Day 5 SQL Learning Notes

Date: November 1, 2025
Topic: Aggregate Functions & Window Functions
Focus: GROUP BY, HAVING, Window Functions (ROW_NUMBER, RANK, Running Totals)

What I Learned Today

Aggregate Functions

Functions that calculate across multiple rows

- **COUNT()**: Counts rows (COUNT(*) includes NULLs, COUNT(column) excludes NULLs)
- **SUM()**: Adds up values
- **AVG()**: Calculates average (ignores NULLs)
- **MIN()**: Finds minimum value
- **MAX()**: Finds maximum value
- All aggregate functions ignore NULL values except COUNT(*)

GROUP BY Clause

Most important for interviews and daily work

- Groups rows with same values into summary rows
- Must include all non-aggregated columns in GROUP BY
- Creates one row per unique combination
- Pattern: `SELECT category, COUNT(*) FROM table GROUP BY category`
- Can group by multiple columns: `GROUP BY col1, col2`
- Calculations happen AFTER grouping

Common mistake: Selecting column not in GROUP BY (causes error)

HAVING Clause

WHERE for aggregated results

- Filters AFTER aggregation (WHERE filters BEFORE)
- Uses aggregate functions in condition
- Pattern: `HAVING COUNT(*) > 5` or `HAVING AVG(balance) > 10000`

- WHERE filters individual rows, HAVING filters groups

Order matters:

1. WHERE (filter rows)
2. GROUP BY (group rows)
3. HAVING (filter groups)
4. ORDER BY (sort results)

Window Functions

Calculate across rows WITHOUT collapsing them

Key difference from GROUP BY:

- GROUP BY: Collapses rows into summary (many → few)
- Window Functions: Keeps all rows, adds calculations (many → many)

ROW_NUMBER():

- Assigns unique sequential number to each row
- Pattern: `ROW_NUMBER() OVER (ORDER BY col)`
- Used for: Deduplication, pagination, ranking

RANK() and DENSE_RANK():

- RANK(): Same rank for ties, skips next number (1,2,2,4)
- DENSE_RANK(): Same rank for ties, no gaps (1,2,2,3)
- Pattern: `RANK() OVER (ORDER BY balance DESC)`

PARTITION BY:

- Divides data into groups for window functions
- Like GROUP BY but doesn't collapse rows
- Pattern: `ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY date)`
- Restarts numbering for each partition

Running Totals:

- `SUM(amount) OVER (ORDER BY date)` - cumulative sum
- Pattern: `SUM(col) OVER (ORDER BY col)`

LAG() and LEAD():

- LAG(): Access previous row value
- LEAD(): Access next row value

- Pattern: `LAG(balance, 1) OVER (ORDER BY date)` - get previous balance

Key Patterns Learned

Pattern 1: Count by category

SELECT category, COUNT(*)
FROM table
GROUP BY category

Pattern 2: Filter groups

SELECT category, SUM(amount)
FROM table
GROUP BY category
HAVING SUM(amount) > 1000

Pattern 3: Ranking within groups

ROW_NUMBER() OVER (PARTITION BY category ORDER BY value DESC)

Pattern 4: Running total

SUM(amount) OVER (ORDER BY date)

Pattern 5: Compare to previous

value - LAG(value) OVER (ORDER BY date) AS change

Q1: Top N per category - Use ROW_NUMBER with PARTITION BY **Q2: Remove duplicates keeping latest** - ROW_NUMBER then filter WHERE rn = 1 **Q3: Running totals** - SUM() OVER (ORDER BY date) **Q4: Rank within groups** - RANK() OVER (PARTITION BY category ORDER BY value) **Q5: Month-over-month growth** - Use LAG to compare to previous period

Real-World Applications

- Sales reporting by region/product (GROUP BY)
- Top 5 customers per state (ROW_NUMBER with PARTITION BY)
- Running balance calculations (SUM OVER)
- Deduplication (ROW_NUMBER = 1)
- Ranking products by sales within categories

- Month-over-month comparisons (LAG)
- Finding gaps in sequences (LEAD/LAG)

Key Takeaways

1. **GROUP BY collapses, Window Functions don't** - critical difference
2. **HAVING filters groups, WHERE filters rows** - different purposes
3. **PARTITION BY is like GROUP BY for window functions** - restarts calculation
4. **ROW_NUMBER always unique, RANK can have ties** - choose based on need
5. **Running totals need ORDER BY** - defines the "running" sequence
6. **Window functions more powerful than GROUP BY** - keeps detail + adds summary

Common Mistakes Avoided

- Forgetting columns in GROUP BY (must include all non-aggregated)
- Using WHERE instead of HAVING for aggregates
- Confusing RANK vs DENSE_RANK (gaps vs no gaps)
- Missing PARTITION BY when need per-group calculations
- Wrong ORDER BY in window functions (affects calculation)

Formula Quick Reference

Basic Aggregation:

```
SELECT category, COUNT(*), SUM(amount), AVG(amount)
FROM table
GROUP BY category
HAVING COUNT(*) > 5
```

Ranking:

```
ROW_NUMBER() OVER (PARTITION BY category ORDER BY value DESC)
```

Running Total:

```
SUM(amount) OVER (ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
```

Previous Value:

LAG(value, 1) OVER (ORDER BY date)
