CHEATSHEET

Aggregations & GROUP BY Cheatsheet

COUNT, SUM, AVG and grouping logic, explained with results

6
Aggregate fns

1
WHERE vs HAVING

2
Pages

The core aggregate functions

Function	Returns
`COUNT(*)`	Number of rows (including NULLs)
`COUNT(col)`	Number of non-NULL values in col
`COUNT(DISTINCT col)`	Number of unique non-NULL values
`SUM(col)`	Total of a numeric column
`AVG(col)`	Mean of a numeric column
`MIN(col) / MAX(col)`	Smallest / largest value

GROUP BY — collapsing rows into buckets

```sql
SELECT department, COUNT(*) AS headcount, AVG(salary) AS avg_salary
FROM employees
GROUP BY department
ORDER BY avg_salary DESC;
```

- Every non-aggregated column in SELECT must appear in GROUP BY (strict SQL / PostgreSQL / SQL Server).
- MySQL is lenient about this by default — don't rely on it, the result is not guaranteed.
- You can GROUP BY multiple columns: GROUP BY department, job_title.

WHERE vs. HAVING — the #1 interview trap

```sql
SELECT department, COUNT(*) AS headcount
FROM employees
WHERE status = 'active'        -- filters ROWS, before grouping
GROUP BY department
HAVING COUNT(*) > 5;           -- filters GROUPS, after aggregating
```

	WHERE	HAVING
`Filters`	Individual rows	Aggregated groups

	WHERE	HAVING
Runs	Before GROUP BY	After GROUP BY
Can use aggregates?	No	Yes

RULE OF THUMB
If your condition mentions COUNT(), SUM(), AVG() etc., it belongs in HAVING. Everything else belongs in WHERE — filtering early is also faster.

NULL handling in aggregates

- Every aggregate function except COUNT(*) silently ignores NULL values.
- AVG(col) divides by the count of non-NULL rows, not the total row count — a common source of bugs.
- Use COALESCE(col, 0) before aggregating if you need NULLs treated as zero.

GROUPING SETS, ROLLUP, CUBE (advanced)

```
SELECT department, job_title, SUM(salary)
FROM employees
GROUP BY ROLLUP (department, job_title);
-- Adds subtotal rows per department, plus a grand total row
```