VOLUME ONE · TSUMUGI · PLATES 01–07

Off the *Clock*

Instructor Field Notes

Seven 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

The academy is closed.

The lights in the hall are on a timer. They went off an hour ago, which is how I know what time it is.

You left a week of queries in my tray on Friday. Nine. Seven were wrong in ways worth writing down, so I wrote them down. That folder is not supposed to leave the building. It has been in my bag since Friday.

Somebody photographed the writing-down. Somebody else printed it at this size. Fine.

Each margin note here is a query I ran myself and the two numbers it returned. Run them yourself; the file is in the download beside this book. I checked them twice.

Don’t dog-ear the pages.

— *Tsumugi*

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

PLATE INDEX

Seven plates, seven gotchas.

How to read a margin note. Each page prints a query exactly as it ran and the question it was meant to answer. Some of these queries are wrong; plate 04'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 **tsumugi-companion.sql**. Rarity labels are packaging only and unlock nothing in the game. The pool, the observatory, the sleeper train and the cathedral are collectible vignettes rather than events in the story.

Plum Robe Field Notes

LEGENDARY

The printouts on the floor are yours. I emptied the folder onto the rug. The table had dinner on it and the dinner was staying where it was. The robe is not a statement. The flat is cold. Page one is the LEFT JOIN where you put the filter in the WHERE. You asked for every customer and their shipped orders and you got two customers. Bruno and Dmitri are still customers. They have been customers this whole time. I have marked it. I am not sending it tonight. You would only fix it badly at midnight.

MARGIN NOTE

Your LEFT JOIN quietly became an INNER JOIN

```
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: tsumugi-companion.sql

PLATE 01

Plum Robe Field Notes

LEGENDARY

Poolside Field Notes

EPIC

I did not bring the folder. The bag it lives in came, and it was inside the bag. Page two. NOT IN, with a NULL sitting in the subquery. Zero rows came back and you read zero rows as good news. Aiko and Bruno are in that subquery, so for them NOT IN is false. Carla and Dmitri are not, but order 204 is a guest checkout with a NULL customer id, so their last comparison comes back unknown. WHERE keeps only what is true. NOT EXISTS returns the two. I am going back in the water. Read the note when I am not looking at you.

MARGIN NOTE

One NULL 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 is empty.

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

PLATE 02

Poolside Field Notes

EPIC

07

Tsundere Couture

LEGENDARY

The book in my lap is not work and the chair is not mine. I was going to give the evening to both of them. Then I remembered the revenue number you sent finance on Wednesday and I have been sitting here with the folder open since. You joined orders to line items and summed the order total. 455.00. The true figure is 245.00. Order 101 is worth 120.00 and got counted twice. Somebody approved that. I would like the name and the approval timestamp. I will be up anyway.

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 and got counted as 240.00.

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

PLATE 03

Tsundere Couture

LEGENDARY

PLATE 04 · DETAIL

Corrects the Join

RARE

In uniform, in front of the room. I put your count on the board this morning with your name cut off the top. COUNT(*) says Aiko has five. COUNT(DISTINCT o.order_id) says two. Both are right. They count different things. Five is line items. Two is orders. You picked five because five is bigger. Carla has no orders at all and still reads one under COUNT(*). The folder went back in my bag at the bell. You worked it out from the board before I finished and did not say so. Good.

MARGIN NOTE

Three counts, 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;
```

AIKO · COUNT(*)	AIKO · DISTINCT
5	2

COUNT(*) counts rows of the joined result and COUNT(DISTINCT o.order_id) counts orders. 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: tsumugi-companion.sql

PLATE 04

Corrects the Join

RARE

Abyssal Query Observatory

LEGENDARY

Two hundred metres down. No signal, and nobody to ask me anything. I brought a novel. The folder was under the novel. Page five is your running total. Payments 2 and 3 both landed on 2026-03-02, which makes them peers. Under a bare ORDER BY the frame defaults to RANGE, and RANGE gives a row its peers as well, so payment 2 at 50.00 reports 225.00. Row order on a screen is not a clock. ROWS and a tiebreaker give it 150.00. Write the frame out. The jellyfish are on a schedule. I had twenty minutes. I spent nine of them writing the frame clause out in longhand for you.

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, so the current row includes every row sharing its ORDER BY value. Name ROWS instead, with payment_id as a tiebreaker, and payment 2 reads 150.00.

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

PLATE 05

Abyssal Query Observatory

LEGENDARY

Midnight Proof Express

LEGENDARY

Sleeper train, 23:00, compartment to myself until you looked. The December close is on the table because it would not fit in my bag with the folder, so the folder is on the bunk. Your BETWEEN found three orders and 245.00. December has four and 745.00. Order 104 was placed at 2026-12-31 23:45:10, and BETWEEN's upper bound is midnight at the start of the 31st, so a 500.00 order fell off the end of the year. You noticed the total looked light. You were right and you shipped it anyway. Next time the total looks light, count December by hand before you send it.

MARGIN NOTE

BETWEEN dropped a 500.00 order on New Year's Eve

```
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 means midnight at the start of the 31st. That midnight instant is in range; anything logged later on the 31st, order 104 at 23:45:10 included, is not.

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

PLATE 06

Midnight Proof Express

LEGENDARY

PLATE 07 · DETAIL

Server Cathedral Encore

LEGENDARY

Last page. I lost a bet, the bet had witnesses, and the costume department keeps records. You found the fanout and you tried to fix it with SUM(DISTINCT). 450.00 was obviously wrong, so somebody would have caught it. 165.00 looks like money and gets filed. DISTINCT inside an aggregate deduplicates values. It does not deduplicate rows. Orders 101 and 102 are both 120.00, so one of them is simply gone. 285.00 is the number. The folder is on your desk. Page seven is on top, face up, marked. Fix it before I get in.

MARGIN NOTE

The DISTINCT fix 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.

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

PLATE 07

Server Cathedral Encore

LEGENDARY

COLOPHON

What this is, and what made it.

01	Plum Robe Field Notes	LEGENDARY	where-vs-on-left-join
02	Poolside Field Notes	EPIC	not-in-null
03	Tsundere Couture	LEGENDARY	sum-fanout-line-items
04	Corrects the Join	RARE	count-distinct-after-join
05	Abyssal Query Observatory	LEGENDARY	rows-vs-range-peers
06	Midnight Proof Express	LEGENDARY	between-timestamps-half-open
07	Server Cathedral Encore	LEGENDARY	distinct-hides-grain

PUBLICATION

Off the Clock: Instructor Field Notes
Volume one · Tsumugi · plates 01–07
TsundereSQL · Orphan Rows
tsunderesql.com

EDITIONS

Dark Edition and Paper Edition.
Both A4, 18 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:
tsumugi-companion.sql

PLATES

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

WHERE TO FIND IT

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