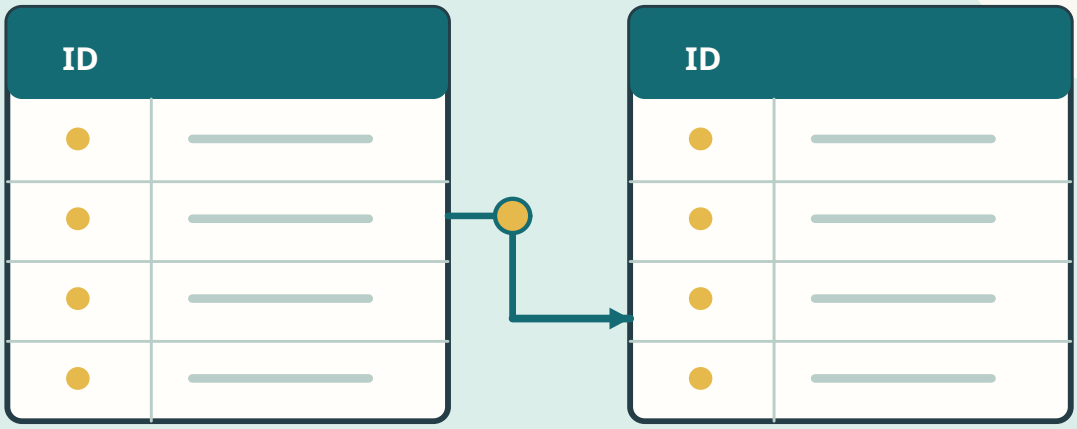
VERIFIED TABLE JOINS

Joining Two Tables with Verified Keys



KEYS / JOIN

Joining Two Tables with Verified Keys

Learning outcomes

- Verify key uniqueness and predict join cardinality before joining.
- Join two related tables using stable identifiers rather than repeated names.
- Reconcile joined row count and monetary totals to the order source.
- Define an event-level join contract, its population, output fields, and independent controls.
- Profile reference keys for duplicates, missing values, empty text, and normalization collisions.
- Predict output cardinality and signed-amount effects from per-event match multiplicities.
- Distinguish INNER and LEFT JOIN results, source-key failures, and missing matched attributes.
- Explain how ON conditions, WHERE filters, projection, DISTINCT, and aggregates change report meaning.
- Use a two-field scoped key without multiplying or misassigning customer relationships.
- Reconcile all-order, matched, unmatched, customer-grain, and current-region populations with exact evidence.
- Deliver a read-only, reproducible capstone with explicit limitations and independently verified controls.

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

Reason about one-to-many relationships

A join connects related records; it does not simply paste two tables side by side. In this fixture, the customers table has one row per customer identifier and the orders table has one row per order. One customer may have many orders. Therefore each order should match exactly one customer row, while several output rows may legitimately repeat the same customer information. Repeated region values after the join are expected and do not imply duplicated orders.

Before writing the join, identify the stable relationship: orders.customer_id refers to customers.customer_id. The similarly named customer_name_snapshot is not the key. Two separate customers share Harbor Co as a display name, and a name can also change over time. A readable description is useful for people but not automatically unique enough for a join.

Predict the output grain and count. If every order has a matching customer and the customer key is unique, enriching orders should preserve one row per order. The expected count is the

order count, not the customer count and not the product of table sizes. Write this prediction before executing the report. It provides a falsifiable expectation that can reveal both missing matches and multiplied matches rather than accepting the first plausible-looking result.

Verify the key instead of trusting its label

Inspect the schema for key declarations, then test the actual key values. A duplicate-key diagnostic groups the customer table by customer_id and reports groups whose count exceeds one. For this exercise it should return no exceptions. Also inspect whether the key can be NULL. A declared key and a descriptive column name are useful clues, but your validation should state what was checked and what the dataset actually contains.

Next check coverage from the order side. Even a unique customer table can fail to contain a customer referenced by an order. An inner join would drop such an order. A read-only unmatched-key check provides evidence that the planned inner join will preserve every order in this fixture. A production database might enforce this relationship with constraints, but the analyst still needs to know the reporting consequences of unmatched input.

Calculate the source amount total before joining. Use integer cents consistently and do not round or convert units during the validation. This creates a second invariant beyond row count. A join could contain the right number of rows yet match them to the wrong customers, so inspect sample identifier pairs and region values as well. Validation is strongest when structure, totals and example relationships support the same conclusion.

Write an explicit, readable relationship

Use short table aliases such as o for orders and c for customers, then qualify columns in the SELECT and ON clauses. The explicit condition o.customer_id = c.customer_id documents the relationship and avoids ambiguity when both tables contain a column with the same name. Select only the fields required by the output contract. This keeps the report at order grain and makes its customer enrichment visible.

Add SQL comments explaining the join key and predicted cardinality. Comments should capture the reason for the key choice, not merely repeat the syntax. For example, note that the customer identifier is unique on the reference side and that every order is expected to match once. Put the expected source total in the comment after you have calculated it independently. Reviewers can then compare intention with the executed controls.

Do not add DISTINCT as an automatic repair for unexpectedly multiplied rows. It can hide some duplicates, fail to remove others and collapse legitimate source records with identical selected values. When the output grows unexpectedly, inspect the matching relationship. The deliberately wrong name-based join is part of this exercise so you can observe how a seemingly reasonable condition can multiply orders associated with a shared name.

Reconcile and explain the rejected alternative

Run controls on the correct joined result: count its rows, count distinct order identifiers and sum amount_cents. Compare these with the source order controls. Then inspect the customers that share a display name and confirm that their orders carry the intended regions. This relationship-level review detects a mistake that totals alone might not reveal, especially if two customers happen to have equal order amounts.

For the name-based alternative, retain only a diagnostic comparison, not a business report. Explain why each Harbor Co order can match two customer rows and therefore contributes its amount twice. Quantify both the row-count inflation and the amount inflation from the actual data. The lesson is not that inner joins are unsafe; it is that the key and relationship cardinality determine what a join means.

Submit the correct query with its comments, the source and joined controls, the duplicate-key and unmatched-key checks, and a short rejection note for the name join. A reviewer should be able to recreate every number and understand why the accepted join is many orders to one customer. Preserve the original tables. The objective is to enrich information while maintaining the intended analytical unit, not to edit the data until a preferred report appears.

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: One reference row can enrich several facts

Sql

```
WITH items(item_id,label) AS (VALUES ('A','Demo Item')), uses(use_id,item_id) AS
    (VALUES ('U1','A'),('U2','A')) SELECT u.use_id,i.label FROM uses AS u JOIN
    items AS i ON u.item_id=i.item_id ORDER BY u.use_id;
```

Explanation

Two use records remain two result rows. Repeating the label is legitimate because the output grain is a use record, not an item.

Foundation practice

Scenario

A fictional supply cooperative wants order details enriched with customer region. Two different customers share the display name Harbor Co, so the display name is deliberately not a safe join key. Order amounts are integer cents.

Database file: 3H-AI-0146_practice/practice.sqlite

Schema file: 3H-AI-0146_practice/schema.sql

Runner file: 3H-AI-0146_practice/run_query.py

Readme file: 3H-AI-0146_practice/README.txt

Tables

customers

customer id	display name	region
C01	Harbor Co	North
C02	Maple Studio	West
C03	Harbor Co	South
C04	Peak Lab	East
C05	Cedar Team	North

Csv file: customers.csv

orders

order id	customer id	customer name snapshot	amount cents
O01	C01	Harbor Co	1200
O02	C01	Harbor Co	1800
O03	C02	Maple Studio	900
O04	C03	Harbor Co	2100
O05	C03	Harbor Co	700
O06	C04	Peak Lab	3000
O07	C05	Cedar Team	1100
O08	C02	Maple Studio	500
O09	C01	Harbor Co	1600

Csv file: orders.csv

Requirements

Create a two-table INNER JOIN that returns order_id, customer_id, display_name, region and amount_cents ordered by order_id. Comment the chosen key, expected nine rows and the source amount total that you calculate independently. Verify the customer key is unique and no order key is unmatched. Contrast with a deliberately incorrect join on customer_name_snapshot = display_name, reporting its row count and total without using it as the report.

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: State both table grains, test customer key uniqueness and calculate the source order count and total.
- Exercise 2: Check order-to-customer coverage and predict the joined row count before running the report.
- Capstone: Deliver the commented identifier-based join and controls, then quantify and explain the incorrect name-join result.

01 Define the report before choosing a join

The expanded workshop concerns the fictional Juniper Supply Circle. Use assets/workshop.sqlite, not the original nine-order database. Complete input records are in assets/customers.csv, assets/orders.csv, and assets/data_dictionary.json. The excerpts below highlight identity and missing-data cases; the report must include every supplied order, not only these excerpts.

A report contract defines the row's meaning, included population, required fields, and correctness evidence. Each order_id identifies a recorded order event. Zero does not remove an event; a negative amount represents the supplied refund. Each customer_id identifies current customer information, not historical attributes at the order date.

The business identifiers connect these two grains. C11 can appear in several orders without violating the order grain because J01, J02, and J09 are different order events. Conversely, two customer records sharing a display name do not become one customer. C11 and C13 both use Apex Supply, but the case explicitly treats them as separate organizations. The report must not infer customer identity from spelling or from the region the analyst expects to see.

For the all-order report, retain the source customer_id even when the reference lookup fails. Select the reference display_name and region as enrichment, and make lookup status visible. Keeping the source identifier lets a reviewer investigate C99 instead of receiving only blank customer fields. The output contract is order_id, source_customer_id, display_name, region, amount_cents, and match_state, sorted by order_id. Later units establish why the correct preservation choice is LEFT JOIN rather than assuming a join type now.

The current active flag does not authorize discarding orders. Unless an exercise changes the population, include every status and inactive customer. Each named fixture is a separate two-table case.

customer id	display name	region	active
C11	Apex Supply	North	1
C13	Apex Supply	South	0
C16	Fern Works	NULL	1

order id	customer id	snapshot	cents	status
J04	C13	Apex Supply	2100	posted
J05	C13	Apex Supply	-300	refund
J10	C99	Ginkgo Shop	700	posted
J11	NULL	Unrecorded	400	posted
J12	C16	Fern Works	0	posted

E01-X1 Write a falsifiable output contract

Using the complete customers and orders tables in assets/workshop.sqlite or their matching CSV files, state both source grains, the required report grain, the all-order population rule, the output fields, and two examples that distinguish customer identity from a display name. Identify which source identifiers must remain inspectable when a lookup fails.

Submit: A concise contract with the exact six output fields, the two named identifier examples, and the source-preservation rule. No SQL is required yet.

02 Establish independent source controls

Calculate source controls directly from orders before adding customer attributes. Record row count, distinct order IDs, populated customer IDs, and signed cents. These measurements test different properties; a joined total cannot serve as its own independent baseline.

The twelve input records contain eleven populated customer identifiers, not eleven different customers. COUNT(customer_id) counts populated cells, whereas COUNT(DISTINCT customer_id) counts different populated identifier values. C11 contributes three populated cells but only one different identifier. NULL is excluded by these expression counts. COUNT(*) counts all order rows, including J11, whose relationship identifier is missing. When documenting a result, name the measure rather than writing an unlabeled number such as twelve.

A population rule also determines which money is being reconciled. The all-status total includes J08's 500-cent draft and J05's negative 300-cent refund. An analyst who excludes drafts has deliberately changed the source population, not repaired the join. An analyst who sums only positive amounts has changed the measure. Neither change is inherently forbidden in a differently specified report, but neither belongs in the all-order contract. Recalculate a direct source control whenever the contract changes.

The worked query below profiles the draft subset, whose single record contributes 500 cents. It is deliberately narrower than the exercise. The status label is a supplied case fact: posted, draft, and refund are exhaustive in this fixture. They do not encode a universal accounting policy. This course asks you to preserve and explain the given signed amounts, not to decide revenue recognition.

Retain integer cents through validation and label their units. Save each control's SQL, headers, population filter, and dataset name. Grouped outputs need declared ordering for exact comparison; a one-row aggregate does not.

```
SELECT COUNT(*) AS draft_rows, SUM(amount_cents) AS draft_cents
FROM orders WHERE status = 'draft';
```

draft rows	draft cents
1	500

E02-X1 Build the all-order control sheet

For orders, compute COUNT(*), COUNT(DISTINCT order_id), COUNT(customer_id), COUNT(DISTINCT customer_id), and SUM(amount_cents). Then group by status and return status, row count, and signed cents in alphabetical status order. Explain why the two customer-identifier counts differ.

Submit: Two saved SELECT statements, both exact result tables, and a two-sentence interpretation of the customer-identifier counts.

03 Profile a proposed reference before trusting it

The alternative customer_incoming export is fully supplied in assets/customer_incoming.csv and workshop.sqlite. Its unique entry_id identifies an export row, not a customer. The diagnostic excerpt below includes conflicting records, repeated information, NULL, and empty text. Keep customers as the approved reference.

Profile uniqueness using the proposed relationship key, customer_id, rather than the export-row key. Every entry_id is unique, but that does not prevent two export rows from describing C12. Grouping by customer_id exposes the repeated relationship values. A duplicate group is an exception to investigate, not a command to delete rows. I02 and I08 disagree about region, whereas I09 and I10 repeat the same descriptive information. Both cases multiply matching pairs if used directly as a reference.

Separate completeness from uniqueness. A NULL identifier is missing; an empty string is a stored value with zero characters. An export can have no repeated nonempty identifiers yet still contain unusable blanks. COUNT(DISTINCT customer_id) excludes NULL but includes the empty string, so it cannot alone establish that all reference records have valid identifiers. Report null and empty-key rows explicitly instead of silently folding them into an invented Unknown customer.

The provided query identifies repeated customer identifiers and returns their number of export rows. It intentionally retains the grouping key so that an investigator can locate the contributing records. To inspect conflicting attributes, return the original rows for the reported identifiers, including entry_id and region. Selecting a convenient MAX(region) would choose a label without evidence that it is authoritative. A deterministic choice can still be an incorrect business decision.

Reject this export as a direct unique-customer reference. The supplied case authorizes using customers instead; it does not authorize choosing an arbitrary region or inventing a deduplication policy.

entry id	customer id	display name	region	active
I02	C12	Birch Studio	West	1
I08	C12	Birch Studio	Central	1
I09	C18	Holly Group	North	1
I10	C18	Holly Group	North	1
I11	NULL	Missing Code	East	1
I12	"" (empty text)	Empty Code	West	1

```
SELECT customer_id, COUNT(*) AS copies
FROM customer_incoming
GROUP BY customer_id HAVING COUNT(*) > 1
ORDER BY customer_id;
```

E03-X1 Triage the reference export

Run the duplicate-key check and a separate query returning entry_id and customer_id for NULL or empty customer identifiers, ordered by entry_id. Classify each duplicate group as conflicting attributes or repeated information. State whether entry_id would be a valid substitute for orders.customer_id.

Submit: Two exception tables, duplicate classifications, and a specific accept/reject decision for this export as a direct customer reference.

04 Distinguish a missing key from a missing attribute

A NULL customer_id does not identify an anonymous customer. Ordinary equality involving NULL is not true, including NULL = NULL. Thus a normal key join does not match missing identifiers to each other merely because both are missing.

Use IS NULL to test missingness. The expression customer_id = NULL is not a reliable missing-value filter and does not select the missing-key order. Empty text is different: it is a text value that can compare equal to another empty text value. If both source and reference contain empty identifiers, an equality join can create a meaningless-looking but syntactically valid match. Validation must reject or separately handle such identifiers according to the supplied case rules, not rely on NULL behavior to catch them.

Missing descriptive information is another condition entirely. C16 is an identified customer with a NULL region. J12 therefore has a successful customer lookup and an incomplete attribute. J11 has a missing source identifier, while J10 has a populated identifier absent from the customer table. These three records can all show a blank region in an export, yet they need different explanations. Testing region IS NULL cannot distinguish them.

The safe match marker in this fixture is c.customer_id because the schema explicitly declares it NOT NULL on real customer rows. After a LEFT JOIN, a NULL in that right-side field indicates no matched customer. The marker is safe because of its declared and checked property, not because every field named id is automatically non-null. In other schemas, choose a non-null row identifier appropriate to the joined reference.

The expression query demonstrates comparison behavior, not an identity policy. Do not introduce null-safe matching to shorten an exception list: several missing keys can belong to different unknown entities. Matching them would invent a relationship.

```
SELECT NULL = NULL AS ordinary_equality,
       NULL IS NULL AS missing_test,
       '' = '' AS empty_equality;
```

ordinary equality	missing test	empty equality
NULL	1	1

E04-X1 Label three distinct conditions

Using orders and customers, classify J10, J11, and J12 as absent reference key, missing source key, or matched customer with missing region. Write a query listing orders whose source customer_id IS NULL. Explain why WHERE c.region IS NULL would be too broad for an unmatched-order report.

Submit: Three order-level classifications, the exact missing-source-key result, and an explanation naming the matched order that a region-null test would wrongly include.

05 Predict row multiplication from match counts

For each source order, count m reference matches. INNER JOIN contributes m rows; LEFT JOIN contributes m when positive, otherwise one placeholder. Sum those contributions to predict result size. LEFT JOIN preserves unmatched events but does not prevent multiplication.

With the approved customers reference, J01 has one match, J10 has none, and J11 has none. With customer_incoming, J03 and J08 each have two matches because C12 appears twice. These extra matches do not create new orders. They create extra output representations of existing orders, and each representation carries the source amount. The raw export therefore threatens both event counts and the money attributed to regions.

For a per-order diagnostic, begin with orders and LEFT JOIN the proposed reference. Group by the unique order_id and count a reference field that is non-null on every real reference row. customer_incoming.entry_id is appropriate here. COUNT(*) is not the correct match count: an unmatched order still receives a LEFT JOIN placeholder row, so COUNT(*) would report one instead of zero. Retain the diagnostic order even when its match count is zero.

The displayed query inspects the three orders J03, J08, and J10. Their counts are two, two, and zero. The first two create four inner-result rows together; the last contributes none. In a left result they create five rows together, because J10 is retained once. This arithmetic does not depend on the names or regions selected for presentation. Hiding fields cannot alter which reference rows matched.

Money repeats with matches too: J03 contributes 900 cents twice under the defective reference. SQL does not divide it between customers. That repetition violates this course's one-current-reference-row enrichment contract; it is not an authorized allocation.

```
SELECT o.order_id, COUNT(c.entry_id) AS matches
FROM orders AS o
LEFT JOIN customer_incoming AS c
  ON o.customer_id = c.customer_id
WHERE o.order_id IN ('J03','J08','J10')
GROUP BY o.order_id ORDER BY o.order_id;
```

order id	matches
J03	2
J08	2

order id	matches
J10	0

E05-X1 Predict the defective-reference result

Compute match counts for all twelve orders against customer_incoming. Before running a report total, predict the INNER JOIN row count and cents and the LEFT JOIN row count and cents. Identify every order responsible for excess rows, and distinguish those from orders with no match.

Submit: The twelve-row multiplicity table, four predicted controls, and a short contribution calculation for each multiplied order.

06 Write an explicit identifier-based INNER JOIN

Separate the input tables, matching condition, and selected fields. With aliases o and c, o.customer_id = c.customer_id compares the intended domains. By contrast, o.customer_id = o.customer_id never tests a relationship with the reference table.

Select fields deliberately. Including both source_customer_id and matched_customer_id can be helpful during diagnosis, but the published field contract may require only the source identifier. Give exported columns stable, meaningful labels. Avoid SELECT * for a reviewed report because it exposes duplicate column names, unnecessary fields, and whatever columns happen to be added later. Explicit selection is a form of specification, not merely a formatting preference.

The worked query below focuses on J01 and J04. Both order snapshots say Apex Supply, but the identifier relationship assigns North to J01 and South to J04. That small pair inspection establishes something a grand total cannot: these orders are associated with different customer entities. The evidence is the recorded identifier and the approved customer reference, not the analyst's familiarity with the display name.

For the entire workshop, an INNER JOIN is suitable for a report explicitly limited to orders with identified reference matches. It is not sufficient for the all-order contract from E01. Ten orders have exactly one approved customer match, while two have none. A valid SQL statement can therefore be the wrong business report. Label an inner result as matched orders; do not quietly present it as all orders because its selected fields look complete.

Use ORDER BY order_id for reproducible comparison. Sorting does not prove uniqueness, and aliases do not enforce constraints. Save the full query with comments identifying the population, verified key, expected matched-order count, and source cents.

```
SELECT o.order_id, o.customer_id AS source_customer_id,
       c.display_name, c.region, o.amount_cents
FROM orders AS o
INNER JOIN customers AS c ON o.customer_id = c.customer_id
WHERE o.order_id IN ('J01','J04')
ORDER BY o.order_id;
```

order id	source customer id	display name	region	amount cents
J01	C11	Apex Supply	North	1200
J04	C13	Apex Supply	South	2100

E06-X1 Produce and correctly label the matched-order report

Write the complete five-column INNER JOIN without the worked example filter. Return order_id, source_customer_id, display_name, region, and amount_cents in that order, sorted by order_id. Calculate rows, distinct order identifiers, and cents. Explain whether this fulfills the all-order contract.

Submit: Commented SQL, the ten exact report rows, three controls, and a population statement naming the omitted orders.

07 Preserve unmatched facts without inventing customers

Put the population to preserve on the left. orders LEFT JOIN customers retains unmatched events; customers LEFT JOIN orders instead preserves customers, including those without orders. The approved reference's uniqueness is what prevents event multiplication.

When the reference lookup fails, SQLite supplies NULLs for the customer fields, not a new customer entity. The source-side fields remain available, including J10's C99 and 700 cents. Select o.customer_id rather than only c.customer_id if the output must retain the unmatched identifier. A report that returns a blank customer ID for J10 loses diagnostic information unnecessarily, even though the order row itself survives.

Build an explicit match_state with a CASE expression. First test the source identifier for NULL, then the non-nullable customer marker for a failed lookup, and otherwise call the row matched. This ordering distinguishes missing_source_key from absent_reference_key. Region completeness is a separate attribute check. J12 is matched even though its region is NULL and its amount is zero. Zero describes the amount, not the existence of a reference row.

The worked exception query below returns only the two unmatched events using c.customer_id IS NULL. It intentionally includes both types of failure. For an investigation list, keep order_id, source_customer_id, and amount_cents so an owner can connect the exception to a concrete input record and its contribution. Do not replace the source identifier with the display name or remove exceptions merely to make regional totals look complete.

COALESCE(c.region,'Unassigned') merges unmatched customers with matched customers lacking region. That may be an explicit presentation bucket, but it is insufficient relationship evidence. Retain the underlying match state in the detail or a companion control.

```
SELECT o.order_id, o.customer_id AS source_customer_id,
       o.amount_cents
FROM orders AS o LEFT JOIN customers AS c
  ON o.customer_id = c.customer_id
WHERE c.customer_id IS NULL ORDER BY o.order_id;
```

order id	source customer id	amount cents
J10	C99	700
J11	NULL	400

E07-X1 Complete the six-field all-order report

Implement the E01 contract with a LEFT JOIN and CASE labels matched, absent_reference_key, and missing_source_key. Use the source identifier, keep NULL region values unchanged, include every order, and sort by order_id. Supply exact rows and the row-count and cents controls.

Submit: A six-column report, saved query, and controls demonstrating that all twelve events remain.

08 Reconcile matched and unmatched populations

Explain population differences as a bridge. Ten matched events plus two unmatched events reconstruct twelve source events; 11900 matched cents plus 1100 exception cents reconstruct 13000. This shows exactly what the INNER JOIN omits.

The bridge works because the approved reference has no duplicate customer identifiers. If the reference multiplied some orders, a joined matched total would already include repeated facts and would not be a clean partition of the source. Establish uniqueness and per-order multiplicity before interpreting matched plus unmatched as a complete decomposition. A formally neat reconciliation built on duplicated input can give a false sense of assurance.

Use the same population filter in every side of the comparison. For example, if the report excludes drafts, the independent source control must also exclude drafts. Comparing a non-draft joined total with an all-status source total creates a discrepancy unrelated to the key. Place the fact-population rule visibly in each query, or save a clearly named filtered source query and verify it separately. Consistency of scope is more important than shortening the SQL.

The worked table below splits the original all-status population by match_state. It includes both reason and monetary contribution, so an absent reference key is not confused with a missing source identifier. The numbers are diagnostic evidence, not a ranking of which records matter most. J12 contributes zero cents but still belongs to the matched event count and must remain inspectable.

Locate omitted or repeated identifiers before rounding or changing filters. Trace differences to their actual contributions, such as J08's 500 cents. Internal reconciliation cannot prove that an external source originally captured every real-world event.

match state	order rows	amount cents
absent_reference_key	1	700

match state	order rows	amount cents
matched	10	11900
missing_source_key	1	400

E08-X1 Reconcile a deliberately changed population

For this exercise only, exclude orders with status = draft. Compute the direct source count and cents, the matched count and cents, and the unmatched count and cents using the approved customer reference. List the excluded draft identifier and show the two arithmetic bridges back to the filtered source.

Submit: Three exact control outputs, the excluded identifier, and count/cents reconciliation equations written in words or numerals.

09 Put conditions where their meaning belongs

Request A retains all orders but shows only active-customer enrichment. Request B returns only orders belonging to active customers. A restricts matching; B restricts the event population. Decide which request applies before placing c.active = 1.

For A, add c.active = 1 to the LEFT JOIN's ON condition. An order linked to an inactive customer has no eligible enrichment match but remains as a left-side event. J04 and J05 therefore retain their amounts while customer fields become NULL in that particular query. Their source customer identifier remains C13. The resulting blank reference marker means no eligible match under the entire ON condition, not necessarily no customer exists anywhere in the reference table.

For B, join by identifier and use WHERE c.active = 1. An unmatched row has no true active flag, so it does not survive that condition. The two C13 orders also fail because the matched customer is inactive. This query deliberately returns a smaller population. Calling it a LEFT JOIN does not undo the later filter's effect. Read the complete statement, not only the join keyword, when deciding whether unmatched facts survive.

The worked example below demonstrates A for J04 and J06. J04 remains with NULL enrichment; J06 carries C14 and East. An analyst who labels both kinds of NULL reference marker absent_reference_key would now misdescribe J04. If a report needs to distinguish inactive from genuinely absent customers, join to all customers first and inspect the active attribute rather than hiding inactive customers inside ON.

An order-side WHERE o.status = 'posted' intentionally selects events regardless of lookup success. A customer-side filter may remove NULL-extended rows. Review the complete population and matching rule, not merely the LEFT JOIN keyword.

```
SELECT o.order_id, c.customer_id, c.region
FROM orders AS o LEFT JOIN customers AS c
 ON o.customer_id = c.customer_id AND c.active = 1
WHERE o.order_id IN ('J04','J06')
ORDER BY o.order_id;
```

order id	customer id	region
J04	NULL	NULL
J06	C14	East

E09-X1 Compare two complete active-customer interpretations

Run A with the active condition in ON and B with the active condition in WHERE across all orders. For each, return row count, COUNT(c.customer_id), and SUM(o.amount_cents). List the orders absent from B. Explain why A cannot use a NULL reference marker alone to mean that the customer identifier does not exist.

Submit: Two commented queries, their three controls, four omitted identifiers for B, and a precise interpretation of A.

10 Measure why a name-based join fails

Four snapshots use the shared name Apex Supply: J01, J02, J04, J05. Joining that name to both C11 and C13 gives each event one correct identifier pair and one unsupported cross-customer pair.

A name join can also lose valid relationships. J09 records Old Apex, while its stable customer identifier is C11 and the current display name is Apex Supply. Its identifier-based lookup succeeds, but its name-based lookup fails. This demonstrates two independent risks in the same query: repeated names multiply some orders, while changed names omit another. Looking only for duplicated output will miss the omission; looking only for blanks will miss the multiplication.

Calculate the effects using actual contributions. The four shared-name events total 4800 cents because the refund contributes negative 300. Their extra matches add another 4800 cents. Losing J09 removes 1600 cents from the approved matched subset. The net difference between the two inner totals is therefore 3200 cents, not simply the sum of all positive Apex amounts. Signed measures require the same careful multiplicity reasoning as positive-only measures.

The worked pair list below shows the unsupported associations for J01 and J04. Both retain plausible region names, and a reader might not notice the mistake without the matched customer identifier. During review, show both o.customer_id and c.customer_id even when the final business output needs only one. An equality of aggregate totals would not prove these pairs correct; a relationship-level check must examine which entities were actually connected.

Do not rescue the name join by OR-ing it with identifier equality: the weak condition can still create false pairs. Choosing the first region is equally unsupported. Keep the name query solely as a quantified rejected diagnostic.

order id	source customer id	name matched customer id	region
J01	C11	C11	North
J01	C11	C13	South
J04	C13	C11	North
J04	C13	C13	South

E10-X1 Audit the complete name-join alternative

Join orders.customer_name_snapshot to customers.display_name using INNER JOIN. Return total rows, distinct order IDs, and cents. Separately list order_id, source_customer_id, matched_customer_id, and region where the name match connects different identifiers. Identify the valid identifier-linked order lost by name matching and reconcile the net amount difference.

Submit: Diagnostic SQL only, exact controls, all four wrong-identity pairs, and the omitted-order contribution. Do not submit this as the accepted business report.

11 Keep identity when selecting fields and deduplicating

R01 and R02 are separate events with the same customer and 500-cent amount. Their distinct order identifiers establish two events, despite visually identical descriptive values. The complete three-record repeat_orders fixture is shown below.

Join repeat_orders to the approved customers table by customer_id. With order_id retained, the result clearly contains three events and 1800 cents. If the analyst selects only customer_id, region, and amount_cents, the first two result rows look identical. Adding DISTINCT to that reduced projection removes one of them because DISTINCT compares selected values, not the source events the analyst meant to preserve. The result then represents different combinations of attributes, not one row per order.

This is why DISTINCT is not a general repair for duplicate joins. It can hide some repeated matches when selected values happen to agree, fail to hide others when reference attributes differ, and remove legitimate events whose visible attributes coincide. Its effect depends on the SELECT list. The proper question is whether the report's grain calls for unique attribute combinations. If the contract calls for order events, preserve the event identifier and fix the relationship instead.

SUM(DISTINCT amount_cents) introduces a related mistake. It sums different amount values, not different orders. In this fixture the distinct amounts are 500 and 800, so that expression returns 1300 even though two valid events carry 500 each. COUNT(DISTINCT order_id) is useful as a control precisely because order_id defines the intended event identity. The word DISTINCT is neither intrinsically safe nor intrinsically unsafe; its argument and purpose determine its meaning.

Retain order_id in audit evidence. Repeated regions are expected and repeated amounts are possible; a repeated order identifier after intended one-to-one enrichment is the stronger warning. Visible similarity alone does not establish duplicated source events.

order id	customer id	amount cents
R01	C11	500
R02	C11	500
R03	C12	800

E11-X1 Demonstrate the cost of an incorrect DISTINCT

Return the correct joined order_id, customer_id, region, and amount_cents. Then run SELECT DISTINCT on only customer_id, region, and amount_cents. Compare result rows and additive amounts, and calculate SUM(DISTINCT amount_cents) directly from repeat_orders. Explain exactly which real event is no longer separately represented.

Submit: Both exact result tables, their counts and sums, the distinct-amount result, and a grain-based rejection note.

12 Reverse the perspective for a customer summary

For every customer and their order count, put customers on the left and group by customer identity and displayed attributes. This intentionally changes grain to customer and preserves C17, Grove Club, which has no orders.

Count a non-nullable order identifier to measure orders. For C17, the LEFT JOIN supplies one placeholder row with NULL order fields. COUNT(*) counts that row and returns one; COUNT(o.order_id) returns zero. The distinction is not a special exception for Grove Club. It follows from what is being counted. For customers with real orders, both expressions happen to agree, which can let an incorrect query look convincing until the zero-order case is tested.

SUM(o.amount_cents) is NULL for C17 because there are no non-NULL order amounts in its group. A summary may explicitly present that as zero with COALESCE, provided it also reports the order count and describes the convention. C16 also has a displayed total of zero, but for a different reason: it has one genuine zero-value order. Showing both order_count and amount_cents keeps no events distinct from an event worth zero.

The worked table below isolates C16 and C17. Both totals can display zero under the stated presentation rule, while their counts differ. Neither customer's region is needed to establish that difference. Including nullable region in the output is fine, but it must not be used as the marker for order existence. The customer table already contains C16 even though its region is missing.

This customer summary excludes J10 and J11 because neither has a reference customer. Reconcile its 10 orders and 11900 cents to the matched-order population; keep the separate 2-order, 1100-cent exception list instead of assigning invented customers.

customer id	COUNT(*)	COUNT(o.order id)	SUM(o.amount cents)	displayed cents
C16	1	1	0	0
C17	1	0	NULL	0

E12-X1 Summarize all seven customers

Return customer_id, display_name, order_count, and amount_cents for every approved customer. Use zero as the stated presentation value for no order amounts. Sort by customer_id. Reconcile the sum of order_count and the sum of displayed cents to the correct matched-order population, and explain why that differs from all orders.

Submit: The seven exact summary rows, SQL, two summary totals, and a specific note about the two unmatched orders.

13 Aggregate enrichment without losing its meaning

Regional totals allocate order amounts through the verified customer relationship. Label them current-customer-region allocations: this reference has no order-date region history. The data cannot establish where each customer was located when an earlier order occurred.

Group only after establishing the correct order-level relationship. If a defective reference gives C12 both West and Central, the same order amount can enter two regional groups. Each group may look individually reasonable, while the grand total is inflated. Aggregation hides event identity, so preserve the detailed join and reconcile its controls before reducing it into a summary. A grouped report is not a substitute for a row-level validation query.

Define exception buckets explicitly. In this fixture, use the actual region label for a matched customer with a populated region, missing_region for a matched customer whose region is NULL, and unmatched for an order with no customer match. These diagnostic bucket labels do not occur as real region values in the supplied data. If real data could use the same labels, keep a separate category field rather than allowing a presentation label to collide with genuine values.

The matched missing-region bucket contains J12, one event and zero cents. The unmatched bucket contains J10 and J11, two events and 1100 cents. Combining both into a single blank-region bucket would yield three events and 1100 cents, concealing the fact that one customer relationship was successful. Whether a business chart later combines those buckets is a presentation decision; the reconciliation evidence should retain the distinction.

North's 5700 cents across four events averages 1425 cents per order. Dividing by its two represented customers instead gives 2850 cents per customer. Neither is an average across all seven customers. State the denominator, not simply average amount.

```
SELECT COUNT(*) AS north_orders, SUM(o.amount_cents) AS north_cents
FROM orders o JOIN customers c ON o.customer_id=c.customer_id
WHERE c.region='North';
```

north orders	north cents
4	5700

E13-X1 Build a reconciled current-region summary

Use the all-order LEFT JOIN and the three-way bucket definition above. Return region_bucket, order_count, and amount_cents, ordered by region_bucket. Keep matched missing regions distinct from unmatched orders. Reconcile both grouped totals to the all-order source, and explain what the current-region qualifier prevents you from claiming.

Submit: Six exact summary rows, saved SQL, two reconciliation totals, and a one-sentence historical-data limitation.

14 Recognize controls that can pass accidentally

The balanced fixture deliberately offsets two defects. Three source events total 1000 cents. Q1 has two reference entries, Q2 has none, and Q3 has one. An inner join repeats B01 and loses B02, whose equal 500-cent amounts conceal the change.

The apparent agreement is misleading. The result contains B01 twice, B03 once, and no B02. COUNT(DISTINCT order_id) falls to two, while a per-order match diagnostic reports two, zero, and one. Those controls expose facts that the total and row count hide. The North and South assignments for B01 also disagree, so selecting either region without a rule would invent an answer to an unresolved reference problem.

A second case shows cancellation without multiplication. cancel_orders contains Z01 for 250 cents, Z02 for 600 cents, Z03 for negative 600 cents, and Z04 for zero. Z02 and Z03 both reference absent customer C98. Joining to the approved customers table drops these two events, but their signed amounts cancel. The inner total still equals 250 cents. Row count, identifiers, and unmatched contributions reveal the loss even though the money agrees.

The zero event reinforces the distinction between existence and value. Losing Z04 would not change the signed amount either. An analyst who checks only SUM(amount_cents) could therefore miss both canceled amounts and a zero-valued record. Keeping the event identifier in the detailed report provides the unit against which absence and repetition can be tested. Negative and zero values are not nuisances to remove from test data; they are valuable cases for evaluating the strength of a validation method.

Combine count, distinct identity, multiplicity, amount, and sample-pair checks. State the narrow conclusion of each passed check. None proves external source completeness, and a matching total must not overrule a failed relationship check.

entry id	customer id	region
B1	Q1	North
B2	Q1	South

entry id	customer id	region
B3	Q3	East

order id	customer id	amount cents
B01	Q1	500
B02	Q2	500
B03	Q3	0

order id	customer id	amount cents
Z01	C11	250
Z02	C98	600
Z03	C98	-600
Z04	C16	0

E14-X1 Diagnose two false reassurances

For balanced_orders joined to balanced_customers, return joined rows, distinct orders, and cents, then identify repeated and omitted events. For cancel_orders joined to customers, compare direct-source and inner-join controls and list the omitted events with their signed amounts. Explain separately why each money check passes.

Submit: Exact source/join controls for both cases, explicit repeated or missing IDs, and two different cancellation explanations.

15 Treat normalization as a proposed key change

The text-code fixtures contain 007, 7, case variants, and a leading-space value. No supplied rule makes these equivalent. Casting, trimming, or uppercasing changes the proposed relationship; more matches alone do not establish better identity resolution.

SQLite text comparisons depend on their collation. The supplied text columns use the default BINARY collation, so case and the leading space matter for these examples. The customers whose codes are ' A1' and 'a1' are separate reference rows. Applying UPPER(TRIM(customer_code)) maps both to A1. A previously unique reference key is no longer unique under that expression. The transformed key must therefore be profiled independently of the original key.

T03 stores A1 and has no exact reference match. Under the proposed trim-and-uppercase condition, it matches two reference rows. T04 stores the exact leading-space value and already had one valid exact match, but normalization gives it two. The increased match count is not evidence that the transformation has recovered true identities. It has expanded

the candidate relationship and introduced ambiguity. Preserve both original values in any diagnostic output so the collision remains visible.

Leading zeros present a related risk. Casting the two codes 007 and 7 to INTEGER maps both to seven. That may be valid in a domain whose documented identifier rule ignores leading zeros, but this fixture supplies no such rule. The fact that an expression returns a familiar-looking number does not establish business equivalence. Avoid experimenting with broad casts on mixed codes; nonnumeric codes can produce results that are not usable identifiers at all.

Before accepting normalization, establish the authorized equivalence rule, inspect collapsing original values, recheck reference uniqueness, and compare per-order matches. Without that evidence, preserve exact values and report unresolved exceptions rather than manufacture identity.

customer code	region
007	North
7	South
' A1'	East
a1	West

order id	customer code	amount cents
T01	007	100
T02	7	200
T03	A1	300
T04	' A1'	400
T05	NULL	500
T06	"" (empty text)	600

```
SELECT UPPER(TRIM(customer_code)) AS proposed_key, COUNT(*) AS copies
FROM code_customers GROUP BY proposed_key HAVING COUNT(*) > 1;
```

E15-X1 Test a proposed normalized join before accepting it

Compute INNER JOIN row count and cents under exact text equality and under UPPER(TRIM(...)) equality on both sides. List the colliding normalized reference key and its copy count. Identify the two orders whose multiplicities increase, and state whether the supplied evidence authorizes treating the original codes as equivalent.

Submit: Two exact control pairs, the collision result, changed-order explanations, and an accept/reject decision for normalization as a verified identity rule.

16 Verify a key made of two fields

In the branch fixture, local_id is unique within branch, not globally. East/01 and West/01 identify different customers. The supplied relationship key is the pair, expressed by two conditions while still joining only two tables.

Profile the pair with GROUP BY branch, local_id. Testing local_id alone reports repeated 01 values, but that repetition is expected under the stated grain. Testing branch alone also fails because a branch contains multiple customers. The meaningful uniqueness claim concerns the complete pair. A constraint or a duplicate query must match the actual identifier domain that the order table uses.

Write the relationship as o.branch = c.branch AND o.local_id = c.local_id. Both equalities must hold for a match. P01 connects to Ada Shop, P02 to Bela Shop, and P03 to Cato Shop. P04 has local identifier 02 in West; East/02 exists, but it is not the same scoped identity. A local-ID-only join would incorrectly connect P04 to Cato Shop even though the complete key has no reference match.

P05 has a missing local_id. Having a populated branch is not sufficient to identify its customer, so ordinary equality produces no complete-key match. Treat the incomplete key as a separate exception from a populated pair absent from the reference. The same missing-source versus absent-reference distinction used for single-field keys still applies, but completeness must now consider every required component.

Do not concatenate components without a collision-safe representation: ('A','12') and ('A1','2') both become A12. These are illustrative pairs, not extra fixture records. Comparing both fields separately avoids such ambiguity; even delimiters need rules when they can occur inside values.

branch	local id	display name
East	01	Ada Shop
West	01	Bela Shop
East	02	Cato Shop

order id	branch	local id	amount cents
P01	East	01	100
P02	West	01	200
P03	East	02	300
P04	West	02	400
P05	East	NULL	500

E16-X1 Compare complete and incomplete key conditions

Verify pair uniqueness, then calculate INNER JOIN rows and cents using both key fields. Compare with a join on local_id alone. List order IDs unmatched under the complete key, including their branch and local_id. Explain which false match appears for P04 and why P01 and P02 multiply under the incomplete condition.

Submit: Uniqueness evidence, two control pairs, exact unmatched records, and an explanation grounded in scoped identities.

17 Audit the full ON condition

Read the condition as a statement about pairs: this order's customer identifier equals this reference customer's identifier. Inspect every AND and OR for a route that can accept a pair without that identity test.

Without an ON or USING condition, the supplied twelve-order and seven-customer tables produce 84 possible pairs. Each order amount appears seven times, for 91000 cents in total. This Cartesian result is useful here only as a diagnostic example of a missing relationship. It is not an order-enrichment report. A database can execute such a statement successfully; lack of a syntax error does not establish analytical correctness.

Parentheses matter when a condition mixes AND and OR. In SQLite, AND binds more tightly than OR. Consider a request for matched orders whose current customer region is North or West. The intended condition requires identifier equality and then one of those two regions. Omitting parentheses can instead require equality for North but allow every West customer to match every order, regardless of identifier.

The two expressions below expose that difference. Under the defective expression, C12 is a West customer and therefore matches all twelve orders. The four North-linked events also match their proper North customers. This creates extra pairs even for orders with missing or absent customer identifiers. J11 can acquire Birch Studio simply because the right-hand region condition is true. Such a result is syntactically valid and semantically indefensible for the supplied identity rule.

Test contrasting rows. J03 succeeds under both expressions and is a weak diagnostic. J10, whose C99 is absent, must never acquire C12 under the intended relationship. Parentheses express the grouping; exact outputs confirm the executed meaning.

```
-- Intended North-or-West population, with verified identity.
SELECT o.order_id,c.customer_id,c.region,o.amount_cents
FROM orders o JOIN customers c
 ON o.customer_id=c.customer_id
 AND (c.region='North' OR c.region='West')
ORDER BY o.order_id,c.customer_id;
```

```
-- Deliberately defective diagnostic: West bypasses identity.
SELECT o.order_id,c.customer_id,c.region,o.amount_cents
FROM orders o JOIN customers c
 ON o.customer_id=c.customer_id AND c.region='North'
 OR c.region='West'
ORDER BY o.order_id,c.customer_id;
```

E17-X1 Quantify a parenthesis defect

Run both expressions as INNER JOIN conditions over orders and customers, returning rows, distinct order IDs, and cents. List the erroneous matched customer for J10 under the defective expression. Write a corrected condition and explain the Boolean grouping rather than simply saying that parentheses are good style.

Submit: Two exact control triples, the J10 wrong pair, and a corrected condition with its intended logical reading.

18 Package a reproducible, read-only review

A reproducible handoff includes SQL, dataset, population contract, controls, and result headers. A screenshot hides relationship and filtering choices. Save UTF-8 SQL and use the supplied read-only runner; source repair is outside this task.

The tested environment is Python's standard sqlite3 module using SQLite 3.53.1. The queries use ordinary SQLite SELECT syntax, INNER JOIN, LEFT JOIN, CASE, and basic aggregates; no external extension is required. Run SELECT sqlite_version() when recording your own environment rather than assuming it matches the author's engine. Exact result assertions have been executed for the supplied fixture. An untested engine or changed input should be recorded as a new validation context, not silently treated as the same run.

Read schema.sql to understand which fields are explicitly non-nullable. The approved customer identifier and each event identifier have PRIMARY KEY NOT NULL declarations. The deliberately defective incoming table has a different grain, with entry_id as its unique row key. Do not infer a relationship constraint from a similar column name. These standalone workshop tables intentionally permit orphan identifiers so the exercises can expose their reporting consequences.

For a new request, rewrite the population statement before changing the query. The review task below asks for posted orders except those assigned to South, while retaining unmatched orders and matched customers with missing region. A bare WHERE c.region <> 'South' would remove NULL-region rows too. The correct specification needs an explicit allowance for NULL regions, alongside the posted-event filter. This is an intentional population change, not a continuation of the original all-order report under an unchanged label.

Record filename, database, engine version, field order, population, expected count, cents, and exception interpretation. Keep setup.sql for reconstructing a new disposable database only. Do not execute it against the original baseline or any working source.

```
SELECT sqlite_version() AS sqlite_version;
```

E18-X1 Deliver a reproducible changed-population report

Return posted orders except matched customers in South, retaining unmatched orders and matched customers with NULL region. Fields must be order_id, source_customer_id, region, amount_cents, ordered by order_id. Supply the nine exact rows, count/cents controls, and an evidence manifest using your actual SQLite version. Explain the NULL allowance in the filter.

Submit: One saved report query, exact output, source-population description, controls, and a short manifest. Do not change the database.

19 Capstone: build an independently checked enrichment

Use cap_customers and cap_orders only. The contract is one row per non-draft order, including refunds, zero amounts, inactive customers, and unmatched identifiers. Preserve signed cents and source customer_id, enrich current attributes, and label the three established match states.

Complete records are in assets/cap_customers.csv, assets/cap_orders.csv, and workshop.sqlite; the excerpts highlight selected cases only. K1/K2 share Stone Shop; V03 has an old name. K3 lacks region, K4 is inactive, and K5 has no order. V05 has an absent key; V06 has a missing key and negative amount.

Start a new control sheet instead of copying 12 and 13000 from the main workshop. First profile the reference relationship key and verify source order identity. Then apply the non-draft population rule directly to cap_orders. Predict how many matches each retained order should have under the approved key. Only after those steps should you run the report and compare row count, distinct order IDs, and signed cents with the independent source controls.

The required field order is order_id, source_customer_id, display_name, region, amount_cents, match_state, sorted by order_id. Retain NULL attributes as NULL in the detailed report. Supply a separate unmatched list containing the order identifier, source customer identifier, signed amount, and reason. The report's label must say non-draft orders, not all orders, because the case deliberately includes a draft that is outside scope.

Run the wrong name-based INNER JOIN on the same non-draft population. Quantify extra and missing contributions relative to the correct identifier INNER JOIN. Do not compare against all-status controls or repair with DISTINCT. Preserve sufficient evidence to distinguish missing region from missing customer.

customer id	display name	region	active
K1	Stone Shop	North	1
K2	Stone Shop	South	1
K3	Willow Lab	NULL	1

order id	customer id	snapshot	cents	status
V03	K1	Old Stone	800	posted
V05	K9	Unknown	600	posted
V06	NULL	Unrecorded	-200	refund
V08	K2	Stone Shop	300	draft

E19-X1 Submit the capstone evidence packet

Provide reference-key uniqueness evidence; direct non-draft source controls; the exact six-field LEFT JOIN report; joined controls; the unmatched list; and wrong name-join controls plus a contribution explanation. Use only supplied records and retain all seven non-draft events. Include a short acceptance statement naming the remaining data limitations.

Submit: A complete SQL-and-results packet with seven report rows, both exception rows, all controls, and a quantified rejection of name matching.

20 Decide whether the evidence supports acceptance

Acceptance requires a chain of evidence: requested population, unique event identity, suitable reference key, expected per-event matches, correct fields, and reconciled controls. A matching grand total cannot compensate for a broken identity relationship.

Acceptance does not mean pretending that incomplete reference information is complete. The capstone's verified all-in-scope LEFT JOIN can be accepted as a non-draft event report while clearly stating that two events have no customer match and one matched customer lacks a region. A request requiring fully assigned regions would need a different decision: the supplied data cannot satisfy that requirement without additional authoritative information. Preserve the distinction between query correctness and source completeness.

Use change impact to decide what to retest. If a customer display name changes but identifiers and regions do not, the accepted key join should preserve membership, counts, and cents while changing the displayed name. If an additional reference row repeats an existing customer identifier, uniqueness and per-order multiplicity must fail even when its attributes are identical. If a new order is added, the old control totals are no longer the correct baseline; recompute them from the new source before comparing the join.

The review scenarios below are independent thought experiments, each starting from the unchanged capstone fixture and its non-draft contract. Do not apply them cumulatively or modify the supplied database. Scenario A changes K1's display name to Stone Cooperative. Scenario B adds a second K2 reference row in an unconstrained proposed export, identical to K2. Scenario C adds V09, customer K9, amount 0, status posted. These facts are sufficient to predict the consequences without guessing any missing values.

For each scenario, state accept, needs revision, or correct with a documented data limitation, supported by exact identifiers and controls. Reject ambiguous identity instead of choosing an

arbitrary reference row; preserve genuine unmatched events when the contract requires them.

independent scenario	single change
A	K1 display_name becomes Stone Cooperative; all other data unchanged.
B	Proposed unconstrained reference export contains one extra identical K2 row.
C	New order V09: K9, Unknown, 0 cents, posted.

E20-X1 Write three evidence-based review decisions

For each independent scenario, predict non-draft LEFT JOIN rows, distinct order IDs, and cents. Identify the exact report rows or diagnostics affected. State whether the key join can be accepted, must be rejected pending reference resolution, or remains correct with a documented unmatched-data limitation. Do not alter workshop.sqlite.

Submit: Three control triples and review decisions, with specific affected identifiers and the reason each check changes or stays stable.

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. What is the intended grain after enriching the original nine orders with customers?

A. One row for each customer, with all orders compressed into it.

B. One row for each distinct customer-region combination.

C. One row for each original order identifier.

D. One row for each distinct amount and display name.

2. Why is display_name unsafe as the relationship key in the original fixture?

A. C01 and C03 are different customers sharing Harbor Co.

B. The customer table contains fewer rows than the order table.

C. Some customers have more than one legitimate order.

D. The display names are not sorted in the same order as the orders.

3. Which diagnostic tests whether customer_id repeats on the original reference side?

A. SELECT DISTINCT region FROM customers;

B. SELECT customer_id FROM customers ORDER BY customer_id LIMIT 1;

C. SELECT COUNT(*) FROM customers;

D. SELECT customer_id, COUNT(*) FROM customers GROUP BY customer_id HAVING COUNT(*) > 1;

4. If an order has no reference customer match, what does an INNER JOIN do with that order?

A. Retains it with NULL customer attributes.

B. Omits it from the joined result.

C. Retains it only when its amount is nonzero.

D. Matches it to another customer with the same row position.

5. What must hold for an order-enrichment INNER JOIN to preserve one output row per order?

A. Every order must match exactly one reference customer row.

B. Each customer must have exactly one order.

C. The amount must be different for every order.

D. The two inputs must contain equal numbers of rows.

6. Why compare the original source cents with the joined cents?

A. To demonstrate that customer names are unique.

B. To replace the need for counting distinct order identifiers.

C. To prove every order was assigned to the correct region without inspecting pairs.

D. To detect possible missing or multiplied monetary facts alongside other checks.

7. An order appears twice after a join that should enrich each order once. What should be checked first?

A. Whether DISTINCT can remove the repeated visible values.

B. The number and identity of reference rows matching that order.

C. Whether dropping order_id makes the report look less repetitive.

D. Whether the amount can be divided by the observed row count.

8. What does qualifying customer_id with aliases such as o and c accomplish?

A. It prevents duplicate customer identifiers from being stored.

B. It makes the selected field sources and relationship comparison explicit.

C. It automatically preserves unmatched order rows.

D. It converts the display name into a stable identifier.

9. A customer has three different orders and its region appears three times. How should that repetition be interpreted?

A. It proves the reference customer row is duplicated.

B. It means the report must be reduced with DISTINCT region.

C. It is expected in a one-row-per-order result.

D. It is acceptable only when all three amounts differ.

10. Which join comment gives a reviewer the most useful validation context?

A. The chosen key, intended relationship, expected order count, and independently calculated source cents.

B. The report uses short aliases and an alphabetic sort.

C. The output looked similar to last month's report.

D. The query ran successfully and therefore needs no further checks.

11. In the approved all-order LEFT JOIN, what are COUNT(*), COUNT(c.customer_id), and COUNT(c.region), respectively?

A. 12, 12, 12

B. 10, 10, 9

C. 12, 10, 9

D. 12, 9, 10

12. The incoming-reference INNER JOIN returns 12 rows, equal to the source count. Which additional result shows that event preservation nevertheless failed?

A. Its SELECT list includes a region.

B. Its rows can be sorted by order_id.

C. It has only 10 distinct order IDs.

D. Its reference table has unique entry_id values.

13. A LEFT JOIN places c.active = 1 in WHERE rather than ON. In the main workshop, which population does it return?

A. All 12 orders, with inactive enrichment hidden.

B. All 10 orders that have any customer match.

C. Only the 2 orders whose customer reference is absent.

D. Only the 8 orders linked to currently active customers.

14. UPPER(TRIM(customer_code)) increases matches in the code fixture. What must be established before accepting it as the identity rule?

A. Only that every expression returns text.

B. Only that the joined amount increases.

C. That the original reference key was unique before transformation.

D. An authorized equivalence rule and uniqueness of the transformed reference key.

15. For C17 in customers LEFT JOIN orders, which values describe COUNT(*), COUNT(o.order_id), and SUM(o.amount_cents) before presentation conversion?

A. 1, 0, NULL

B. 0, 0, 0

C. 1, 1, 0

D. 0, 1, NULL

16. The balanced fixture preserves both row count and cents after joining. Why is the result still invalid as one row per source order?

A. A valid join must have equal source-table sizes.

B. B03 has a zero amount, which joins cannot preserve.

C. Two different region labels must always be combined.

D. B01 repeats while B02 disappears.

17. Which complete relationship is supported by the branch fixture?

A. Match local_id alone because it is called an ID.

B. Match equal branch AND equal local_id.

C. Match branch alone because every order has a branch.

D. Match equal branch OR equal local_id.

18. Z02 has 600 cents and Z03 has -600 cents; both are unmatched. What conclusion follows if the INNER JOIN amount equals the source amount?

A. The omitted amounts may cancel, so event identities still need checking.

B. The negative event should be deleted before checking.

C. A DISTINCT amount sum will prove coverage.

D. Neither order can have been dropped.

19. In the capstone, V04 joins to K3, whose region is NULL. Which classification is correct?

A. missing_source_key, because region is missing

B. absent_reference_key, because its amount is zero

C. matched, with an incomplete region attribute

D. unmatched, because NULL must mean no customer

20. Starting from the unchanged non-draft capstone, add only V09 with absent customer K9, zero cents, and posted status. What should a verified LEFT JOIN produce?

A. 7 rows and 4100 cents

B. 8 rows and 4100 cents

C. 8 rows and 4700 cents

D. 7 rows and 3500 cents

Sources and further reading

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

SQLite built-in aggregate functions
https://www.sqlite.org/lang_aggfunc.html

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

Datatypes in SQLite
https://www.sqlite.org/datatype3.html

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

SQLite foreign key support
https://www.sqlite.org/foreignkeys.html

SQLite built-in scalar SQL functions
https://www.sqlite.org/lang_corefunc.html