevents this strict projector from claiming a complete prefix.

Read in context: 6.3.6

Side effect

an action beyond calculating the returned value — an external change that must not accidentally be repeated during reconstruction.

Read in context: 6.4.6

Significant figures

the digits deliberately retained in reporting; technically, a numerical presentation convention that does not alone specify a full uncertainty model.

Read in context: 4.4.6

Snapshot

a saved position at a known boundary — a serialised state with enough metadata to resume a compatible interpretation.

Read in context: 6.4.6

Split

revising an association into distinct entities; technically, a correction of mapping relationships whose downstream effects may require separate repair.

Read in context: 5.6.6

SQL

a language for working with data; technically, Structured Query Language and its standardised and product-specific features.

Read in context: 1.3.6

Stale request

a request based on an older position — an operation whose assumed version no longer matches the accepted state.

Read in context: 6.2.6

Standard uncertainty

uncertainty expressed like a standard deviation; technically, measurement uncertainty expressed as a standard deviation under the stated model.

Read in context: 4.6.6

State

the position the model currently holds — a representation at a defined logical boundary.

Read in context: 6.1.6

State transition

an allowed move between positions — a specified change from one model state to another.

one defined move from an old state to a new state — an operation with explicit preconditions, writes and outcomes.

Read in context: 6.2.6

Stock adjustment

an explicit change to the expected stock record — a domain operation distinct from a goods movement or a count observation.

Read in context: 6.1.6

Storage type

the engine’s representation category; technically, a supported type or storage class with documented operations and limits.

the kind of value an engine stores in a field — an implementation-level representation with defined conversion and operation rules.

Read in context: 3.1.6

Stream sequence

a record's position in one accepted history — a scoped ordering value used for interpretation or concurrency checks.

Read in context: 6.3.6

Surrogate key

an extra identifier created for the record; technically, a key whose principal role is row identity rather than existing business meaning.

Read in context: 5.2.6

Syntax

the shape of a representation; technically, the grammar used to recognise valid constructions.

Read in context: 2.1.6

T

Tare mass

the packaging amount taken away; technically, the measured or assigned container mass used in a net-mass calculation.

Read in context: 4.1.6

Target population

everything the question is meant to describe; technically, the defined set of units about which an inference or descriptive claim is intended.

Read in context: 4.5.6

Tenant scope

the customer boundary in a shared service; technically, the domain boundary used to associate records and permitted operations with a tenant, requiring access control beyond identification.

Read in context: 5.3.6

Transaction

a controlled unit of database work; technically, operations managed together under an engine's atomicity and related semantics.

a defined group of database work — a unit with a controlled commit or rollback boundary.

Read in context: 1.5.6

Type

the kind of value; technically, a classification that governs representation and permitted or defined operations.

Read in context: 3.1.6

U

Uncertainty budget

the explanation behind the uncertainty figure; technically, a statement of contributing uncertainty components, their evaluation and their combination in the measurement model.

Read in context: 4.6.6

Unique constraint

a rule against repeated key values; technically, database enforcement that the relevant value combination cannot appear in conflicting rows under its null and comparison rules.

a prohibition on repeated constrained values — an invariant over one column or a composite value under the engine's comparison and NULL rules.

Read in context: 5.2.6

Unit

the agreed size of a numerical step; technically, a reference quantity used to express values of quantities of the same kind.

Read in context: 4.2.6

UTC offset

the difference from UTC in a representation; technically, a signed hours-and-minutes offset, not a complete named-zone rule set.

Read in context: 3.5.6

UUID

a standardised large identifier; technically, a 128-bit identifier whose version and variant determine the interpretation and generation rules.

Read in context: 5.2.6

V

Validation

checking the stated rules; here, testing whether a representation meets declared structural or domain requirements.

Read in context: 1.2.6

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