SQL Operators

- `=`: Equal to (e.g., `age = 25`)
- `!=` **or** `<>`: Not equal to (e.g., `age != 25`)
- `>`: Greater than (e.g., `age > 25`)
- `>=`: Greater than or equal to (e.g., `age >= 25`)
- `<`: Less than (e.g., `age < 25`)
- `<=`: Less than or equal to (e.g., `age <= 25`)
- `AND`: Combines multiple conditions (e.g., `age > 20 AND city = 'Kuala Lumpur'`)
- `OR`: One of multiple conditions is true (e.g., `age < 18 OR age > 65`)
- `BETWEEN`: Range of values (e.g., `age BETWEEN 20 AND 30`)
- `LIKE`: Pattern matching (e.g., `name LIKE 'A%'`)
    - `%`: Matches any sequence of characters.
    - `_`: Matches a single character.
- `IN`: Matches any value in a list (e.g., `city IN ('KL', 'Penang', 'Johor')`)
- `DISTINCT`: Selects unique values (e.g., `SELECT DISTINCT city FROM customers`)

Basic SQL Commands

1. SELECT

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
SELECT name, age
FROM users
WHERE age >= 18;
```

2. INSERT

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO users (name, age, city)
VALUES ('John Doe', 25, 'KL');
```

3. UPDATE

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example:

```
UPDATE users
SET city = 'Penang'
WHERE name = 'John Doe';
```

4. DELETE

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM users
WHERE age < 18;
```

5. CREATE TABLE

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    ...
);
```

Example:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    city VARCHAR(50)
);
```

6. DROP TABLE

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE users;
```

SQL Joins

1. INNER JOIN

Returns records that have matching values in both tables.

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

2. LEFT JOIN (LEFT OUTER JOIN)

Returns all records from the left table and matched records from the right table.

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

3. RIGHT JOIN (RIGHT OUTER JOIN)

Returns all records from the right table and matched records from the left table.

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;
```

4. FULL OUTER JOIN

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

```
SELECT columns
FROM table1
FULL OUTER JOIN table2
ON table1.column = table2.column;
```

Grouping, Filtering, and Sorting

1. GROUP BY

Groups rows that have the same values into summary rows.

```
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1;
```

Example:

```
SELECT city, COUNT(*)
FROM users
GROUP BY city;
```

2. HAVING

Filters groups based on a condition (used after `GROUP BY`).

```
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1
HAVING COUNT(*) > 1;
```

3. WHERE

Filters rows before grouping or aggregation.

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

4. ORDER BY

Sorts the result set by one or more columns.

```
SELECT columns
FROM table_name
ORDER BY column1 ASC|DESC;
```

Example:

```
SELECT name, age
FROM users
ORDER BY age DESC;
```

Aggregate Functions

1. AVG (Average)

Calculates the average value.

```
SELECT AVG(column_name)
FROM table_name;
```

Example:

```
SELECT AVG(age)
FROM users;
```

2. SUM (Sum)

Calculates the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name;
```

Example:

```
SELECT SUM(salary)
FROM employees;
```

3. COUNT

Counts the number of rows.

```
SELECT COUNT(column_name)
FROM table_name;
```

Example:

```
SELECT COUNT(*)
FROM users;
```

4. MIN (Minimum)

Finds the smallest value.

```
SELECT MIN(column_name)
FROM table_name;
```

Example:

```
SELECT MIN(salary)
FROM employees;
```

5. MAX (Maximum)

Finds the largest value.

```
SELECT MAX(column_name)
FROM table_name;
```

Example:

```
SELECT MAX(age)
FROM users;
```