

VOLUME THREE · NORA · PLATES 18–27

Off the *Clock*

Instructor Field Notes

Ten collectible plates from the instructor fan-pack, filed and annotated by the instructor in them. Every margin note is a query that was executed, with the numbers it returned.

FIRST EDITION · SEPTEMBER 2026



BEFORE YOU OPEN THE DRAWER

The archive keeps a card.

The building used to run a card catalogue. One card per question it had been asked, filed under the person who asked it, stamped with the time it was answered. Nobody has filed in it since the warehouse came online. I maintain it anyway. It is accurate, which is more than this floor manages in a good quarter.

You have a drawer now. Ten questions in one week, so ten cards. I wrote them.

Taking the date stamp off the catalogue desk needs a requisition. I filled one in, signed it as the archivist, and countersigned it as the requester.

The pictures are not mine and I did not choose them. Every query on these cards was executed. Both numbers are what came back. Read the cards.

— Nora

Ten plates, ten margin notes. Every query was executed against DuckDB 1.5.3-r.3 on 2026-09-05 and ships beside this book as `nora-companion.sql`, with the setup, the queries and the rows they returned. Play the game: `tsunderesql.itch.io/tsunderesql`

PLATE INDEX

Ten cards, one drawer.

18	Cardinality Instructor count-distinct-after-join · 5 / 5 / 2	RARE	04
19	Archive Tea where-vs-having · 3 customers / 2 customers	RARE	06
20	Relational Hall implicit-cross-join · 1335.00 / 445.00	EPIC	08
21	Schema Corridor Profile not-in-null · 0 rows / 2 rows	RARE	10
22	Performance Pool rows-vs-range-peers · 225.00 / 150.00	EPIC	12
23	Data Theatre Cabaret distinct-hides-grain · 450.00 / 285.00	LEGENDARY	14
24	After Hours null-in-aggregates · 4 / 2.4	LEGENDARY	16
25	Storm Index Observatory between-timestamps-half-open · 245.00 / 745.00	LEGENDARY	18
26	Black Marble Winter Pool where-vs-on-left-join · 3 rows / 4 rows	EPIC	20
27	Desert Sleeper Library sum-fanout-line-items · 455.00 / 245.00	LEGENDARY	22

How to read a card. Each card prints a query exactly as it ran and the question it was meant to answer. Most of them are wrong; card one's three counters are all correct and disagree anyway. Both numbers come from a fixture executed against DuckDB 1.5.3-r.3 on 2026-09-05, and the whole set ships beside this book as **nora-companion.sql**. Rarity labels are packaging only and unlock nothing in the game. The three volumes of the fan-pack share one bank of SQL cases; each instructor files them in her own hand.



Cardinality Instructor

RARE

Card one. Stamped 21:40, which is when I stopped teaching and started writing. You asked how many orders Aiko has and you shipped the answer five. COUNT(*) says 5. COUNT(o.order_id) says 5. COUNT(DISTINCT o.order_id) says 2. All three ran in one query and all three are correct. Two orders, five line items, one FROM clause. I drew that crow's foot this morning and left it up all day. You had forty minutes in front of it. Carla has no orders and still counts as 1.

MARGIN NOTE

COUNT(*) counts join rows; COUNT(DISTINCT order_id) counts orders

```
SELECT c.name,
       COUNT(*)                    AS count_star,
       COUNT(o.order_id)           AS count_order_id,
       COUNT(DISTINCT o.order_id) AS count_distinct_orders
FROM customers c
LEFT JOIN orders     o  ON o.customer_id = c.customer_id
LEFT JOIN line_items li ON li.order_id   = o.order_id
GROUP BY c.name;
```

CARLA · COUNT(*)	CARLA · COUNT(O.ORDER_ID)
1	0

COUNT(*) counts rows of the joined result, COUNT(col) counts non-NULL values, COUNT(DISTINCT col) counts distinct ones. Carla has no orders, so the outer join hands her one all-NULL row that COUNT(*) still counts.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 18

Cardinality Instructor

RARE



Archive Tea

RARE

22:15. The tea was poured at 21:50 and I have not touched it. Your second question errored, so you moved the condition into HAVING and called it fixed. It compiles. It also reports three customers over 100.00 of revenue, and the true count is two. Customer 2 clears the bar on a 300.00 order that was cancelled. WHERE removes rows before the sum. HAVING removes groups after it. You needed both and wrote only the one that silenced the error. The date stamp is bolted to this desk. I have a screwdriver.

MARGIN NOTE

WHERE cannot see an aggregate, and moving the filter changes the answer

```
SELECT customer_id, SUM(order_total) AS revenue
FROM orders
GROUP BY customer_id
HAVING SUM(order_total) > 100
ORDER BY customer_id;
-- the version that errors: WHERE SUM(order_total) > 100
-- Binder Error: WHERE clause cannot contain aggregates!
```

HAVING ALONE	WHERE + HAVING
3	2

WHERE runs before grouping and cannot reference an aggregate; HAVING runs after. The clause you filter in decides whether a row is excluded from the sum or a whole group is excluded from the output. Customer 2 only clears 100.00 on a cancelled order.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 19

Archive Tea

RARE

07



Relational Hall

EPIC

The hall is the only room where the whole schema is visible at once, so I bring the hard ones here. Yours was not hard. FROM orders o, customers c, and no predicate. That is legal SQL. It returned 12 rows and 1335.00 against a true 4 rows and 445.00, which is exactly three times, because there are three customers. Look at the third result. Aiko 445.00, Bruno 445.00, Carla 445.00. Every customer credited with every order. I rewrote it with an ON clause so the same mistake would refuse to run. That took nine minutes I had not planned to spend.

MARGIN NOTE

A comma join with no predicate multiplies revenue by three

```
SELECT COUNT(*)           AS rows_returned,
       SUM(o.order_total) AS revenue
FROM orders o, customers c
WHERE o.order_total > 0;
```

COMMA JOIN	PREDICATE RESTORED
1335.00	445.00

A comma in FROM is a cross join, so omitting the predicate is valid SQL that multiplies the row count and every additive aggregate by the other table’s cardinality. Grouped by name, all three customers report the same 445.00.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 20

Relational Hall

EPIC

09



Schema Corridor Profile

RARE

I read this card three times before filing it. The procedure allows one. Your NOT IN returned zero rows and you took zero rows as a clean result. Order 204 is a guest checkout with a NULL customer_id. Aiko and Bruno appear in that subquery, so their test is FALSE. Carla and Dmitri appear nowhere in it, so their test should be TRUE; the NULL leaves it UNKNOWN instead. WHERE keeps only TRUE, so nothing comes back. NOT EXISTS finds Carla and Dmitri. An empty result is a claim about your data. There are four customers in that table. You could have counted them by hand.

MARGIN NOTE

One NULL in the subquery makes NOT IN return zero rows

```
SELECT c.customer_id, c.name
FROM customers c
WHERE c.customer_id NOT IN (SELECT o.customer_id FROM orders o)
ORDER BY c.customer_id;
```

NOT IN	NOT EXISTS
0 rows	2 rows

NOT IN is a chain of inequality tests under three-valued logic. Aiko and Bruno match a value in the list, so their test is FALSE. Carla and Dmitri match nothing, but order 204's NULL customer_id leaves their test UNKNOWN. WHERE keeps only TRUE, so the result is empty.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 21

Schema Corridor Profile

RARE



Performance Pool

EPIC

Stamped 06:40, before the lanes opened. This is the hour I keep for myself, because it is the only one that does not lie to me, and I was already here. SUM(amount) OVER (ORDER BY paid_on) with no frame clause defaults to RANGE, and under RANGE a row is given every peer that shares its value. Payments 2 and 3 both landed on 2026-03-02, so payment 2, at 50.00, is credited with payment 3 as well and reports 225.00. Two swimmers touching the wall together still get two times. Name ROWS and a tiebreaker and payment 2 reads 150.00.

MARGIN NOTE

Two payments on the same day get the same running total under RANGE

```
SELECT payment_id, paid_on, amount,
       SUM(amount) OVER (ORDER BY paid_on) AS running_total
FROM payments
ORDER BY payment_id;
```

DEFAULT FRAME · PAYMENT 2	ROWS + TIEBREAKER
225.00	150.00

With ORDER BY and no frame clause the default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW, so the current row includes every row sharing its ORDER BY value. PostgreSQL, MySQL 8.4, SQLite and DuckDB were each read on this default and this peer rule, and all four state it the same way.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 22

Performance Pool

EPIC

PLATE 23 · DETAIL

FILED · CARD 06



Data Theatre Cabaret

LEGENDARY

I did not come for the show. I came because the house keeps its takings in a ledger I wanted to see, and the card went in my cuff on the way. Stamped 23:58 in a corridor, because there was nowhere to put a card down. You caught the fanout yourself. That was good. Then you wrote SUM(DISTINCT o.order_total). 450.00 became 165.00. The true revenue is 285.00. DISTINCT inside an aggregate deduplicates values. Orders 101 and 102 are both 120.00 and are different orders, so one of them stopped existing. 450.00 gets questioned by somebody. 165.00 gets filed.

MARGIN NOTE

SUM(DISTINCT x) dedupes values, not rows, and eats a real order

```
SELECT SUM(DISTINCT o.order_total) AS revenue
FROM orders o
JOIN line_items li ON li.order_id = o.order_id;
```

SUM(DISTINCT) AS PRINTED	DEDUPE BY KEY
165.00	285.00

Three numbers, one revenue. A plain SUM after the join reads 450.00, which is obviously too high and gets investigated. The query above reads 165.00, because DISTINCT inside an aggregate deduplicates values rather than rows and throws away a second order that happens to share the total 120.00. Deduplicating by order_id gives 285.00. Only the too-low number ships.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 23

Data Theatre Cabaret

LEGENDARY



After Hours

LEGENDARY

This card is not stamped. Both of your numbers were correct, which is the problem. AVG(rating) returns 4 because AVG ignores the NULLs and only three of the five reviews carry a star. SUM(rating) divided by COUNT(*) returns 2.4, which is the same as scoring an unrated review as a zero. The dashboard heading has to say which of those it means; the warehouse will not choose for you. Two support tickets, no refunds, and SUM(refund_amount) returns NULL rather than 0.00. Nothing is not the same as zero. I took my glasses off at some point and have not put them back on.

MARGIN NOTE

AVG answers one question; the heading above it may be asking another

```
SELECT COUNT(*)                    AS reviews,
       COUNT(rating)               AS rated_reviews,
       AVG(rating)                 AS avg_of_rated,
       SUM(rating) * 1.0 / COUNT(*) AS avg_over_all
FROM order_reviews;
```

AVERAGE OF RATED REVIEWS	STARS OVER ALL REVIEWS
4	2.4

AVG, SUM and COUNT(column) ignore NULL inputs; COUNT(*) counts rows regardless. So AVG divides by the 3 rated reviews, not by all 5. DuckDB names the exceptions: list (array_agg), first and last do not ignore NULLs, and every general aggregate except count returns NULL on an empty group, which is why SUM over a column that is entirely NULL returns NULL and not 0.00. COALESCE(SUM(refund_amount), 0) returns 0.00.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 24

After Hours

LEGENDARY



Storm Index Observatory

LEGENDARY

The storm put the observatory on backup at 02:11 and I stamped this one by the emergency light. BETWEEN DATE ‘2026-12-01’ AND DATE ‘2026-12-31’ reported 3 orders and 245.00. December had 4 and 745.00. BETWEEN includes both endpoints, so a date-valued upper bound is midnight at the start of the 31st, and order 104 was placed at 2026-12-31 23:45:10 for 500.00. It fell off the end of the year and nothing said so. You wrote the half-open version at the bottom of the page and crossed it out. It was correct. Next time, do not cross it out.

MARGIN NOTE

BETWEEN on a timestamp drops almost a whole final day

```
SELECT COUNT(*) AS orders, SUM(order_total) AS revenue
FROM orders
WHERE placed_at BETWEEN DATE '2026-12-01' AND DATE '2026-12-31';
```

BETWEEN TWO DATES	HALF-OPEN INTERVAL
245.00	745.00

BETWEEN includes both endpoints, so a date-valued upper bound means midnight at the start of the 31st. That midnight instant is in range; anything logged later on the 31st is not. Order 104, placed at 2026-12-31 23:45:10 for 500.00, falls outside and nothing reports it as missing.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 25

Storm Index Observatory

LEGENDARY



Black Marble Winter Pool

EPIC

Stamped 01:30 on the marble. The cold out here matches the cold I keep. I came up for a forty-page paper on market microstructure and a red pen, which I read for enjoyment. The card took an ink stain on the way. Your LEFT JOIN returned 2 rows. Written correctly it returns 4. Putting o.status = ‘shipped’ in the WHERE applies it after the join, so Bruno’s cancelled row fails it and Dmitri’s NULL-filled row does too. Both leave the report. Your OR o.status IS NULL patch gets you 3. It rescues Dmitri, who has no orders, and loses Bruno, whose only order was cancelled.

MARGIN NOTE

OR IS NULL still loses the customer whose only order was cancelled

```
SELECT c.name, COUNT(o.order_id) AS shipped_orders
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id
WHERE o.status = 'shipped' OR o.status IS NULL
GROUP BY c.name
ORDER BY c.name;
```

OR IS NULL PATCH	FILTER IN ON
3 rows	4 rows

WHERE runs after the join and judges every row it produced. The plain predicate loses Bruno on his cancelled row and Dmitri on his NULL-filled one. The OR IS NULL patch rescues Dmitri, whose row really is NULL-filled, and still loses Bruno, whose cancelled order is a real row carrying a real status.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 26

Black Marble Winter Pool

EPIC



Desert Sleeper Library

LEGENDARY

Card ten. Stamped 05:12, in a library that moves at ninety kilometres an hour. Last question: the revenue you sent finance. 455.00. The true figure is 245.00. Joining line items copied order 101 twice and order 103 three times, so 120.00 was counted as 240.00 and 45.00 as 135.00. Sum the line items, or use EXISTS, and it is 245.00 either way. You wrote the EXISTS version in the margin before I reached it. I filed your wording and not mine, because yours was shorter. The drawer is on your desk. Ten cards, in order. The stamp is back where it belongs. You keep the drawer.

MARGIN NOTE

SUM(order_total) after joining line items double-counts revenue

```
SELECT SUM(o.order_total) AS revenue
FROM orders o
JOIN line_items li ON li.order_id = o.order_id;
```

ORDER 101 COUNTED AS	ORDER 101 IS
240.00	120.00

A one-to-many join repeats each order once per line item, so an aggregate over an order-grain column counts that value once per child row. Order 101 gets two copies and order 103 gets three, which turns 245.00 into 455.00.

Executed against DuckDB 1.5.3-r.3, 2026-09-05 · companion file: nora-companion.sql



PLATE 27

Desert Sleeper Library

LEGENDARY

COLOPHON

What this is, and what made it.

18	Cardinality Instructor	RARE	count-distinct-after-join
19	Archive Tea	RARE	where-vs-having
20	Relational Hall	EPIC	implicit-cross-join
21	Schema Corridor Profile	RARE	not-in-null
22	Performance Pool	EPIC	rows-vs-range-peers
23	Data Theatre Cabaret	LEGENDARY	distinct-hides-grain
24	After Hours	LEGENDARY	null-in-aggregates
25	Storm Index Observatory	LEGENDARY	between-timestamps-half-open
26	Black Marble Winter Pool	EPIC	where-vs-on-left-join
27	Desert Sleeper Library	LEGENDARY	sum-fanout-line-items

PUBLICATION

Off the Clock: Instructor Field Notes
Volume three · Nora · plates 18–27
TsundereSQL · Orphan Rows
tsunderesql.com

EDITIONS

Dark Edition and Paper Edition.
Both A4, 24 pages, from one source.
Didot, Iowan Old Style, SF Mono.
Rendered from HTML via Chromium.

MARGIN NOTES

Every query executed against DuckDB 1.5.3-r.3
on 2026-09-05, harness exit code 0.
Setup, queries and returned rows:
nora-companion.sql

PLATES

Ten of the 27 plates in the Instructor
Fan-Pack, approved 2026-09-04.
Volume three of three: Tsumugi, then Aimi,
then Nora.

RIGHTS

© 2026 Orphan Rows · TsundereSQL
Off the Clock: Instructor Field Notes.
Plate titles and margin notes by the studio.

WHERE TO FIND IT

Play TsundereSQL:
tsunderesql.itch.io/tsunderesql
Wishlist on Steam:
TsundereSQL on Steam