Day 4 SQL Learning Notes

Date: October 31, 2025
Topic: Set Operations & Subqueries

What I Learned Today

UNION vs UNION ALL - Combining Query Results

Most asked interview question about set operations

- **UNION**: Combines results from multiple queries and REMOVES duplicates (slower)
- **UNION ALL**: Combines results and KEEPS all rows including duplicates (faster)
- In real work, UNION ALL is used 80% of the time because it's faster
- Both queries must have same number of columns and compatible data types
- Column names come from first query
- ORDER BY goes at the end (applies to combined result)

When to use which:

- Use UNION when you need unique records only
- Use UNION ALL when combining non-overlapping data or duplicates don't matter
- UNION is slower because it checks for duplicates

Critical rules:

- Same column count in both queries
- Matching data types in order
- Only one ORDER BY at the very end
- Column names from first SELECT

Subqueries - Queries Inside Queries

The most important concept for interviews and daily work

Three places to use subqueries:

1. Subquery in WHERE clause (most common)

- Used for filtering based on calculated values

- Inner query runs first, result used in outer query
- Example: Find accounts with above-average balance
- Pattern: `WHERE column > (SELECT AVG(column) FROM table)`

2. Subquery in FROM clause (derived tables)

- Treat subquery result as a temporary table
- Must give it an alias
- Used to break complex queries into logical steps
- Pattern: `FROM (SELECT ...) AS alias`

3. Subquery in SELECT clause

- Creates calculated columns
- Runs for each row if correlated
- Pattern: `SELECT col, (SELECT ...) AS calculated_col`

IN vs EXISTS - Performance Pattern

Critical interview question - know the difference

IN operator:

- Subquery returns a complete list
- Checks if value is IN that list
- Better for small result sets
- Example: `WHERE customer_id IN (SELECT customer_id FROM accounts)`

EXISTS operator:

- Tests if subquery returns ANY row
- Stops at first match (early termination)
- Better for large result sets
- Example: `WHERE EXISTS (SELECT 1 FROM accounts WHERE customer_id = c.customer_id)`

Key differences:

- EXISTS is usually faster for large datasets (stops at first match)
- IN materializes entire list in memory
- NOT EXISTS handles NULLs better than NOT IN
- EXISTS just checks existence, doesn't care about values

"EXISTS is faster when checking if related records exist because it stops at first match. IN is better for small, static lists. For finding missing relationships, always use NOT EXISTS instead of NOT IN because it handles NULLs correctly."

Correlated vs Non-Correlated Subqueries ⭐⭐

Non-correlated subquery:

- Runs ONCE, independent of outer query
- Result is same for all rows
- Example: Compare all balances to global average
- Faster

Correlated subquery:

- Runs for EACH row of outer query
- References outer query columns
- Example: Compare each account to its customer's average
- Slower but more flexible
- Pattern: `WHERE col > (SELECT AVG(col2) FROM table WHERE id = outer.id)`

Use correlated when: Need row-specific calculations (per customer, per department, etc.)

Set Operations - Other Types

INTERSECT:

- Returns rows that appear in BOTH queries
- Used for finding common records
- Data reconciliation between systems

EXCEPT (or MINUS):

- Returns rows in first query but NOT in second
- Used for finding differences
- Identifying new or deleted records

Real-World Patterns Learned

Pattern 1: Find records above average

WHERE column > (SELECT AVG(column) FROM table)

Pattern 2: Find missing relationships

WHERE NOT EXISTS (SELECT 1 FROM related_table WHERE id = main.id)

Pattern 3: Multi-step analysis

FROM (SELECT ... GROUP BY ...) AS summary
WHERE summary.calculated_col > threshold

Pattern 4: Combine data from multiple sources

SELECT ... FROM source1
UNION ALL
SELECT ... FROM source2

Pattern 5: Row-by-row comparison

WHERE value > (SELECT AVG(value) FROM table WHERE category = outer.category)

Q1: Second highest value

- `WHERE value < (SELECT MAX(value) FROM table)` then get MAX
- Or use `ORDER BY value DESC LIMIT 1 OFFSET 1`

Q2: Find duplicates

- `GROUP BY column HAVING COUNT(*) > 1`

Q3: Customers without orders

- Use NOT EXISTS or LEFT JOIN with WHERE NULL

Q4: Above average performers

- Subquery in HAVING clause
- Compare grouped result to overall average

Q5: Running totals

- Correlated subquery summing all previous rows

Key Takeaways for Interviews

1. **UNION ALL is faster than UNION** - use it when possible
2. **EXISTS > IN** for checking existence in large tables
3. **NOT EXISTS is safer than NOT IN** (handles NULLs)
4. **Subqueries in WHERE are most common** - master this first
5. **Correlated subqueries run per row** - slower but powerful
6. **Derived tables break complexity** - use for multi-step logic
7. **Always alias derived tables** - mandatory syntax

Performance Tips Learned

- UNION ALL faster than UNION (no duplicate checking)
- EXISTS faster than IN for large result sets (early exit)
- NOT EXISTS safer than NOT IN (NULL handling)
- Non-correlated faster than correlated (runs once)
- Subqueries in FROM can be optimized by database
- Index columns used in subquery joins

Common Mistakes to Avoid

- Using UNION when UNION ALL is sufficient (slower for no reason)
- NOT IN with NULLs returns no results (use NOT EXISTS)
- Forgetting to alias derived tables in FROM clause
- Putting ORDER BY in each UNION query instead of at end
- Mismatched column counts or types in UNION
- Using correlated subquery when non-correlated would work (slower)

Daily Work Applications

- Combining data from current and archive tables (UNION ALL)
- Finding customers without recent activity (NOT EXISTS)
- Identifying above-average performers (subquery in WHERE)
- Multi-step aggregations (derived tables)
- Data quality checks (INTERSECT/EXCEPT for reconciliation)
- Dynamic filtering based on aggregates (subqueries in HAVING)

Formula Quick Reference

UNION Pattern:

```
SELECT cols FROM table1
UNION ALL
SELECT cols FROM table2
```

ORDER BY col;

Subquery in WHERE:

WHERE column > (SELECT AVG(column) FROM table)

EXISTS Pattern:

WHERE EXISTS (SELECT 1 FROM table WHERE id = outer.id)

Derived Table:

FROM (SELECT ... GROUP BY ...) AS alias

Find Missing:

WHERE NOT EXISTS (SELECT 1 FROM related WHERE id = main.id)