SQL Quick Revision Notes

Quick Revision Notes · by Mahendra Singh · CrackAnalytics

Order of execution (how SQL actually runs)

FROM → JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT. This is why you can't use a SELECT alias inside WHERE, but can inside ORDER BY.

Joins in one line each

- **INNER** — only matching rows from both tables
- **LEFT** — all left rows + matches (NULL where no match)
- **RIGHT** — all right rows + matches
- **FULL** — everything from both sides
- **CROSS** — every combination (cartesian)
- **SELF** — table joined to itself (employee–manager)

Trap: duplicate join keys multiply rows — x(1,1,1,1,1) join z(1,1,1) = 15 rows for ALL join types.

Window functions (the interview favourites)

```
ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary DESC)  -- 1,2,3 unique
RANK()       -- 1,1,3  (ties share, gap after)
DENSE_RANK() -- 1,1,2  (ties share, no gap)
LAG(x)/LEAD(x) -- previous / next row value
SUM(x) OVER (ORDER BY dt ROWS UNBOUNDED PRECEDING) -- running total
```

Must-know query patterns

2nd highest salary:

```
SELECT MAX(salary) FROM emp
WHERE salary < (SELECT MAX(salary) FROM emp);
```

Nth highest (window):

```
SELECT salary FROM (SELECT salary,
  DENSE_RANK() OVER (ORDER BY salary DESC) rnk FROM emp) t
WHERE rnk = N;
```

Duplicates:

```
SELECT email, COUNT(*) FROM c GROUP BY email HAVING COUNT(*)>1;
```

Delete duplicates keeping one:

```
WITH r AS (SELECT id, ROW_NUMBER() OVER
 (PARTITION BY email ORDER BY id) rn FROM c)
DELETE FROM c WHERE id IN (SELECT id FROM r WHERE rn>1);
```

Differences asked in every interview

- **WHERE vs HAVING** — rows before grouping vs groups after GROUP BY
- **UNION vs UNION ALL** — dedup (slow) vs keep all (fast)
- **DELETE vs TRUNCATE vs DROP** — rows with WHERE / all rows fast / whole table
- **CTE vs View** — one query, temporary vs saved reusable object
- **Clustered vs Non-clustered index** — physical row order (1/table) vs separate pointer structure (many)
- **COALESCE vs ISNULL** — many args, standard vs 2 args, SQL Server only

NULL rules

- NULL = absence of value; never compare with '=' — use IS NULL / IS NOT NULL
- Aggregates ignore NULL, but COUNT(*) counts every row
- NOT IN with a NULL in the list returns NOTHING — classic trap
- Count nulls: COUNT(CASE WHEN col IS NULL THEN 1 END)

Query optimization checklist

- Index WHERE/JOIN/ORDER BY columns · read EXPLAIN plan · filter early
- Never SELECT * in production · prefer set-based over cursors/loops
- For huge joins: partitioning, bucketing, broadcast small tables