Ultimate SQL Cheat Sheet

CORE CONCEPTS

Relational basics

- **Database:** named collection of tables
- **Table:** rows and columns
- **Row:** one record
- **Column:** named attribute with a data type
- **Primary key:** unique, not null identifier
- **Foreign key:** column that references a primary key in another table

Keys

- **Candidate key:** any column set that can be primary key
- **Composite key:** key with several columns
- **Surrogate key:** artificial key with no business meaning

Null

- **Means unknown or missing**
- **Use IS NULL or IS NOT NULL**
- **Most aggregate functions skip null values.**
- **COUNT(column) skips null values**
- **COUNT(*) counts every row.**

DATA TYPES

Numeric

- **INT, BIGINT, SMALLINT**
- **DECIMAL or NUMERIC**
- **FLOAT, REAL, DOUBLE**

Text

- **CHAR(n) fixed length**
- **VARCHAR(n) variable length**
- **TEXT long text**

Date and Time

- **DATE**
- **TIME**
- **TIMESTAMP or DATETIME**

Other

- **BOOLEAN**
- **BLOB or BYTEA**
- **JSON, JSONB where available**
- **UUID type in many systems**

QUERY ORDER

```
SELECT [DISTINCT] col_list
FROM   table_or_joins
WHERE  row_condition
GROUP  BY group_cols
HAVING group_condition
ORDER  BY sort_cols
LIMIT n OFFSET m
```

Example

```
SELECT first_name, last_name
FROM employees
WHERE department = 'Sales'
ORDER BY last_name, first_name;
```

HANDY UTILITY FUNCTIONS

LENGTH()
Gives you how many characters are in a string.

CAST()
Changes a value from one data type to another type you choose.

NOW()
Returns the current date and time from the database server.

CEILING()
Rounds a number up to the nearest whole number.

FLOOR()
Rounds a number down to the nearest whole number.

TRIM()
Removes extra spaces from the start and end of a string.

CONCAT()
Joins two or more strings together into one.

COALESCE()
Returns the first value in the list that is not NULL.

FILTERING & CONDITIONS

Comparisons

=, <, >, <=, >=, <>

```
SELECT * FROM
...
```

Patterns

```
WHERE name LIKE 'A%'      -- starts with A
WHERE name LIKE '%son'    -- ends with son
WHERE name LIKE '%car%'   -- contains car
```

Sets and ranges

```
WHERE country IN ('US', 'CA', 'MX')
WHERE salary BETWEEN 50000 AND 80000
```

Null checks

```
WHERE deleted_at IS NULL
WHERE middle_name IS NOT NULL
```

Boolean logic

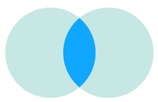
AND before OR in precedence, use parentheses:

```
WHERE status = 'Active'
AND (country = 'US' OR country = 'CA');
```

JOINS

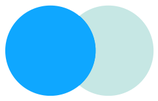
INNER JOIN: Combines rows from two or more tables based on a related column between them

```
SELECT e.employee_id, e.first_name, d.name AS
department_name
FROM employees e
JOIN departments d
ON e.department_id = d.department_id;
```



LEFT JOIN: Returns all rows from the left table and the matching rows from the right table

```
SELECT e.employee_id, e.first_name, d.name AS
department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;
```



SIMPLE SELECT QUERIES

Returns only users whose status is exactly *Active*. Good for basic filtering by a single column value.

```
SELECT user_id, first_name, last_name
FROM users
WHERE status = 'Active';
```

Returns all employees who belong to department 10. Classic lookup by foreign key.

```
SELECT employee_id, first_name, last_name
FROM  employees
WHERE department_id = 10;
```

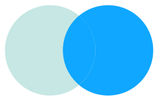
Returns products with prices between 10 and 50. Shows how to combine conditions with AND.

```
SELECT product_id, name, price
FROM products
WHERE price >= 10
 AND price <= 50;
```

MORE JOINS

RIGHT JOIN: Returns all records from the right-hand table in a join, and the matching records from the left-hand table

```
SELECT
  e.emp_id,
  e.emp_name,
  d.dept_id,
  d.dept_name
FROM Employees AS e
RIGHT JOIN Departments AS d
ON e.dept_id = d.dept_id;
```



FULL OUTER JOIN: Returns all rows when there is a match in either the left or right table

```
SELECT
  e.emp_id,
  e.emp_name,
  d.dept_id,
  d.dept_name
FROM Employees AS e
FULL OUTER JOIN Departments AS d
ON e.dept_id = d.dept_id;
```

SELECT WITH ORDER BY

Returns all users with results sorted alphabetically by last name from A to Z.

```
SELECT user_id, first_name, last_name
FROM users
ORDER BY last_name ASC;
```

Returns all products with the priciest items at the top and the cheapest at the bottom.

```
SELECT product_id, name, price
FROM  products
ORDER BY price DESC;
```

Sorts employees by department and, within each department, by last name.

```
SELECT employee_id, first_name, last_name,
department_id
FROM employees
ORDER BY department_id ASC, last_name ASC;
```

ALIASES

An alias is a temporary name you give to a column or table in a query using the AS keyword.

This query renames first_name to fname and last_name to lname in the result set

```
SELECT first_name AS fname,
last_name AS lname
FROM employees;
```

Here users is given the table alias u. Inside the query, you can refer to columns as u.user_id, u.first_name, and so on

```
SELECT u.user_id,
    u.first_name,
    u.last_name
FROM users AS u
WHERE u.status = 'Active';
```

BETWEEN

BETWEEN is a comparison operator that checks whether a value falls within a given range, including both endpoints

```
SELECT *
FROM orders
WHERE order_date BETWEEN '2025-01-01' AND
'2025-01-31';
```

NOTE: BETWEEN includes both endpoints. For timestamps, many teams prefer a pattern like >= '2025-01-01' and < '2025-02-01'.

LIKE

LIKE is used in a WHERE clause to match text patterns.

Common wildcards:

% matches zero or more characters
_ matches exactly one character

Finds all customers whose name begins with the letter A.

```
SELECT customer_id,
full_name
FROM customers
WHERE full_name LIKE 'A%';
```

Returns users whose email ends with @gmail.com.

```
SELECT user_id, email
FROM users
WHERE email LIKE
'%@gmail.com';
```

Finds products whose name contains the word “phone” anywhere.

```
SELECT product_id,
product_name
FROM products
WHERE product_name LIKE
'%phone%';
```

NOT & NULL

Returns all rows where diagnosis_code has a value and is not missing.

```
SELECT user_id,
    diagnosis_code
FROM claims
WHERE diagnosis_code IS NOT
NULL;
```

Returns all users whose status is anything other than Inactive.

```
SELECT user_id,
    email,
    status
FROM users
WHERE NOT status = 'Inactive';
```

Ultimate SQL Cheat Sheet

CTEs

A common table expression (CTE) is a temporary named result set that you define with WITH and use in a query that follows it. You can think of it as a short lived table that exists only for that single query and makes complex logic easier to read and reuse.

The CTE dept_counts computes employees per department once, then the main query filters to departments with at least ten employees.

```
WITH dept_counts AS (
SELECT department_id,
COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
)
SELECT department_id,
employee_count
FROM dept_counts
WHERE employee_count >= 10;
```

The CTE ranked_salaries assigns a row number based on salary. The outer query then picks the five highest paid employees.

```
WITH ranked_salaries AS (
  SELECT employee_id,
      first_name,
      last_name,
      salary,
      ROW_NUMBER() OVER (ORDER BY salary DESC) AS rn
  FROM  employees
)
SELECT employee_id,
    first_name,
    last_name,
    salary
FROM  ranked_salaries
WHERE  rn <= 5;
```

SUBQUERY

A subquery is a query nested inside another SQL statement (for example in SELECT, FROM, or WHERE). The outer query uses the result of the inner query as if it were a value, a table, or a condition.

The inner query calculates the average salary. The outer query then returns only employees whose salary is higher than that average.

```
SELECT employee_id,
    first_name,
    last_name,
    salary
FROM  employees
WHERE salary > (
  SELECT AVG(salary)
  FROM  employees
);
```

The subquery creates a temporary result with one row per department and its employee count. The outer query filters that result to departments with at least five employees.

```
SELECT d.department_id,
    d.employee_count
FROM (
  SELECT department_id,
      COUNT(*) AS employee_count
FROM employees
GROUP BY department_id
) AS d
WHERE d.employee_count >= 5;
```

CASE EXPRESSIONS

A CASE expression lets you create new values in a query based on conditions. It checks each condition in order and returns the result for the first condition that is true, otherwise it returns an optional ELSE value.

```
SELECT employee_id,
    first_name,
    last_name,
    salary,
    CASE
      WHEN salary >= 100000 THEN 'High'
      WHEN salary >= 60000  THEN 'Medium'
      ELSE 'Low'
    END AS salary_band
FROM employees;
```

This query adds a new column called salary_band. Each employee is labeled High, Medium, or Low depending on their salary.

PARTITION BY

PARTITION BY is used inside a window function to split rows into groups. The function is then calculated separately inside each group while still returning one row per input row.

```
SELECT department_id,
    employee_id,
    first_name,
    salary,
    AVG(salary) OVER (
     PARTITION BY department_id
    ) AS dept_avg_salary
FROM employees;
```

UNION vs UNION ALL

UNION

Combines the results of two queries and removes duplicate rows from the combined result set. Columns and data types must line up in both queries.

```
SELECT customer_id, full_name
FROM online_customers
UNION
SELECT customer_id, full_name
FROM store_customers;
```

UNION ALL

Combines the results of two queries and keeps every row, including duplicates. Usually faster because the database does not check for duplicates

```
SELECT customer_id, full_name
FROM online_customers
UNION ALL
SELECT customer_id, full_name
FROM store_customers;
```

RANK

RANK is a window function that gives each row a position within an ordered group. Rows with the same sort value receive the same rank, and the next rank number jumps after ties. It works together with OVER, PARTITION BY, and ORDER BY

```
SELECT department_id,
    employee_id,
    first_name,
    salary,
    RANK() OVER (
     PARTITION BY department_id
     ORDER BY salary DESC
    ) AS salary_rank_in_dept
FROM employees;
```