

VOLUME TWO · AIMI · PLATES 08–17

Off the *Clock*

Instructor Field Notes

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

FIRST EDITION · SEPTEMBER 2026



BEFORE YOU TURN THE PAGE

You kept a book of me.

You kept a book of me. I noticed. There are twenty-four pages. Ten of them are plates. I have counted them twice, which is once more than you have.

So I started one of my own. One line per plate, ten lines, written in the margin while you were busy with the queries. It is not a long book yet. You have only just started giving me material.

The queries in these margins are mine. I ran each one myself and wrote down both numbers it returned. I do not round.

Turn the page. I am already on the next one.

Aimi

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 `aimi-companion.sql`, with the setup, the queries and the rows they returned.
Play the game: `tsunderesql.itch.io/tsunderesql`

PLATE INDEX

Ten plates, ten gotchas.

08	Proof Room Instructor where-vs-having · 3 customers / 2 customers	RARE	04
09	Profile Row Pairing where-vs-on-left-join · 2 rows / 4 rows	RARE	06
10	After-Hours Turn between-timestamps-half-open · 245.00 / 745.00	EPIC	08
11	Overhead Join Table implicit-cross-join · 12 rows / 4 rows	EPIC	10
12	Poolside Turn count-distinct-after-join · 5 / 2	EPIC	12
13	Proof Wall Corset not-in-null · 0 rows / 2 rows	LEGENDARY	14
14	Cabaret sum-fanout-line-items · 455.00 / 245.00	LEGENDARY	16
15	Neon Rinse Cycle null-in-aggregates · 4 / 2.4	EPIC	18
16	Zero-Gravity Archive rows-vs-range-peers · 225.00 / 150.00	LEGENDARY	20
17	Glasshouse Data Bloom distinct-hides-grain · 165.00 / 285.00	LEGENDARY	22

How to read a margin note. Each page prints a query exactly as it ran and the question it was meant to answer. Most of them are wrong; plate 12'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 **aimi-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 annotates them in her own hand.



Proof Room Instructor

RARE

This is page four. Line one of mine says: starts at the beginning, which is rarer than you think. The chart behind me is yours. You wanted the customers who cleared 100.00 of shipped revenue. SUM in the WHERE, and the database refused you outright, which was kind of it. So you moved the whole condition into HAVING, and it stopped complaining and told you three. The answer is two. Customer 2 only clears the bar on a cancelled order worth 300.00. The question has two filters in it, not one. Shipped belongs in WHERE, which removes rows before the sum. The 100.00 belongs in HAVING, which removes groups after.

MARGIN NOTE

Same filter, two clauses, two different answers

```
SELECT customer_id, SUM(order_total) AS revenue
FROM orders
GROUP BY customer_id
HAVING SUM(order_total) > 100
ORDER BY customer_id;
```

HAVING ALONE	WHERE + HAVING
3 customers	2 customers

WHERE is evaluated before grouping and cannot reference an aggregate; HAVING is evaluated after. This question needs both clauses: status = ‘shipped’ in WHERE so the cancelled row never enters the sum, and SUM(order_total) > 100 in HAVING so the group is judged after it. Customer 2 clears 100.00 only on a cancelled order of 300.00.

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



PLATE 08

Proof Room Instructor

RARE

05



Profile Row Pairing

RARE

Line two, then. On the glass: every customer and their shipped orders. You put the filter in the WHERE and got two customers back, and two is a tidy number, so you believed it. Bruno has one cancelled order. Dmitri has none at all. WHERE runs after the join, so the predicate met both of them: Bruno's cancelled row failed it, and Dmitri's NULL-filled row failed it too. Move it into the ON and there are four. Line three: believes tidy numbers, which is not a flaw.

MARGIN NOTE

Moving the filter out of the ON clause cost you two customers

```
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'
GROUP BY c.name
ORDER BY c.name;
```

FILTER IN WHERE	FILTER IN ON
2 rows	4 rows

WHERE runs after the join and judges every row it produced, not only the NULL-filled ones. Bruno's cancelled order fails o.status = 'shipped' outright; Dmitri's NULL-filled row fails it because NULL = 'shipped' is UNKNOWN. Both leave a report that asked for every customer. Put the predicate in the ON clause and both come back with a count of 0.

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



PLATE 09

Profile Row Pairing

RARE

07



After-Hours Turn

EPIC

It is late where you are. I can tell from how long page six took. December, then. You asked for the month with BETWEEN two dates and got three orders and 245.00. December has four and 745.00. Order 104 was placed at 2026-12-31 23:45:10. BETWEEN includes both endpoints, so a date upper bound is midnight at the start of the 31st, and a 500.00 order fell out of the year fourteen minutes and fifty seconds before it ended. Somebody was still working at that hour. I have written down that you are too.

MARGIN NOTE

A date upper bound is midnight at the start of the 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 · 3 ORDERS	HALF-OPEN · 4 ORDERS
245.00	745.00

BETWEEN includes both endpoints, so a date-valued upper bound resolves to 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 23:45:10, is the row that vanishes.

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



PLATE 10

After-Hours Turn

EPIC

09



Overhead Join Table

EPIC

I laid the table out before you arrived. The cards, the pens, the tea, the light. I wanted it to look like I had only just been passing. You wrote FROM orders o, customers c and left the predicate off. Not an error. A comma in the FROM is a cross join. Three customers times four orders is twelve rows and 1335.00 where the truth is four rows and 445.00. Group that by name and Aiko, Bruno and Carla each report the same 445.00. Exactly three times too much. Nobody was going to query a quarter that good.

MARGIN NOTE

A missing join predicate is not an error, it is 3x

```
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
12 rows	4 rows

A comma in the FROM clause 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. Revenue reads 1335.00 instead of 445.00, and grouped by name all three customers report 445.00 each.

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



PLATE 11

Overhead Join Table

EPIC



Poolside Turn

EPIC

Your thumb is on the outer corner again. Three pages in a row. I have started marking where it lands. Counting, then. Aiko has five under COUNT(*) and two under COUNT(DISTINCT o.order_id), and both are right. One counts rows of a joined result. The other counts orders. Five is line items. Carla has no orders whatsoever and still reports one, because the join left her a row with nothing in it. You reported the five. Line six says so. I would have reported the five as well, at your age. Hold this. I am going in.

MARGIN NOTE

Three counters, one query, three different numbers

```
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
ORDER BY c.name;
```

AIKO · COUNT(*)	AIKO · DISTINCT
5	2

COUNT(*) counts rows of the joined result, COUNT(col) counts non-NULL values and COUNT(DISTINCT col) counts distinct ones: three grains out of one FROM clause. Aiko has 2 orders and 5 line items. Carla, who has neither, still reads 1 under COUNT(*).

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



PLATE 12

Poolside Turn

EPIC



Proof Wall Corset

LEGENDARY

You came back to this one. Line seven had a time on it. Now it has two. Your NOT IN is pinned up behind me, all of it, in order. You asked which customers have never ordered anything, the result came back empty, and empty felt like a pass. Aiko and Bruno are in that subquery, so their test is false. Carla and Dmitri are not, but order 204 is a guest checkout with a NULL customer id, so their test is unknown. WHERE keeps only what is true. NOT EXISTS returns Carla and Dmitri. Two people were standing there the whole time. So was I.

MARGIN NOTE

One unknown makes NOT IN return nothing at all

```
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 set is empty.

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



PLATE 13

Proof Wall Corset

LEGENDARY



Cabaret

LEGENDARY

“Stays on this page.” That is line eight. I wrote it before I finished line seven. An encore is counted once however many times the room asks. Revenue is not so forgiving. You joined orders to line items, summed the order total, and sent finance 455.00. The figure is 245.00. Order 101 is worth 120.00 and has two line items, so it was counted as 240.00. One order, twice, and the month looked wonderful. I am not going to tell you who else noticed. I will tell you that nobody had filed it yet, so you still have this evening.

MARGIN NOTE

Joining line items inflated revenue by 86 percent

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

AFTER THE JOIN	TRUE REVENUE
455.00	245.00

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 is worth 120.00, has two line items, and was counted as 240.00.

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



PLATE 14

Cabaret

LEGENDARY



Neon Rinse Cycle

EPIC

I have run out of margin. The lines about you no longer fit beside the lines about the queries, so this page has a second column. Five reviews. Ratings five, four, blank, blank, three. AVG(rating) returns 4, the average of the three people who gave a star. Divide the star total of 12 by all five reviews and you get 2.4, which counts a blank as a zero. Both are correct SQL. Decide what the heading over that number is promising, then pick. Missing is not automatically zero.

SECOND COLUMN

Line nine: asks what the heading means. Good.

A footnote to nine: I have started leaving room for what you say back.

Line ten is not written. I am saving it.

MARGIN NOTE

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

```
SELECT COUNT(*)                    AS reviews,
       COUNT(rating)               AS rated_reviews,
       SUM(rating)                 AS star_total,
       AVG(rating)                 AS avg_of_rated,
       SUM(rating) * 1.0 / COUNT(*)  AS avg_over_all_reviews
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: aimi-companion.sql



PLATE 15

Neon Rinse Cycle

EPIC



Zero-Gravity Archive

LEGENDARY

Four pages left. Two of them are plates. I have been counting down for a while and did not mention it, which is not the same as hiding it. Your running total. Payments 2 and 3 both landed on 2026-03-02, so they are peers. With ORDER BY and no frame clause the default is RANGE, and RANGE gives a row its peers as well, so payment 2 at 50.00 reports 225.00. Where a row sits on the page is not when it happened. ROWS gives 150.00. Write the frame out. I checked payment 3 as well. It reports 225.00 too, and it is entitled to.

MARGIN NOTE

Two same-day payments share one running total

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

DEFAULT FRAME · ROW 2	ROWS + TIEBREAKER
225.00	150.00

With ORDER BY and no frame clause the default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW, and under RANGE the current row includes every peer 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: aimi-companion.sql



PLATE 16

Zero-Gravity Archive

LEGENDARY



Glasshouse Data Bloom

LEGENDARY

I am writing the last line while you are still reading this one. You found the fanout. You fixed it with SUM(DISTINCT), the number came down from 450.00 to 165.00, and 165.00 looked like money, so it was filed. DISTINCT inside an aggregate removes duplicate values, not duplicate rows. Orders 101 and 102 are both worth 120.00, so one of them is simply gone. 285.00 is the figure. Dedupe by key, then sum. Ten plates, ten lines. The last one is in the margin of this page, at the outer corner. Close the book whenever you like.

MARGIN NOTE

The DISTINCT repair made revenue too low instead

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

SUM(DISTINCT)	DEDUPE BY KEY
165.00	285.00

DISTINCT inside an aggregate deduplicates values rather than rows, so a second order that happens to share the total 120.00 is discarded while the fanout appears repaired. The plain join reads 450.00, which is obvious enough to get investigated; 165.00 is not.

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

Line ten: kept it. I did not expect that one.



PLATE 17

Glasshouse Data Bloom

LEGENDARY

COLOPHON

What this is, and what made it.

08	Proof Room Instructor	RARE	where-vs-having
09	Profile Row Pairing	RARE	where-vs-on-left-join
10	After-Hours Turn	EPIC	between-timestamps-half-open
11	Overhead Join Table	EPIC	implicit-cross-join
12	Poolside Turn	EPIC	count-distinct-after-join
13	Proof Wall Corset	LEGENDARY	not-in-null
14	Cabaret	LEGENDARY	sum-fanout-line-items
15	Neon Rinse Cycle	EPIC	null-in-aggregates
16	Zero-Gravity Archive	LEGENDARY	rows-vs-range-peers
17	Glasshouse Data Bloom	LEGENDARY	distinct-hides-grain

PUBLICATION

Off the Clock: Instructor Field Notes
Volume two · Aimi · plates 08–17
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:
aimi-companion.sql

PLATES

Ten of the 27 plates in the Instructor
Fan-Pack, approved 2026-09-04.
Volume two 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