SQL CHEATSHEET

By ABHINAY MAHATO

WHAT IS SQL?

SQL stands for Structured Query Language. It is a declarative query language designed to work with relational databases. It enables users to access, manipulate, and control data stored in databases.

TYPES OF SQL COMMANDS

1. DDL - Data Definition Language
2. DML - Data Manipulation Language
3. DQL - Data Query Language
4. DCL - Data Control Language
5. TCL - Transaction Control Language

1. DDL - DATA DEFINITION LANGUAGE

Used to define or modify database structure.
Examples: CREATE, ALTER, DROP, TRUNCATE

- CREATE - create database objects
- ALTER - modify structure
- DROP - delete objects
- TRUNCATE - remove all rows (faster than DELETE)
- RENAME - rename objects

EXAMPLE:

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    salary INT
);
```

2. DML - DATA MANIPULATION LANGUAGE

Used to insert, update, or delete data in tables.
Examples: INSERT, UPDATE, DELETE

- INSERT - add data
- UPDATE - modify data
- DELETE - remove data

EXAMPLE: UPDATE employees
SET salary = 60000
WHERE id = 1;

3. DQL - DATA QUERY LANGUAGE

- Used to query and retrieve data.
  Example: SELECT

- SELECT - fetch data from tables

- EXAMPLE: SELECT * FROM employees;

4. DCL - DATA CONTROL LANGUAGE

- Used to control access and permissions.
  Examples: GRANT, REVOKE

- GRANT - give permissions
  REVOKE - remove permissions

- EXAMPLE: GRANT SELECT, INSERT
  ON employees
  TO user1;

5. TCL - TRANSACTION CONTROL LANGUAGE

- Used to manage transactions and ensure data consistency.
  Examples: COMMIT, ROLLBACK, SAVEPOINT

  COMMIT - save changes
  ROLLBACK - undo changes
  SAVEPOINT - create checkpoint in transaction

- EXAMPLE:

```
UPDATE employees
SET salary = 60000
WHERE id = 101;

COMMIT;
```

ESSENTIAL SQL OPERATORS

=	=>	EQUAL	=>	SALARY=5000
!=/<>	=>	NOT EQUAL	=>	AGE<>30
>,<,>=,<=	=>	COMPARISON	=>	MARKS>75
BETWEEN	=>	RANGE FILTER	=>	AGE BETWEEN 20 AND 40
IN	=>	MATCHES VALUES	=>	DEPT IN ('HR', 'IT')
LIKE	=>	PATTERN MATCH	=>	NAME LIKE 'A%'
IS NULL	=>	CHECK NULL	=>	NAME IS NULL
AND	=>	BOTH CONDITION	=>	A>B AND A>C
OR	=>	AT LEAST ONE	=>	A>B OR A>C
NOT	=>	NEGATION	=>	NOT SAL>3000
+,-.*./,%	=>	MATHEMATICAL CALCULATION ON NUMERIC		

IMPORTANT SQL FUNCTIONS

A. Aggregate Functions

- USED FOR SUMMARIZATION

1. COUNT() => SELECT COUNT(*) FROM customers;
2. SUM() => SELECT SUM(amount) FROM orders;
3. AVG() => SELECT AVG(salary) FROM employees;
4. MIN() => SELECT MIN(age) FROM users;
5 MAX() => SELECT MAX(AGE) FROM USERS;
6. GROUP BY => SELECT department, AVG(salary)
   FROM employees
   GROUP BY department;

IMPORTANT SQL FUNCTIONS

B. Window Functions

- PERFORM CALCULATIONS ACROSS A SET OF RELATED ROWS

1. ROW_NUMBER() => SELECT name, salary,
ROW_NUMBER() OVER (ORDER BY salary DESC) AS rank FROM employees;

2. RANK() / DENSE_RANK() => SELECT name, marks,
RANK() OVER (ORDER BY marks DESC) FROM students;

3. SUM() OVER() => SELECT order_id, amount,
SUM(amount) OVER (ORDER BY order_id) FROM orders;

4. AVG() OVER(PARTITION BY) => SELECT name, dept, salary,
AVG(salary) OVER (PARTITION BY dept) AS avg_dept_sal
FROM employees;

IMPORTANT SQL FUNCTIONS

C. String Functions

- USED TO MANIPULATE, FORMAT, EXTRACT, AND SEARCH TEXTUAL DATA

1. CONCAT() => SELECT CONCAT(first_name, ' ', last_name) AS full_name FROM emp;

2. UPPER() / LOWER() => SELECT UPPER(city) FROM customers;

3. SUBSTRING() => SELECT SUBSTRING(phone, 1, 3) AS country_code FROM customers;

4. LENGTH() => SELECT LENGTH(name) FROM employees;

5. REPLACE() => SELECT REPLACE(address, 'Street', 'St') FROM customers;

IMPORTANT SQL FUNCTIONS

D. Date Functions

- IMPORTANT FOR TIME-SERIES ANALYSIS

1. NOW(), CURRENT_DATE => SELECT CURRENT_DATE;

2. DATEADD() / ADD_MONTHS() => SELECT DATEADD(day, 7, order_date)
FROM orders;

3. DATEDIFF() => SELECT DATEDIFF(day, start_date, end_date) AS days_diff;

4. EXTRACT() => SELECT EXTRACT(YEAR FROM order_date) AS order_year
FROM orders;

JOINS

SQL JOIN clauses are fundamental operations used to combine rows from two or more tables based on a related column between them.

1. INNER JOIN => SELECT *

```
FROM orders o
INNER JOIN customers c ON o.customer_id = c.id;
```

2. LEFT JOIN/RIGHT JOIN => SELECT *

```
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id;
```

3. FULL JOIN => SELECT *

```
FROM
FULL OUTER JOIN B ON A.id = B.id;
```

4. CROSS JOIN => SELECT *

```
FROM products CROSS JOIN colors;
```

SUBQUERY

The primary purpose of a subquery is to return data that the outer (main) query can use to filter results dynamically, perform complex calculations, or act as a temporary table.

1. Subquery in WHERE => SELECT name

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

2. Subquery in FROM => SELECT dept, avg_sal

```
FROM (
   SELECT department AS dept, AVG(salary) AS avg_sal
   FROM employees
   GROUP BY department
) AS t;
```

THANK YOU