SQL A-to-Z Handbook

Your ultimate guide to mastering SQL – designed for quick revision even after a year.

Table of Contents

1. Basics & CRUD
2. Filtering & Operators
3. Sorting & Limiting
4. Aggregates, GROUP BY, HAVING
5. Joins
6. Subqueries
7. Set Operations
8. Date/Time Functions
9. Window Functions
10. Case Statements
11. CTEs
12. Transactions
13. Indexes, Keys, Constraints
14. NULL Handling
15. Interview SQL Tips

■ Basics & CRUD

SQL is used for CRUD operations: Create, Read, Update, Delete. These are the foundation of database interactions.
Syntax:
```
SELECT * FROM table; INSERT INTO table (col1, col2) VALUES (val1, val2); UPDATE table SET col = val WHERE condition; DELETE FROM table WHERE condition;
```

Example:
```
SELECT name, salary FROM Employees; UPDATE Employees SET salary = 60000 WHERE id = 1;
```

Result: Retrieve all employees' names and salaries; update one employee's salary.

■ Filtering & Operators

Use WHERE with operators to filter rows.
Syntax:
```
SELECT * FROM Employees WHERE salary > 50000;
```

Example:
```
SELECT name FROM Employees WHERE department IN ('HR','IT') AND salary BETWEEN 40000 AND 70000;
```

Result: Employees in HR/IT with salary between 40k and 70k.

■ Sorting & Limiting

ORDER BY sorts results; LIMIT/TOP restricts rows.
Syntax:
```
SELECT * FROM Employees ORDER BY salary DESC LIMIT 5;
```

Example:
```
SELECT TOP 5 name, salary FROM Employees ORDER BY salary DESC;
```

Result: Top 5 employees by salary.

■ Aggregates, GROUP BY, HAVING

Aggregate functions summarize data. GROUP BY groups rows, HAVING filters groups.
Syntax:
```
SELECT department, AVG(salary) FROM Employees GROUP BY department HAVING AVG(salary) > 50000;
```

Example:
```
SELECT department, COUNT(*) AS num FROM Employees GROUP BY department;
```

Result: Departments with avg salary > 50k and their employee counts.

■ Joins

Joins combine rows from different tables based on a condition.
Syntax:
```
SELECT e.name, d.dept_name FROM Employees e JOIN Department d ON e.dept_id = d.id;
```

Example:
```
SELECT e.name, m.name AS manager FROM Employees e JOIN Employees m ON e.managerId = m.id;
```

Result: Employees matched with their departments and managers.

■ Subqueries

Subqueries are queries inside other queries, used for filtering, comparison, or as virtual tables.
Syntax:
```
SELECT name FROM Employees WHERE salary > (SELECT AVG(salary) FROM Employees);
```

Example:
```
SELECT name FROM Employees WHERE dept_id IN (SELECT id FROM Department WHERE location = 'NY');
```

Result: Employees with salary above average or working in NY.

■ Set Operations

Set operations combine multiple result sets.
Syntax:
```
SELECT name FROM Customers UNION SELECT name FROM Suppliers;
```

Example:
```
SELECT name FROM TableA INTERSECT SELECT name FROM TableB;
```

Result: All unique names from Customers and Suppliers; common names in A and B.

■ Date/Time Functions

Date functions are key in real-world queries for reports and filtering.
Syntax:
```
SELECT CURRENT_DATE, NOW(); SELECT DATEDIFF(CURDATE(), hire_date) FROM Employees;
```

Example:
```
SELECT DATE_ADD(hire_date, INTERVAL 30 DAY) FROM Employees;
```

Result: Show today's date, days worked, or add 30 days to hire_date.

■ Window Functions

Window functions perform calculations across related rows without collapsing results.
Syntax:
```
SELECT name, salary, ROW_NUMBER() OVER (ORDER BY salary DESC) FROM Employees;
```

Example:
```
SELECT department, name, RANK() OVER (PARTITION BY department ORDER BY salary DESC) FROM
Employees;
```

Result: Ranks employees overall or within departments.

■ Case Statements

CASE applies conditional logic inside queries.
Syntax:
```
SELECT name, CASE WHEN salary >= 100000 THEN 'High' ELSE 'Low' END FROM Employees;
```

Example:
```
SELECT name, CASE WHEN salary > 80000 THEN 'High' WHEN salary > 50000 THEN 'Medium' ELSE 'Low'
END FROM Employees;
```

Result: Categorize employees into High/Medium/Low salary bands.

■ CTEs

CTEs (WITH clauses) make queries more readable and reusable.
Syntax:
```
WITH HighEarners AS (SELECT name FROM Employees WHERE salary > 80000) SELECT * FROM HighEarners;
```

Example:
```
WITH DeptCount AS (SELECT dept_id, COUNT(*) AS cnt FROM Employees GROUP BY dept_id) SELECT * FROM DeptCount WHERE cnt > 5;
```

Result: Reusable subqueries as temporary tables.

■ Transactions

Transactions group statements into an atomic unit – either all succeed or none.
Syntax:
```
BEGIN; UPDATE Accounts SET balance = balance - 100 WHERE id = 1; UPDATE Accounts SET balance = balance + 100 WHERE id = 2; COMMIT;
```

Example:
```
BEGIN; ... ROLLBACK;
```

Result: Ensures consistent account balances during transfer.

■ Indexes, Keys, Constraints

Indexes speed up queries, keys maintain integrity, constraints ensure rules.
Syntax:
```
PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, INDEX
```

Example:
```
CREATE INDEX idx_name ON Employees(name);
```

Result: Faster search on name column.

■ NULL Handling

NULL means missing/unknown value. Use IS NULL, COALESCE, NULLIF to handle it.
Syntax:
```
SELECT * FROM Employees WHERE managerId IS NULL;
```

Example:
```
SELECT COALESCE(phone, 'N/A') FROM Employees;
```

Result: Find employees without manager; replace null phone with 'N/A'.

■ Interview SQL Tips

1. **Start simple**: Write a query that fetches the needed rows, even if it has duplicates or extra columns. Then refine.
2. **Think JOIN vs Subquery**: If relating two tables → try JOIN. If filtering → try subquery.
3. **Check NULLs**: Many interview traps involve NULL handling.
4. **GROUP BY + HAVING**: Always confirm whether the condition is row-level (WHERE) or group-level (HAVING).
5. **Window functions**: If you need ranking, running totals, or 'previous/next row' logic → use them.

6. **Debug step by step**: Run the subquery alone, or the join without filters, to verify partial results.
7. **Order of execution**: FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY. Useful for troubleshooting.
8. **Be clear**: Use aliases, CTEs, and indentation. In interviews, clarity matters as much as correctness.