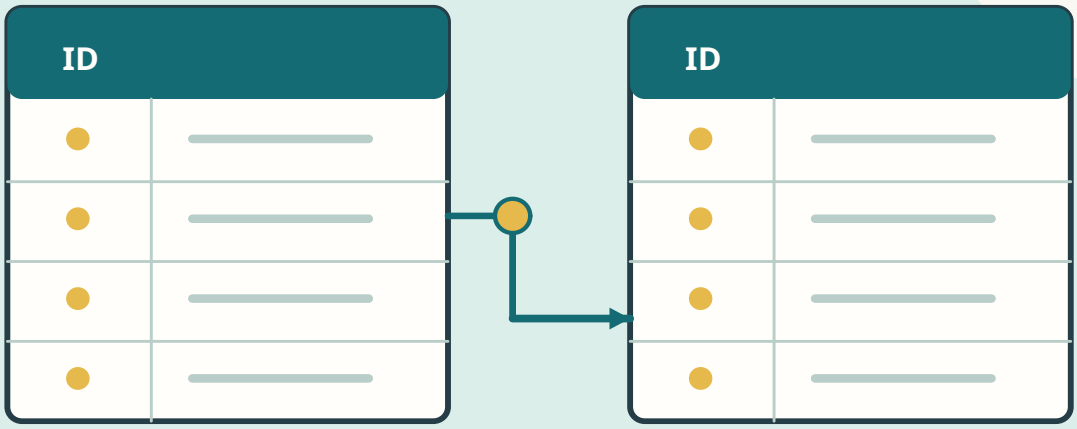
UNMATCHED SQL RECORDS

Finding Unmatched Records with SQL



MATCH / UNMATCHED

Finding Unmatched Records with SQL

Learning outcomes

- Find all five fact records without a matching reference key.
- Use a nonnullable matched key as the outer-join absence indicator.
- Report unmatched keys, counts and example identifiers without misclassifying valid matches.
- Define absence relative to a named population, relationship and snapshot.
- Separate NULL input keys, empty input keys, absent reference rows and incomplete reference attributes.
- Explain NOT IN counterexamples and verify anti-match queries by identifier set.
- Reconcile affected rows, raw-key groups and signed units without conflating their meanings.
- Challenge exception reports with negative controls and bounded evidence claims.

Study the foundations, work through the applied units, complete the practice tasks, and then attempt the assessment. Use only the supplied fictional practice data. Keep your original inputs and your revised work separate.

Core concepts

Define absence on the relationship, not the label

A missing-reference check asks whether a source record can find a corresponding key in a reference table. It is not the same as asking whether the reference record has a complete description. The catalog intentionally contains a valid product code whose label is NULL. Receipts for that code are matched and should not appear in the exception list. If you test the label for NULL, you confuse missing descriptive data with an absent reference record.

Choose a reference-side column that is guaranteed non-NULL when a real catalog row exists. In this fixture that indicator is catalog.product_code, declared both primary key and NOT NULL. After a left join, a NULL in that matched-side key indicates that no catalog row was attached. The source product_code is different: it may itself be NULL or empty, and its original representation should remain visible in the result.

State the output grain as one row per unmatched receipt. Several receipts can share the same unmatched product code. Those are separate business records and all must be found. The later grouped summary is a second artifact with one row per unmatched key representation. Keeping these grains separate prevents an analyst from claiming that four unmatched keys means only four affected receipts.

Preserve the source side with an outer join

Begin with receipts on the left and catalog on the right. A LEFT JOIN preserves each receipt even when no catalog row matches the ON condition. The unmatched row receives NULL values in the catalog-side fields. Filter those joined rows using the catalog's nonnullable key indicator. The source receipt identifier remains available so a reviewer can trace each exception to its original record.

An inner join would remove precisely the unmatched records you are trying to find. A filter on a catalog description can also defeat the purpose by excluding the NULL-extended rows or admitting valid records with missing labels. Keep the relationship condition in ON and the missing-match test in WHERE. Read them aloud as “match the product codes, then retain receipts for which no catalog key was found.”

Do not normalize, fill or replace the original keys during this diagnostic exercise. An empty string and NULL both fail to find a catalog match here, but they are different input states that may require different operational remediation. Converting them to a single placeholder too early loses evidence. A presentation label can describe the state in an extra column, but the original key should remain in the detail output and the summary should distinguish its representations.

Cross-check using an existence test

NOT EXISTS offers another way to express the same question: keep a receipt when there is no catalog row whose key equals that receipt's product code. It is useful as a cross-check because it reasons about the existence of a match rather than relying on an outer-join indicator. Compare the actual receipt-ID sets from both approaches, not just their counts. Agreement increases confidence when the two implementations are reviewed independently.

Be cautious with NOT IN when NULL can appear in the compared set. Three-valued logic can make the result different from an ordinary “not a member” intuition. This course does not require you to memorize every NULL truth-table case. The important habit is choosing an expression whose behavior matches the missing-reference question and testing explicit NULL cases before trusting the output.

The practice dataset includes two receipts for the same unknown code, a second unknown code, a NULL code and an empty code. It also includes repeated valid codes and a valid reference with no label. These are deliberate controls. Check that every expected unmatched source record is present, that repeated unmatched codes are not collapsed in the detail report and that the valid code with a missing label is excluded from the exception set.

Make exceptions actionable without repairing them

A useful exception pack contains a detail list and a grouped summary. The detail list provides all affected receipt IDs, original keys and units. The summary counts affected receipts by key and includes an example ID so an operator can inspect a representative record quickly. A MIN(receipt_id) example is deterministic and simple; it is an example, not proof that only one receipt is affected.

Reconcile the grouped receipt counts to the detail-list row count. Also reconcile matched plus unmatched receipt counts to the source population, using the same matching rule. This checks that no source records were silently discarded. Do not count catalog labels as the matched-row measure, because the deliberately missing label would undercount a valid match.

Your role in this exercise ends with a verified diagnostic report. Adding invented catalog products, assigning replacement codes or deleting receipts would be unauthorized source changes and could destroy useful evidence. State that the exceptions need review by the data owner rather than pretending to know which input is wrong. The practical submission should contain both implementations, the summary and a note distinguishing absent references from incomplete reference attributes. That distinction is the central quality safeguard in this task.

Practice protocol and submission standard

All organizations, identifiers, transactions and observations in this pack are fictional. Use only the supplied local database, not a workplace connection. Open it through the read-only runner in the practice folder. Save your SQL in a separate UTF-8 text file and retain the result headers. A query that returns plausible numbers is not automatically correct: compare the output against the stated population, grain, field order and edge cases. Keep the original database unchanged. The exercise asks for evidence of a repeatable method, not a screenshot alone. Submit the SQL, the exported result and a short explanation of checks. This practice course is not an accredited qualification.

Worked example

Title: Missing description is not a missing reference

Sql

```
WITH ref(code,label) AS (VALUES ('A',NULL)), fact(id,code) AS (VALUES ('X','A'),
    ('Y','B')) SELECT f.id FROM fact f LEFT JOIN ref r ON f.code=r.code WHERE
    r.code IS NULL;
```

Explanation: Only Y is unmatched. X has a real reference row even though its descriptive label is missing.

Foundation practice

Scenario

A fictional warehouse receives product receipts from an intake system. Some product codes are absent from the current catalog, and one valid catalog record has a missing display label. The task is a missing-reference report, not a source-data correction.

Database file: 3H-AI-0147_practice/practice.sqlite

Schema file: 3H-AI-0147_practice/schema.sql

Runner file: 3H-AI-0147_practice/run_query.py

Readme file: 3H-AI-0147_practice/README.txt

Tables

catalog

product code	label
P01	Clip Pack
P02	NULL
P03	Binder
P04	Marker
P05	Folder

Csv file: catalog.csv

receipts

receipt id	product code	units
RC01	P01	10
RC02	P02	6
RC03	P03	12
RC04	P90	3
RC05	P04	5
RC06	P90	7
RC07	P91	2
RC08	NULL	4
RC09		1
RC10	P05	9
RC11	P02	8
RC12	P01	11

Csv file: receipts.csv

Requirements

Find every receipt with no matching catalog product_code. Treat NULL and empty product_code as unmatched because the catalog contains neither. Return receipt_id, original product_code and units. Produce a grouped exception summary with original unmatched key, receipt_count and one example receipt_id per key, preserving the distinction between NULL and empty text. Verify exactly five missing-reference receipt rows and no valid matched rows, including those whose catalog label is NULL.

Null note

JSON null and SQL NULL mean missing. CSV uses <NULL> as a display marker; empty text remains empty text. No import is required.

Run instructions

Open the practice folder, save one SELECT or WITH ... SELECT statement as my_query.sql, then run python3 run_query.py my_query.sql (or py run_query.py my_query.sql on Windows). The runner opens the supplied SQLite database read-only.

Submission

Saved SQL statements, exported query outputs, specified validation evidence and a short interpretation. Keep the database unchanged.

Foundation tasks

- Exercise 1: Identify a safe matched-side indicator and explain why catalog.label is unsafe.
- Exercise 2: Write both LEFT JOIN and NOT EXISTS diagnostics and compare receipt-ID sets.
- Capstone: Deliver all five unmatched detail rows plus the key/count/example summary, preserving NULL versus empty text and excluding valid label-missing products.

01 Define what is absent before searching for it

The fictional Alder Tool Library records movements between storage locations. A movement is an event with its own identifier and a signed unit change. The location reference is a separate snapshot naming recognized location codes. The investigation asks which recorded movements cannot find a location in that supplied reference. It does not ask whether stock is physically missing, whether a person made an error, or whether the latest location list is historically complete.

Begin by writing a question that could be falsified: for every movement in the supplied snapshot, does at least one reference row have exactly the same location_code? This formulation makes the left-hand population, matching fields and evidence boundary visible. A vague instruction such as find bad locations invites unsupported substitutions. An unrecognized code could be a mistyped source value, an omitted reference entry or an export mismatch. The available relationship test alone cannot distinguish these causes.

All twenty movements are in scope. Retain signed adjustments, observed zeros and movements linked to an inactive location. The active flag describes a supplied reference

attribute; it does not authorize filtering this all-movement report. No date of location validity is provided. Do not infer that an inactive location was invalid when a movement occurred, and do not invent an effective date from a filename or the date on which you run the query.

The detail grain is one unmatched movement. Required fields are movement_id, original location_code and unit_delta, ordered by movement_id. The grouped report has another grain: one original unmatched key representation. It must contain the key, affected movement count, signed net units and a deterministic example identifier. These are two views of the same exception population, not competing definitions of how many problems exist.

Assets include the full movements and locations tables, a dictionary and a read-only runner. The small tables shown in later lessons demonstrate particular failures; they do not replace the complete inputs for the main task. Keep the original practice receipts case separate. Its product codes and five exceptions remain a foundation exercise, while this workshop tests the same reasoning with a new population and additional failure modes.

Contract element	Meaning
Population	Every supplied movement
Relationship	Exact location_code equality
Detail grain	One affected movement
Reference scope	Supplied location snapshot only
Authority	Diagnose and report; do not repair

E01-X1 Draft the investigation contract

Write the exact detail fields and population rule. Explain how M09/L04 and M10/L02 should be treated before searching for absence. State two claims the supplied data cannot establish.

Submit: A contract plus two named record interpretations.

02 Read source representations without silently repairing them

A missing SQL value, empty text and a text value containing spaces are different observations. The workshop CSV uses the literal display marker <NULL> to encode SQL NULL. An empty CSV field means empty text. M06 and M17 have NULL codes; M07 and M18 have empty text. M13 begins with one space before L01, and M14 uses lower-case l01. The reference contains L01, not either altered spelling.

The case requires exact comparison using the supplied text values. Treating the leading space as irrelevant might be a reasonable proposal for a different workflow, but it is not the current rule. Trimming or uppercasing in the join would change the measured exception

population. Preserve the original field even when you add a display column to make its state easier to see. A display label should never become an undocumented replacement key.

Inspection should distinguish a key representation from a business interpretation. Empty does not prove a deliberate blank; NULL does not prove that no location exists in the real world. Both tell you only what this source recorded. Likewise, the resemblance of l01 to L01 does not establish that they identify the same place. A reviewer may investigate that hypothesis after receiving the evidence, but the diagnostic query must not settle it by assumption.

A useful display approach brackets ordinary text and separately labels missing and empty values. Brackets reveal the space in [L01] without editing it. The labels MISSING_INPUT and EMPTY_INPUT describe states, not repair instructions. When exporting, document whether the result uses JSON null, a CSV marker or a printed label. Otherwise a downstream spreadsheet may collapse two distinct states into visually identical blank cells.

Inspect the schema as well as the samples. location_code is explicitly NOT NULL in the reference, and movement_id is a nonmissing unique event identifier. Those declarations support later absence and population tests. A generic assertion that every primary key is automatically nonnullable in every SQLite table is unsafe; rely on the actual declared schema and the independently checked data. Source profiling is useful even when a small fictional fixture already satisfies its constraints.

```
SELECT movement_id, location_code,
       CASE WHEN location_code IS NULL THEN 'MISSING_INPUT'
            WHEN location_code = '' THEN 'EMPTY_INPUT'
            ELSE '[' || location_code || ']' END AS key_display
FROM movements
WHERE movement_id IN ('M06','M07','M13','M14')
ORDER BY movement_id;
```

E02-X1 Make raw keys inspectable

Return movement_id, original location_code and key_state for all movements where the source code is NULL or empty. Use MISSING_INPUT and EMPTY_INPUT, in ID order. Do not include whitespace or case variants in these two categories.

Submit: Runnable SQL, the four output rows and a sentence about M13.

03 Choose an absence indicator that cannot be a real NULL

The central danger in an outer-join diagnostic is confusing an absent reference row with a present reference row whose attribute is missing. In this fixture L02 is a recognized location with a missing label. M02 and M10 match that reference. They must not appear in the missing-location list. A query testing l.label IS NULL would include them along with genuine absence cases, producing false positives that look plausible on a screen full of blanks.

Use the reference-side location_code as the absence indicator because the supplied schema explicitly prohibits NULL in that column. After matching, a real reference row carries a nonmissing key. A NULL in that selected reference key therefore means the outer join had no

matching row. This reasoning depends on the full relationship condition and the declared reference domain. It is not a magical property of a column whose name happens to contain code.

The source-side key answers a different question. m.location_code IS NULL finds movements with missing input codes, but it cannot find M03/L90 or M08/L91. These contain recorded text that fails reference membership. Conversely, a reference key indicator tells you that a match is absent without telling you why. Keep both source representation and relationship status available so the explanation is not reduced to one overloaded NULL flag.

Make a negative control explicit before executing the diagnostic. Predict that M02 and M10 will be excluded, because their missing label is not a missing relationship. Also predict that M09 will be excluded from the exception list because L04 is present, regardless of its active flag. These are more useful controls than a random matched record: they exercise tempting but incorrect interpretations of the source attributes.

A practical review can ask two separate questions about M02: did a location row match, and is its label populated? The answers are yes and no. Both findings may deserve attention in different reports. Combining them under the label unmatched destroys that distinction and inflates the count used to prioritize missing-reference investigation. An accurate report can be narrow; it need not solve every data-quality issue at once.

Movement	Reference exists?	Label known?	Missing-reference exception?
M02 / L02	Yes	No	No
M03 / L90	No	Not applicable	Yes
M09 / L04	Yes	Yes	No

E03-X1 Measure the nullable-label false positives

Run the wrong l.label IS NULL diagnostic and the correct reference-key diagnostic. Return the extra movement IDs admitted by the wrong query. Explain why each is wrong, not merely that the counts differ.

Submit: Both SQL statements and the false-positive ID list.

04 Build an anti-match detail list that preserves evidence

An anti-match report retains facts for which a related row cannot be found. Read the LEFT JOIN pattern as two operations: attach any reference row whose code equals the movement code, then keep only movements whose attached nonnullable reference key is missing. Selecting the source movement identifier and original source code ensures the resulting exception can be traced without guessing what the missing reference would have contained.

Do not select only the reference code in an exception list. Every retained reference code is NULL by construction, so that output erases the unknown text you need to investigate. M03 and M05 both contain L90. Their identifiers, signed deltas and shared code are separate pieces of evidence. Selecting DISTINCT location_code would create a useful key inventory but would no longer be the required one-row-per-movement detail artifact.

The reference table has one row per code. This makes each movement either matched once or unmatched. For an existence question, duplicate reference candidates would not create additional unmatched facts, but they could complicate a broader joined report and its matched count. Keep the uniqueness check visible as a precondition rather than trying to use DISTINCT as an unexplained cleanup step. The required diagnostic should be understandable without hiding multiplicity.

Order the detail by movement_id. Ordering is presentation, not classification, yet deterministic output makes review and comparison simpler. Do not rely on the apparent order of rows in a table or a previous export. Keep the column order explicit as well. When reviewers compare files, an undocumented column rearrangement can be mistaken for changed data or make a script inspect the wrong field.

The worked pattern below uses a deliberately tiny inline fixture. It shows why retaining the source key matters: the exception carries B, not only a NULL reference field. Apply the pattern to the complete warehouse fixture rather than treating the example result as the final answer. A reliable analyst can explain each retained record in the language of the relationship condition, and can explain each excluded control without appealing to a memorized expected count.

```
WITH ref(code, label) AS (VALUES ('A', NULL)),
     fact(id, code, units) AS (
       VALUES ('W1','A',5), ('W2','B',2), ('W3',NULL,0)
     )
SELECT f.id, f.code, f.units
FROM fact AS f
LEFT JOIN ref AS r ON r.code = f.code
WHERE r.code IS NULL
ORDER BY f.id;
```

E04-X1 Return the complete location exception detail

Write the LEFT JOIN anti-match query against movements and locations. Return movement_id, original location_code and unit_delta, sorted by ID. Explain why repeated L90 records stay separate.

Submit: The complete ten-row detail result and runnable SQL.

05 Cross-check existence rather than copying the same join

NOT EXISTS asks whether the correlated subquery finds no qualifying reference row. The correlation is essential: the inner location code must be compared with the current outer movement code. A subquery that merely selects any location without a correlation answers whether the entire reference table is empty. Because this fixture has locations, that mistaken query would return no movements and could be misread as excellent data quality.

The selected expression inside EXISTS does not need to be the reference label. SELECT 1 is a conventional way to show that the question concerns row existence rather than an attribute value. A matching row with a NULL label still exists. This explains why changing SELECT 1 to SELECT label does not make missing labels become missing matches. The predicate in the subquery, not the chosen display value, determines which rows qualify.

For a NULL source code, equality with every reference code fails to establish a true match. The correlated subquery returns no rows, so NOT EXISTS retains that movement. This is exactly the workshop contract: missing input keys remain visible as unmatched. No artificial code needs to be inserted into the source or reference to force this behavior.

Compare the two implementations by complete movement-ID sets. Equal row counts are necessary but weak: one false positive and one missing exception can cancel. A symmetric comparison asks for IDs in the first result but not the second, and then IDs in the second but not the first. Both difference lists should be empty. Confirm that each result itself has one row per movement before relying on a set comparison, because a set can conceal repeated rows.

Independence is a matter of reasoning, not just syntax. If both queries copy an unauthorized condition such as active=1, they can agree on the wrong population. First compare each against the written contract and negative controls, then use agreement as additional evidence. A second query is most useful when it could disagree for an identifiable reason and when the reviewer knows which edge cases distinguish the implementations.

```
SELECT m.movement_id
FROM movements AS m
WHERE NOT EXISTS (
  SELECT l.label FROM locations AS l
  WHERE l.location_code = m.location_code
)
ORDER BY m.movement_id;
```

E05-X1 Compare identifiers, not only counts

Write a correlated NOT EXISTS version of the required detail query. Compare it with E04 using two EXCEPT queries or an equivalent exact identifier comparison. Report both directional differences and state a shared-error limitation.

Submit: Runnable alternative, exact matching IDs and two empty differences.

06 Explain the two different NULL traps in NOT IN

NOT IN can look like a shorter replacement for an anti-match query, but its NULL behavior must be part of the design. There are two separate hazards. A NULL among values returned by the subquery can prevent an unlisted non-NULL candidate from producing a true result. A NULL candidate on the left can also fail to produce true when the compared set is nonempty, even if the reference itself contains no NULL values.

Use the membership_probe and candidate_probe tables to make these hazards concrete. The reference probe contains A and NULL. Candidate V1 contains A, V2 contains B, V3 contains NULL and V4 contains empty text. With code NOT IN the unfiltered reference probe, none of the candidates is retained. V1 is known to be present. The other candidates do not become known nonmembers simply because the reference includes an unknown comparison value.

Filtering NULL out of the subquery is an improvement for nonmissing candidates, not a complete solution for this task. V2 and V4 then survive, but V3 still does not. If the contract says missing source keys count as unmatched, excluding V3 changes the report. A carefully written NOT IN expression can add an explicit source IS NULL branch and exclude NULL reference values, but it becomes more complicated than the direct NOT EXISTS expression.

The empty-set case is another reason to test rather than memorize a slogan. In SQLite, a NOT IN comparison against an empty subquery is true even for a NULL left operand. Thus the same query shape can treat a missing candidate differently when the reference population becomes empty. Teach the relevant truth states and verify the intended data conditions; do not describe NOT IN as always wrong or universally interchangeable with NOT EXISTS.

These probes are separate teaching data, not additional location rows. Their reference permits NULL deliberately so that the hazard is observable. The main locations reference prohibits NULL, yet a naive NOT IN against it still omits M06 and M17 because their source keys are missing. A successful query on clean keys is not evidence that the nullable-input contract has been satisfied.

Candidate	Probe reference A, NULL	Reference filtered to A
A	Not retained	Not retained
B	Not retained	Retained
NULL	Not retained	Not retained
Empty text	Not retained	Retained

E06-X1 Test all three membership designs

Run NOT IN against membership_probe, NOT IN after excluding reference NULL, and correlated NOT EXISTS. Return ID lists for each. Then use an empty reference subquery with NOT IN and explain V3.

Submit: Four runnable queries and exact ID lists.

07 Separate affected events from unmatched key groups

An operator may need a work queue grouped by code, while an analyst needs the full affected-event population. Grouping answers how many movements share each original key representation. It does not make the other movements disappear from the evidence. L90 occurs on two movements with different unit deltas, and L91 does too. Report both the affected movement count and the number of distinct groups with unambiguous labels.

NULL requires special attention when counting groups. GROUP BY collects the missing source keys into one group, but COUNT(DISTINCT location_code) ignores NULL. In the main exception population, counting distinct non-NULL key values therefore gives one fewer than counting the rows of the grouped key report. Neither calculation is inherently meaningless; they answer different questions. The mistake is labeling the former as every key representation while silently omitting the missing-input group.

Use COUNT(*) for the number of movements in each group. COUNT(location_code) would report zero for the NULL group despite its affected movements. Include SUM(unit_delta) as a separate signed quantity, not a replacement for record count. The missing-input group contains a positive movement and a zero movement; both contribute to its affected-record count, while the zero contributes nothing to net units.

MIN(movement_id) gives a deterministic example identifier. It is an investigation pointer, not the earliest movement in time. No event timestamps are supplied, and lexical identifier order does not establish chronology. Avoid phrases such as first incident or oldest error unless the source contains the relevant ordering evidence. A representative ID can be useful without carrying a historical claim.

The grouped output should preserve the original key alongside any display state. If a downstream report shows all missing and empty values as blank, include a state field or documented encoding so the groups remain distinguishable. The goal is an actionable summary that can be reconciled to detail, not a prettier table that merges the categories most likely to require different investigations.

E07-X1 Build the exception grouping and diagnose undercounts

Return original location_code, movement_count, net_units and example_id, ordered by key. Also compute COUNT(DISTINCT location_code) on exceptions and the actual number of grouped rows. Explain the difference and show what COUNT(location_code) would do to the NULL group.

Submit: The six-row summary, two group counts and a concrete NULL explanation.

08 Reconcile a partition rather than celebrating one total

A complete existence classification partitions the source population into matched and unmatched movements. Every source event must appear in exactly one part, and neither part may contain extra events. Record-count reconciliation tests that the two counts sum to the source count. Identifier reconciliation strengthens it by checking that their union is the source set and their intersection is empty. Each control has a different purpose.

The signed unit measure needs its own reconciliation. Add the net units of matched and unmatched movements and compare with the source net units. Do not drop negative movements merely to make the summary easier to read. M11 and M19 are matched adjustments, while M10 and M17 are zero movements in different relationship states. A net sum alone cannot establish that all four were preserved because zeros and offsetting quantities can hide omissions.

For this fixture, the whole source has twenty movements and eighty-three net units. The unmatched result has ten movements and twenty-nine net units. The matched result must therefore have ten movements and fifty-four net units. Derive the matched side directly with EXISTS rather than subtracting alone; an independently executed query can expose an accidental mismatch between the two definitions. Then compare the direct results with the arithmetic partition.

A summary-to-detail check is separate from a source-to-partition check. Summing movement_count across the six unmatched key groups should give ten, and summing their net_units should give twenty-nine. Those agreements show that grouping preserved the selected exception population. They do not establish that the exception population was chosen correctly. A wrong selection can still group and reconcile perfectly within itself.

A strong evidence note names the expected relationship, actual result and conclusion for each check. For example, source rows equal matched plus unmatched rows is a passed conservation check, not proof that a source code is factually correct. Keeping the scope of each inference narrow helps a reviewer decide what remains unknown without undermining the useful conclusions already supported by the data.

Population	Movements	Net units
All source	20	83
Matched	10	54
Unmatched	10	29

E08-X1 Prove the population partition

Produce source, EXISTS-matched and NOT EXISTS-unmatched count and sum controls. Reconcile the grouped exception totals. State why removing M17 would escape a net-unit check but fail a record-count check.

Submit: Three exact control rows, grouped totals and the M17 explanation.

09 Keep absence separate from eligibility and inactivity

A location can exist while being inactive. That is a different condition from having no location record. If the requested report changes to no active location match, the matching predicate legitimately changes and M09 becomes an exception. Under the original all-location membership question, M09 is matched. The two results can both be correct when they are labeled with different contracts; they cannot be silently substituted for one another.

Consider a condition active=1 added to the ON clause. The outer join now searches only for reference rows that have both the matching code and active status. A source with an inactive reference receives no qualifying attachment and is reported by the absence test. This does not establish that the code is absent from the complete reference. The name no active match communicates the actual query; missing location would overstate the finding.

Putting active=1 in WHERE after the outer join creates another problem. For rows with no reference attachment, active is NULL, so the predicate is not true. Combining that predicate with the reference-key IS NULL test produces no genuine unmatched rows. The query may return an empty result that looks reassuring even though it has excluded the population it was meant to inspect.

The broader principle is to read every condition in the absence test, not merely the key equality. Does the subquery require a particular status, a nonmissing label or a later event? Each additional condition narrows what counts as a match. That may be precisely what a separate question requires, but the report must disclose it. A relationship is defined by the whole predicate, not the table names alone.

Treat requests to broaden or narrow the diagnostic as changes to the report contract. Preserve the original results, state the altered predicate and compare the affected IDs. This makes a policy discussion concrete: the active-only version adds M09, rather than vaguely finding more bad data. No database repair is needed to show that distinction, and the comparison does not grant authority to retire or reactivate a location.

```
SELECT m.movement_id
FROM movements AS m
LEFT JOIN locations AS l
  ON l.location_code = m.location_code AND l.active = 1
WHERE l.location_code IS NULL
ORDER BY m.movement_id;
```

E09-X1 Compare two explicitly different absence questions

Return IDs with no active location match using NOT EXISTS. Compare them with the all-location exception list. Then execute the incorrect WHERE active=1 plus missing-key query and explain its empty result.

Submit: Eleven active-only exceptions, the added ID and a concrete explanation.

10 Design controls that can reject a plausible wrong report

A useful control should have a named failure it can detect. Re-running a query without changing anything checks repeatability, but it does not challenge the query's meaning. The workshop contains deliberate negative controls: matched NULL-label records, a matched inactive location, missing input keys, empty keys, formatting variants, repeated unknown codes and zero movements. Each distinguishes a plausible wrong implementation from the intended one.

An ID-set comparison can catch an error that survives both row count and net-unit totals. Imagine a candidate that removes unmatched M17, whose delta is zero, and inserts matched M10, whose delta is also zero. The candidate still has ten rows and twenty-nine net units. A totals-only reviewer would pass it. An identity review finds missing M17 and unexpected M10. This is a small counterexample to the claim that count and sum prove the exception list is correct.

Mutation testing deliberately changes one piece of query logic and checks that the evidence rejects it. Replace the safe indicator with label, drop NULL source codes, normalize key spelling, or add an active-only condition. Each mutation has a concrete expected effect on this fixture. This is not an instruction to modify the frozen database. It is a way to assess the sensitivity of the proposed checks using alternative read-only queries or temporary in-memory counterexamples.

Record what a control cannot detect as well. A count detects losing M17, but not replacing it with M10. A sum detects removing positive M03, but not a zero event. An exact ID list detects both of those defects, yet it does not verify that the reported unit values are copied correctly. For a complete detail contract, compare identifiers and the required fields or preserve the original values directly in the query.

Avoid building a control by copying the candidate's suspected condition into both sides. If both the report and control normalize location codes, they may agree while violating the raw-key contract. The independent manual key inventory and CSV representation check provide a different route to the answer. Independence is strongest when the reviewer can identify which assumption the control does not share with the candidate.

Intentional defect	Evidence expected to reject it
Use nullable label	Unexpected M02 and M10
Drop NULL source keys	Missing M06 and M17

Intentional defect	Evidence expected to reject it
Trim and uppercase	Missing M13 and M14
Swap zero IDs	Missing M17, unexpected M10

E10-X1 Audit a balanced but wrong exception list

A candidate equals the correct ten-row detail except M17 is replaced by M10, keeping original values. Report its count and net units, the two directional ID discrepancies and the verdict. State which coarse checks still pass.

Submit: A concrete control finding, not a general warning.

11 Interpret zero exceptions and empty populations honestly

An empty exception result is meaningful only when its population and matching rule are known. If a source contains zero movements, a missing-reference query should return zero exceptions, but that does not demonstrate that a reference is complete. There were no source relationships to test. If the reference contains zero rows while movements remain, every movement is unmatched under the existence rule, including a NULL source key.

Distinguish an empty result set from an aggregate row containing zero or NULL. A plain detail SELECT over no exceptions has zero rows. COUNT(*) over that same empty population returns one row containing zero, while SUM(unit_delta) returns NULL because there are no contributing values. A presentation may display a documented zero net for an empty exception population, but it should preserve the count so that absence of records is not confused with observed values summing to zero.

Observed zeros are different. M17 is an actual unmatched movement with unit_delta zero. Its detail row must remain visible. A subset containing only M17 has one record and a net sum of zero. A subset containing no movements has count zero and raw SUM NULL. These outputs carry different evidence about activity even if a polished report displays both amounts as zero.

A populated source can also be entirely matched. A two-record test with codes A and A against reference A returns no exceptions and two matched records. That supports complete membership for those two test records, not a claim about unseen codes or future batches. State the inspected population and snapshot rather than saying no problems exist. A query is only as broad as the supplied inputs and contract.

These tests are useful during review because edge states expose accidental reliance on nonempty outputs. A script that assumes there is always at least one exception may fail on a clean dataset. A manual reviewer may mistake no data for a successful control. Explicitly testing empty source, empty reference and a zero-valued exception gives operational meaning to each state without needing to change the real fixture.

```
WITH empty_values(delta) AS (
```

```
  SELECT unit_delta FROM movements WHERE 0
)
SELECT COUNT(*) AS rows_seen, SUM(delta) AS raw_net,
       COALESCE(SUM(delta), 0) AS display_net
FROM empty_values;
```

E11-X1 Contrast empty and observed-zero states

Write three controls: no movements; only M17; and all movements against an empty reference subquery. For each return count and raw SUM(unit_delta). Explain what can be concluded from each.

Submit: Exact outputs (0, NULL), (1, 0) and (20, 83), with SQL.

12 Transfer absence logic to missing acknowledgments

Not every unmatched-record question is a lookup against a code catalog. The notices table contains four issued notices, and acknowledgments contains recorded responses. The separate question is which notices have no acknowledgment. A notice may receive several acknowledgments, so this is naturally an existence test rather than a request to enrich each notice with exactly one response.

The relationship now uses notice_id, not location_code. N1 has two acknowledgments, including K1 whose comment is NULL. N3 has one acknowledgment. N2 and N4 have none. The validity of the notice's location is a separate issue: N3 uses unrecognized L90 but has been acknowledged. N2 uses recognized L02 but has not. Combining these ideas would misclassify both notices and confuse the workflow.

A missing response comment is not a missing response row. K1 exists regardless of its comment, and K2 gives N1 an additional acknowledgment. A label-like nullable field is therefore still an unsafe absence indicator. If you use a LEFT JOIN for this relationship, test ack_id, which is explicitly nonnullable, not comment. NOT EXISTS expresses the no-response question directly and does not multiply a notice because there are several responses.

Without timestamps, you cannot say that N2 is overdue, late or ignored. The snapshot shows no recorded acknowledgment, not how long the notice has waited or what happened outside the supplied system. An escalation policy would need an authorized deadline, an observation time and the relevant event times. Do not manufacture urgency from the absence of a row alone.

This transfer exercise tests whether the method is understood beyond memorized table names. Identify the preserved population, matching condition, reliable presence indicator and permitted conclusion again. The required output is one row per unacknowledged notice, retaining its original location_code for context. It is not a list of bad locations and not a count of response messages. The distinction lets a reviewer route the finding to the appropriate process without conflating independent questions.

Notice	Location code	Acknowledgment evidence
N1	L01	K1 and K2
N2	L02	None supplied
N3	L90	K3
N4	NULL	None supplied

E12-X1 Find unacknowledged notices without inferring lateness

Use NOT EXISTS to return notice_id and original location_code for notices with no matching acknowledgment. Reconcile acknowledged and unacknowledged notice counts to four, counting notices rather than messages. State why N3 is excluded and why no overdue claim is supported.

Submit: Two exact detail rows, notice-level counts and limitations.

13 Build a review queue without claiming a repair decision

The dispatches and destinations tables form an independent transfer case. There are six dispatch events. A destination reference may have a missing label, and dispatch units can be zero or signed. The task is to identify dispatches without a destination-code match using exact values, then group the exceptions by their original key representation. Do not apply the warehouse movement counts to this new case.

A useful exception queue separates observed state from possible explanation. MISSING_INPUT and EMPTY_INPUT can classify the source-key representation. UNKNOWN_CODE can describe a recorded nonempty code that has no match. Those descriptions remain diagnostic. A possible explanation such as late reference loading must be labeled as a hypothesis and must not be written back into the source as a fact.

Prioritization by affected-record count is possible without inventing monetary or operational risk. A code shared by several exceptions affects several dispatch records. That does not prove it is more costly than a single unknown code, because no cost model is provided. Net units also have limits: signed quantities may offset and zero events may still require investigation. Present the measures and let the authorized owner decide how to respond.

A reproducible handoff includes the raw key, affected IDs, count, signed net units, matching rule and source snapshot name. It should distinguish the underlying data from the analyst's explanatory prose. In a real organization, sending or changing records would require the relevant authorization; this exercise stops at preparing a fictional local review pack. No new destination should be inserted simply to make the exception list disappear.

Before judging the transfer case, predict its negative control: D01 matches R1 even though R1's label is NULL. D03 is an actual zero-unit dispatch with a missing input code and must remain in the exception detail. D04 carries empty text and a negative unit adjustment. These

three records test relationship existence, source missingness and measure preservation independently. A sound solution will explain each rather than relying only on the final number of rows.

E13-X1 Prepare the dispatch exception pack

For dispatches versus destinations, return dispatch_id, original destination_code and units with LEFT JOIN and NOT EXISTS. Produce key/count/net/example summary, reconcile all/matched/unmatched counts and units, and state an evidence-only review note.

Submit: Exact detail, summary, controls, both implementations and a bounded note.

14 Deliver falsifiable evidence and a bounded conclusion

The final movement review pack should let another analyst reproduce not just the output but the basis for accepting it. Save the two detail queries, the raw-key summary, the population controls and the counterexample results as separate readable files. Preserve result headers and explicit ordering. Include the exact local fixture identity so that a later run does not accidentally compare a different location snapshot with the original movement extract.

Use the supplied read-only runner rather than opening a workplace database. It opens the local fixture in read-only mode and rejects operations outside its permitted query actions. Read-only access protects the original evidence; it does not prove that a SELECT implements the right business rule. A wrong report can be produced without changing a single byte, which is why semantic checks remain necessary alongside file-integrity checks.

An acceptance note should state what was tested: all twenty supplied movements, exact raw-code matching, ten exceptions, six raw-key groups, twenty-nine unmatched net units and the matched controls. It should also name the negative controls that stayed out of the exception list, including M02, M09 and M10. Mention the zero-for-zero counterexample because it shows why identity checks were required beyond aggregate reconciliation.

State limits plainly. The analysis does not determine the true intended code, physical stock loss, historical location validity, lateness or permission to alter source records. It establishes a reproducible missing-reference result for the supplied snapshots under the stated equality rule. This narrow conclusion is useful precisely because a reviewer can inspect its inputs and challenge its conditions.

When a later reference snapshot arrives, rerun rather than silently overwriting the earlier report. Keep the snapshots and conclusions distinguishable. A disappearance from the exception list might reflect a newly supplied reference row, a source correction or a changed matching policy. Without comparing those inputs you cannot identify the cause. A stable query plus explicit snapshot identity makes changes explainable, but it does not eliminate the need for contextual review.

Finally, keep the diagnostic scores separate from the overall practical rubric. Individual exercises reveal specific misconceptions; the final submission should demonstrate an integrated method. A complete exception pack is not a collection of screenshots with

matching totals. It is a set of executable statements, exact outputs, independent controls and appropriately limited interpretations that another reviewer can use to reach the same conclusion.

E14-X1 Submit the complete movement review pack

Assemble the main detail results, six-group summary, source partition, NULL membership probes, balanced-error finding and a final acceptance note. Include one concrete failure that each selected control could detect. Do not alter any fixture or infer an event date.

Submit: Runnable SQL and exact rows from E04-E11, plus an evidence matrix and conclusion.

15 Interpret a changed reference without inventing a history

An exception is relative to the reference used in the test. Suppose a reviewer asks what would happen if L90 were present in a proposed reference. This is a counterfactual membership question, not permission to create the location. A read-only query can define a proposed reference as the supplied codes plus L90, then repeat the same existence test against the unchanged movements. Keep the original result alongside the proposed result.

The comparison isolates one declared change. Because the source movements and matching rule stay fixed, any removed exceptions must be movements whose codes match the added reference value. The result does not establish that L90 is a valid real location or that it should have existed earlier. It establishes only how the report responds to the stated hypothetical reference addition. This is valuable evidence for a review meeting without pretending that the decision has already been approved.

Compare identifier changes before comparing totals. If two movements disappear from the exception set, name both and retain their original quantities. A decrease of ten net units could otherwise be mistaken for one corrected ten-unit movement. The example demonstrates why event-level evidence remains useful even when the proposed change is defined at reference-key grain. Several facts can be affected by one reference change.

A different scenario would allow both the movement extract and reference snapshot to change. Then a smaller exception count could reflect several causes: new reference entries, corrected source keys, removed source events or a different population filter. The count alone cannot attribute the improvement to a particular action. Hold one component fixed where possible, or compare both input snapshots before explaining the difference. Do not convert a correlation between two report runs into an unsupported history of repairs.

A disciplined comparison records the original rule, proposed change, removed and added exception IDs, and the remaining limitations. Preserve the two outputs rather than overwriting the first. If an owner later authorizes a real reference update, the operational workflow can use this evidence, but authorization and execution are separate from the analytical rehearsal. The current task remains entirely read-only and uses fictional local records.

E15-X1 Rehearse adding one reference key without changing the database

Use a CTE that selects location_code from locations and UNIONs the literal L90. Recompute movement exceptions against that proposed set. Return the exact remaining detail, name the two removed IDs and reconcile the new count and net units. Do not claim any real correction occurred.

Submit: Runnable read-only SQL, eight detail rows and a bounded comparison note.

Assessment

Choose one best answer per question. Questions 1–10 check the foundation concepts; questions 11–20 apply the extended study units. Complete and submit the practical tasks separately. A strong quiz score does not replace correct practical work.

1. A catalog key is explicitly NOT NULL, while its label may be NULL. Which post-LEFT-JOIN test reliably detects no matching catalog row?

A. catalog.label IS NULL

B. receipt.product_code IS NULL

C. catalog.product_code IS NULL

D. receipt.receipt_id IS NULL

2. The report must list every receipt lacking a catalog match. Why preserve receipts as the left side of a LEFT JOIN?

A. To return only catalog entries never used by receipts.

B. To merge repeated receipt product codes into one row.

C. To retain receipts even when the reference attachment fails.

D. To discard receipts with incomplete source keys before matching.

3. RC02 and RC11 match catalog P02, whose label is NULL. How should a missing-reference report treat these receipts?

A. Include both because label completeness is part of key membership.

B. Exclude both because the catalog key matches.

C. Include RC02 only as a representative exception.

D. Exclude only the receipt with fewer units.

4. Two receipts contain the same unknown code and have different receipt IDs. The required detail grain is one receipt. How many detail rows belong in the exception list?

A. Two, one per affected receipt.

B. One, after grouping by the unknown code.

C. One, selecting the receipt with the largest units.

D. Zero, because repetition makes the code a known value.

5. What must the correlated NOT EXISTS subquery test for each receipt?

A. Whether the catalog table contains any rows at all.

B. Whether any catalog row has product_code equal to this receipt code.

C. Whether any catalog label is non-NULL.

D. Whether another receipt repeats this product code.

6. The source has NULL and empty product codes, and neither appears in the catalog. Why keep their original representations separate?

A. To prove they refer to different physical products.

B. To preserve distinct source states without inventing their meanings.

C. To force SQL to sort them into different product families.

D. To classify empty text as matched because it is not NULL.

7. Which check verifies that a grouped exception summary preserves the selected detail population?

A. Compare the number of groups with the total catalog rows.

B. Compare maximum units with maximum receipt ID.

C. Count nonmissing labels in the reference.

D. Sum group receipt counts and compare with exception detail rows.

8. Why does filtering NULL out of the right-hand NOT IN subquery not necessarily satisfy a contract that includes NULL source keys as unmatched?

A. A NULL source key still does not make NOT IN true against a nonempty set.

B. It duplicates all nonmissing source rows.

C. It forces the right-hand set to be empty.

D. It changes the source keys into empty text.

9. A grouped report uses MIN(receipt_id) but supplies no timestamps. What does this value establish?

A. The earliest receipt in time.

B. A deterministic example under the identifier ordering.

C. The receipt responsible for the unknown code.

D. The only receipt affected by the group.

10. The authorized task is diagnostic only. What follows after verifying missing-reference rows?

A. Remove unmatched receipts from the source snapshot.

B. Insert placeholder catalog rows to make the check pass.

C. Change unknown codes to the nearest spelling.

D. Prepare the exceptions and limitations for owner review.

11. A candidate swaps unmatched zero-delta M17 for matched zero-delta M10. Count and net units remain unchanged. Which evidence rejects it?

A. Count of columns in the output.

B. A repeated execution of the same candidate query.

C. An exact identifier comparison finding missing M17 and extra M10.

D. A net-unit total alone.

12. The ten main exceptions form six raw-key groups, including NULL. What does COUNT(DISTINCT location_code) return on those exceptions?

A. 10

B. 6

C. 5

D. 0

13. M09 uses L04, an existing inactive location. Which finding is correct?

A. It is matched under all-location membership but lacks an active-location match.

B. It proves the location was inactive at the movement time.

C. It is matched under both all-location and active-only tests.

D. It is missing from all locations because inactive means absent.

14. A NULL candidate is compared with a deliberately empty subquery in SQLite using NOT IN. What is the predicate result?

A. True.

B. False.

C. NULL.

D. An automatic type error.

15. A source has one unmatched movement with delta zero. Which aggregate pair describes it?

A. COUNT(*)=0 and SUM(delta)=NULL.

B. COUNT(*)=1 and SUM(delta)=NULL.

C. COUNT(*)=1 and SUM(delta)=0.

D. COUNT(*)=0 and SUM(delta)=0.

16. N3 has location L90 and acknowledgment K3. The report asks for notices without acknowledgments. How should N3 be treated?

A. Exclude it because the notice-response relationship exists.

B. Include it unless K3 has two comments.

C. Include it because L90 is unknown.

D. Replace L90 before deciding.

17. Two anti-match queries return ten IDs each. What additional comparison establishes equality of their identifier sets?

A. Compare the longest ID strings.

B. Compare total unit deltas only.

C. Check that both use the same table aliases.

D. Check both directional set differences are empty.

18. Which statement about M13 containing a leading space before L01 follows from the exact-match contract?

A. It is a NULL input because the first character is blank.

B. It is automatically L01 because leading spaces have no meaning.

C. Its source value should be overwritten before reporting.

D. It remains unmatched unless the matching policy is explicitly changed.

19. A joined query uses WHERE l.active=1 AND l.location_code IS NULL. Why does it lose genuine unmatched movements?

A. The reference key must always equal the movement ID.

B. The NULL-extended active value does not satisfy active=1.

C. All unmatched movements have negative units.

D. SQLite replaces missing active values with one.

20. The notice snapshot contains no response for N2 but supplies no times or deadlines. Which conclusion is supported?

A. N2 is overdue.

B. The recipient ignored N2.

C. N2 should be cancelled immediately.

D. No matching acknowledgment for N2 is recorded in this snapshot.

Sources and further reading

SQLite SELECT documentation
https://www.sqlite.org/lang_select.html

SQLite SQL language expressions
https://www.sqlite.org/lang_expr.html

SQLite CREATE TABLE and constraints
https://www.sqlite.org/lang_createtable.html

SQLite aggregates
https://www.sqlite.org/lang_aggfunc.html

SQLite URI filenames
https://www.sqlite.org/uri.html