

Short version: your `QUALIFY ROW_NUMBER()` is collapsing duplicates from the **complaints** table.

You're partitioning by `(c.member_id, c.complaint_time)`. If the same complaint appears multiple times (different category/subcategory rows) with the **same** `(member_id, complaint_time)`, then all those rows land in the *same* partition. `QUALIFY ... = 1` keeps only one of them and filters out the rest — even when there's **no** matching session. That's why your 295k complaint rows drop to ~the number of **distinct (member_id, complaint_time)** (~155k). Your numbers match that: `COUNT(DISTINCT member_id, complaint_time) ≈ 155k`, and your result is ~154k. (Also, $95k + 49k \neq 154k$ — probably a typo; if it's 59k, it adds up.)

How to keep all 295k complaint rows and still attach the latest session (≤ 7 days)

Use a **LEFT JOIN LATERAL** (per-row correlated subquery). It picks the best session for *each* complaint row without deduplicating the complaints:

```
SELECT
  c.*,
  s1.session_id,
  s1.session_start_time
FROM complaints c
LEFT JOIN LATERAL (
  SELECT s.session_id, s.session_start_time
  FROM sessions s
  WHERE s.member_id = c.member_id
        AND s.session_start_time BETWEEN c.complaint_time - INTERVAL '7
DAY' AND c.complaint_time
  ORDER BY s.session_start_time DESC, s.session_id DESC
  LIMIT 1
) s1 ON TRUE;
```

This will return ~295k rows: one per complaint row; if no session in the window, the `s1.*` columns are NULL.

If you want one “best session” per distinct complaint key and then repeat it on every duplicate

Do it in two steps so the `QUALIFY` only runs on the *distinct keys*, then join back:

```

WITH targets AS (
    SELECT DISTINCT member_id, complaint_time
    FROM complaints
),
best AS (
    SELECT
        t.member_id,
        t.complaint_time,
        s.session_id,
        s.session_start_time,
        ROW_NUMBER() OVER (
            PARTITION BY t.member_id, t.complaint_time
            ORDER BY s.session_start_time DESC, s.session_id DESC
        ) AS rn
    FROM targets t
    LEFT JOIN sessions s
        ON s.member_id = t.member_id
        AND s.session_start_time BETWEEN t.complaint_time - INTERVAL '7
DAY' AND t.complaint_time
)
SELECT
    c.*,
    b.session_id,
    b.session_start_time
FROM complaints c
LEFT JOIN (
    SELECT member_id, complaint_time, session_id, session_start_time
    FROM best
    WHERE rn = 1
) b
    USING (member_id, complaint_time);

```

Quick sanity check counts

```

SELECT
    COUNT(*)                AS total_complaints,
    COUNT(b.session_id)     AS with_session,
    COUNT(*) - COUNT(b.session_id) AS without_session

```

```
FROM (  
    -- paste the final SELECT here  
) x;
```

TL;DR: `QUALIFY ROW_NUMBER() OVER (PARTITION BY member_id, complaint_time) = 1` after a LEFT JOIN effectively **dedupes your complaints** to one row per `(member_id, complaint_time)`. Use a LATERAL join per complaint row (or dedupe first, pick best session, and join back) to keep all rows.

You said:

if I had a unique identifier for complaints and used it in `QUALIFY ROW_NUMBER() OVER (PARTITION BY,)` this issue would have solved?

ChatGPT said:

Thought for 1m 1s

Short answer: **It depends what you want.**

If your goal is one row per *unique complaint* (collapsing category/subcategory duplicates), then **yes**: partition by that unique complaint key (e.g., `c.complaint_id`) and keep `ROW_NUMBER() = 1`. You'll end up with `COUNT(DISTINCT complaint_id)` rows, not 295k.

```
SELECT c.*, s.session_id, s.session_start_time  
FROM complaints c  
LEFT JOIN sessions s  
    ON s.member_id = c.member_id  
    AND s.session_start_time BETWEEN c.complaint_time - INTERVAL '7 DAY'  
    AND c.complaint_time  
QUALIFY ROW_NUMBER() OVER (  
    PARTITION BY c.complaint_id  
    ORDER BY s.session_start_time DESC, s.session_id DESC  
) = 1;
```

- This keeps one “best” session per complaint; complaints with no session still appear (with NULLs).

If your goal is to keep *all 295k complaint rows* (i.e., not collapsing category duplicates) and still attach the latest session per row, then partitioning by a unique complaint **row** id would also work. But if you don't have a stable per-row key, the safer pattern is a **LEFT JOIN LATERAL**:

```
SELECT c.*,
```

```

        s1.session_id,
        s1.session_start_time
FROM complaints c
LEFT JOIN LATERAL (
    SELECT s.session_id, s.session_start_time
    FROM sessions s
    WHERE s.member_id = c.member_id
        AND s.session_start_time BETWEEN c.complaint_time - INTERVAL '7
DAY' AND c.complaint_time
    ORDER BY s.session_start_time DESC, s.session_id DESC
    LIMIT 1
) s1 ON TRUE;

```

•

Why your current query shrank to ~155k: your **QUALIFY** partitions by (**member_id**, **complaint_time**), so it **dedupes** all complaint rows that share that pair (i.e., the category duplicates). Switching the partition to a **unique complaint key** dedupes to *one per complaint* (good if that's your intent). If you need *every* complaint row, use a per-row key (or the lateral join).

You said:

when I used LEFT JOIN LATERAL , snowflake throwed a unsopported format error im not sure i was using LEFT JOIN LATERAL (SELECT s.session_id, s.session_start_time FROM sessions s WHERE s.member_id = c.member_id AND s.session_start_time BETWEEN c.complaint_time - INTERVAL '7 DAY' AND c.complaint_time ORDER BY s.session_start_time DESC, s.session_id DESC LIMIT 1) s1 ON TRUE;

ChatGPT said:

Thought for 1m 9s

You're super close — Snowflake **does** support **LEFT JOIN LATERAL**, but two things often trigger errors:

1. the interval literal (**INTERVAL '7 DAY'**) formatting, and
2. minor syntax around **LIMIT/FETCH**.

Use Snowflake-friendly date math and standard **FETCH** to be safe.

Option A — LATERAL (one row per complaint row)

```

SELECT
  c.*,
  s1.session_id,
  s1.session_start_time
FROM complaints c
LEFT JOIN LATERAL (
  SELECT s.session_id, s.session_start_time
  FROM sessions s
  WHERE s.member_id = c.member_id
        AND s.session_start_time <= c.complaint_time
        AND s.session_start_time >= DATEADD('day', -7, c.complaint_time)
  ORDER BY s.session_start_time DESC, s.session_id DESC
  FETCH FIRST 1 ROWS ONLY
) s1 ON TRUE;

```

Notes:

- `DATEADD('day', -7, c.complaint_time)` avoids picky `INTERVAL` parsing.
- `FETCH FIRST 1 ROWS ONLY` is ANSI and works well in Snowflake (you can also use `LIMIT 1`).
- `ON TRUE` is the standard way to finish a `LEFT JOIN LATERAL`.

Option B — No LATERAL, keep all 295k rows using a per-row surrogate key

If LATERAL still throws an error in your environment, do this instead. It preserves every complaint row and still picks the latest session in the 7-day window:

```

WITH c AS (
  SELECT
    c.*,
    SEQ4() AS complaint_row_id  -- unique per row for this query run
  FROM complaints c
)
SELECT
  c.*,
  s.session_id,
  s.session_start_time

```

```

FROM c
LEFT JOIN sessions s
  ON s.member_id = c.member_id
  AND s.session_start_time <= c.complaint_time
  AND s.session_start_time >= DATEADD('day', -7, c.complaint_time)
QUALIFY ROW_NUMBER() OVER (
  PARTITION BY c.complaint_row_id
  ORDER BY s.session_start_time DESC, s.session_id DESC
) = 1;

```

Why this works:

- `SEQ4()` gives each source complaint row a unique id.
- The `LEFT JOIN` brings in all sessions in the window (or NULLs if none).
- `QUALIFY ROW_NUMBER() ... PARTITION BY complaint_row_id` picks **one best session per original complaint row** without collapsing duplicates across categories.

Sanity checks

```

-- Should be ~295k
SELECT COUNT(*) FROM (
  /* paste the final SELECT here */
) x;

-- With vs without sessions
SELECT
  COUNT(*) AS total_rows,
  COUNT(session_id) AS with_session,
  COUNT(*) - COUNT(session_id) AS without_session
FROM (
  /* paste the final SELECT here */
) x;

```

If you still get an error, paste the exact message. In most cases it's the interval literal or a missing `ON TRUE`.