SQL Intermediate Command Reference (Interview-Ready)

This document covers **INTERMEDIATE SQL**, which forms the **core of Data Analyst / Data Scientist / Backend interviews**. These topics test your ability to **solve real business problems**, not just write syntax.

---

1. AGGREGATE FUNCTIONS (Core Analytical SQL)

Definition

Aggregate functions perform calculations on multiple rows and return a **single summarized value**.

Common Aggregate Functions

```
COUNT(column)
SUM(column)
AVG(column)
MIN(column)
MAX(column)
```

Examples

```
SELECT COUNT(*) FROM employees;
SELECT AVG(salary) FROM employees;
SELECT MAX(salary) FROM employees;
```

📌Interview Tip: `COUNT(*)` counts rows, while `COUNT(column)` ignores NULLs.

---

2. GROUP BY

Definition

`GROUP BY` groups rows that have the same values in specified columns and is used **with aggregate functions**.

Syntax

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department;
```

📌Rule: Every column in SELECT must either be in GROUP BY or be an aggregate.

---

3. HAVING vs WHERE

WHERE	HAVING
Filters rows	Filters groups
Used before GROUP BY	Used after GROUP BY
Cannot use aggregates	Can use aggregates

Example

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) > 60000;
```

---

4. JOINS (MOST IMPORTANT 🔥)

4.1 INNER JOIN

Returns matching records from both tables.

```
SELECT e.name, d.dept_name
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.dept_id;
```

4.2 LEFT JOIN

Returns all rows from left table and matching rows from right table.

```sql
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d
ON e.dept_id = d.dept_id;
```

4.3 RIGHT JOIN

Returns all rows from right table.

```sql
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d
ON e.dept_id = d.dept_id;
```

4.4 FULL OUTER JOIN (Conceptual in MySQL)

Returns all records when there is a match in either table.

4.5 SELF JOIN

A table joined with itself.

```sql
SELECT e1.name AS employee, e2.name AS manager
FROM employees e1
JOIN employees e2
ON e1.manager_id = e2.emp_id;
```

---

5. SUBQUERIES

Definition

A query nested inside another query.

5.1 Single-row Subquery

```
SELECT *
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

5.2 Multi-row Subquery

```
SELECT *
FROM employees
WHERE dept_id IN (SELECT dept_id FROM departments WHERE location = 'Pune');
```

5.3 EXISTS

```
SELECT name
FROM employees e
WHERE EXISTS (
  SELECT 1 FROM departments d WHERE d.dept_id = e.dept_id
);
```

---

6. STRING FUNCTIONS

```
CONCAT(first_name, ' ', last_name)
UPPER(name)
LOWER(name)
TRIM(name)
SUBSTRING(name, 1, 4)
LENGTH(name)
REPLACE(name, 'a', 'A')
```

---

7. DATE & TIME FUNCTIONS

```
NOW()
CURDATE()
DATEDIFF(date1, date2)
```

```
DATE_ADD(date, INTERVAL 7 DAY)
DATE_SUB(date, INTERVAL 1 MONTH)
YEAR(date)
MONTH(date)
```

---

8. CONDITIONAL LOGIC

CASE WHEN

```
SELECT name,
CASE
  WHEN salary >= 80000 THEN 'High'
  WHEN salary >= 50000 THEN 'Medium'
  ELSE 'Low'
END AS salary_band
FROM employees;
```

---

9. CONVERSION & SCALAR FUNCTIONS

CAST / CONVERT

```
SELECT CAST(salary AS CHAR) FROM employees;
SELECT CONVERT('2026-01-01', DATE);
```

NULL Handling

```
COALESCE(salary, 0)
NULLIF(a, b)
```

---

10. SET OPERATORS (INTERMEDIATE USE)

```
SELECT city FROM customers
UNION ALL
SELECT city FROM suppliers;
```

- UNION → removes duplicates
- UNION ALL → keeps duplicates

INTERVIEW SUMMARY (VERY IMPORTANT)

- Aggregates + GROUP BY → Analytics
- Joins → Data relationships
- Subqueries → Logical depth
- CASE → Business logic
- Date & string functions → Real-world data

🔔**Mastering this section makes you interview-ready for most SQL roles.**