SQL Cheat Sheet

A quick reference for everyday SQL — syntax, joins, functions, and patterns.

Author: Digamber Jha • **Coding India**

1. Querying Data

The full logical order of a SELECT: FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT.

```
SELECT col1, col2
FROM   table_name
WHERE  condition
ORDER  BY col1 DESC
LIMIT  10;
```

Common shortcuts:

Clause	Purpose
SELECT *	Return every column (avoid in production queries)
DISTINCT	Remove duplicate rows from the result
AS alias	Rename a column or table in output
LIMIT n	Cap rows returned (Postgres/MySQL); use TOP n on SQL Server
OFFSET n	Skip the first n rows — used for pagination

2. Filtering — WHERE

Operator	Matches
= <> < > <= >=	Standard comparisons (<> and != both mean not-equal)
AND / OR / NOT	Combine or negate conditions
BETWEEN a AND b	Value within an inclusive range
IN (v1, v2, ...)	Value matches any item in the list
LIKE 'a%'	Pattern match: % = any string, _ = single char
IS NULL	Test for missing value (never use = NULL)

```
SELECT name, salary
FROM   employees
WHERE  department = 'Sales'
  AND  salary BETWEEN 40000 AND 90000
  AND  name LIKE 'A%';
```

3. Joins

Combine rows from two or more tables on a related column.

Join Type	Returns
INNER JOIN	Only rows with a match in both tables
LEFT JOIN	All left rows + matched right rows (NULLs if none)
RIGHT JOIN	All right rows + matched left rows
FULL OUTER JOIN	All rows from both, matched where possible
CROSS JOIN	Every combination (Cartesian product)
SELF JOIN	Table joined to itself via aliases

```
SELECT o.id, c.name, o.amount
FROM   orders o
JOIN   customers c ON c.id = o.customer_id
LEFT   JOIN refunds r ON r.order_id = o.id
WHERE  o.amount > 100;
```

4. Aggregation & Grouping

Function	Result
COUNT(*)	Number of rows
COUNT(col)	Non-NULL values in a column
SUM(col)	Total of numeric values
AVG(col)	Mean value
MIN / MAX(col)	Smallest / largest value

Filter groups with HAVING (WHERE filters rows before grouping; HAVING filters after).

```
SELECT   department, COUNT(*) AS headcount, AVG(salary) AS avg_pay
FROM     employees
GROUP BY department
HAVING   COUNT(*) > 5
ORDER BY avg_pay DESC;
```

5. Subqueries & CTEs

Subquery

```
SELECT name, salary
FROM   employees
WHERE  salary > (SELECT AVG(salary) FROM employees);
```

Common Table Expression (CTE) — cleaner for complex logic

```
WITH top_earners AS (
   SELECT department, MAX(salary) AS max_pay
   FROM   employees
   GROUP  BY department
)
SELECT e.name, e.department
FROM   employees e
JOIN   top_earners t
  ON   t.department = e.department AND e.salary = t.max_pay;
```

6. Set Operations

Operator	Behaviour
UNION	Combine results, remove duplicates
UNION ALL	Combine results, keep duplicates (faster)
INTERSECT	Rows present in both queries
EXCEPT / MINUS	Rows in the first query but not the second

Each query must return the same number of columns with compatible types.

7. Modifying Data (DML)

Insert

```
INSERT INTO employees (name, department, salary)
VALUES ('Asha', 'Sales', 55000),
       ('Ravi', 'Tech',  72000);
```

Update

```
UPDATE employees
SET    salary = salary * 1.10
WHERE  department = 'Tech';
```

Delete

```
DELETE FROM employees
WHERE  id = 42;
```

Always pair UPDATE and DELETE with a WHERE **clause — without one, every row is affected.**

8. Defining Structure (DDL)

```
CREATE TABLE employees (
   id         INT PRIMARY KEY,
   name       VARCHAR(100) NOT NULL,
   department VARCHAR(50),
   salary     DECIMAL(10,2) DEFAULT 0,
   hired_on   DATE
);

ALTER TABLE employees ADD COLUMN email VARCHAR(120);
ALTER TABLE employees DROP COLUMN email;

DROP TABLE employees;      -- removes table + data
TRUNCATE TABLE employees;  -- empties table, keeps structure
```

9. Constraints

Constraint	Enforces
PRIMARY KEY	Unique, non-NULL row identifier
FOREIGN KEY	Value must exist in another table
UNIQUE	No duplicate values in the column
NOT NULL	Column must have a value
CHECK (expr)	Value must satisfy a condition
DEFAULT val	Value used when none is supplied

10. Built-in Functions

String

Function	Does
UPPER / LOWER(s)	Change case
LENGTH(s)	Character count
TRIM(s)	Strip leading/trailing spaces
SUBSTRING(s,p,n)	Extract n chars from position p

Function	Does
CONCAT(a,b)	Join strings (or a || b)
REPLACE(s,x,y)	Swap all x for y

Date

Function	Does
NOW() / CURRENT_DATE	Current timestamp / date
EXTRACT(YEAR FROM d)	Pull a part out of a date
DATE_ADD / INTERVAL	Shift a date by a duration
DATEDIFF(a,b)	Days between two dates

Numeric & NULL handling

Function	Does
ROUND(n,d)	Round to d decimal places
CEIL / FLOOR(n)	Round up / down to integer
ABS(n) / MOD(a,b)	Absolute value / remainder
COALESCE(a,b,...)	First non-NULL argument
NULLIF(a,b)	NULL if a = b, else a

11. Conditional Logic — CASE

```
SELECT name,
    salary,
    CASE
      WHEN salary >= 80000 THEN 'High'
      WHEN salary >= 50000 THEN 'Mid'
      ELSE 'Entry'
    END AS band
FROM   employees;
```

12. Window Functions

Compute across a set of rows without collapsing them, using OVER (PARTITION BY ... ORDER BY ...).

Function	Gives per window
ROW_NUMBER()	Sequential number, no ties
RANK()	Rank with gaps on ties
DENSE_RANK()	Rank with no gaps on ties
LAG / LEAD(col)	Previous / next row's value
SUM/AVG(col) OVER	Running or partitioned aggregate

```
SELECT name, department, salary,
    RANK() OVER (PARTITION BY department
                 ORDER BY salary DESC) AS dept_rank
FROM   employees;
```

13. Indexes, Views & Transactions

```
-- Speed up lookups on a column
CREATE INDEX idx_emp_dept ON employees(department);
```

```
-- Saved query you can treat like a table
CREATE VIEW sales_team AS
SELECT name, salary FROM employees WHERE department = 'Sales';

-- All-or-nothing block
BEGIN;
  UPDATE accounts SET balance = balance - 100 WHERE id = 1;
  UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;   -- or ROLLBACK to undo
```

14. Practical Tips

- **Name columns explicitly** — instead of SELECT * — it's faster and won't break when the schema changes.
- **Filter early** — put restrictive conditions in WHERE so joins process fewer rows.
- **Comparisons with NULL** — always yield unknown — use IS NULL / IS NOT NULL, not = or <>.
- **Index the columns** — you join and filter on most, but don't over-index write-heavy tables.
- **Test destructive queries** — run the WHERE as a SELECT first before UPDATE or DELETE.