**1. SELECT \* and Specific Columns**

* The SELECT statement is the foundation of SQL. It is used to **retrieve data** from a table.
* When you write SELECT \*, the \* means **all columns** in the table will be displayed.
* If you only want certain columns, you list them specifically: SELECT column1, column2.

**Theory:**

* **SELECT \*** → Useful when you need to quickly see all the data.
* **Specific columns** → Used when you only care about some attributes, which makes queries faster and results clearer.

**Code :**

**-- Select everything**

**SELECT \***

**FROM employees;**

**-- Select only name and salary**

**SELECT name, salary**

**FROM employees;**

**-- Select only department and city**

**SELECT department, city**

**FROM employees;**

**2. Apply WHERE, AND, OR, LIKE, BETWEEN**

The WHERE clause is used to **filter data**. Without WHERE, SQL returns **all rows**.

**(a) WHERE**

* Adds a **condition** to the query.
* Example: Only employees with salary greater than 50,000.
* Theory: Used to restrict the result set based on specific rules.

Code :

-- Employees with salary greater than 50,000

SELECT name, salary

FROM employees

WHERE salary > 50000;

**(b) AND**

* Combines two or more conditions.
* Both conditions must be true for a row to be selected.
* Theory: Used when you want to apply **multiple strict filters together**.

Code:

-- Employees in HR AND salary greater than 40,000

SELECT name, department, salary

FROM employees

WHERE department = 'HR' AND salary > 40000;

**(c) OR**

* At least one condition must be true.
* Theory: Used when you want results that match **either one condition or another**.

Code :

-- Employees in HR OR IT

SELECT name, department

FROM employees

WHERE department = 'HR' OR department = 'IT';

**(d) LIKE**

* Used for **pattern matching** in text.
* Special characters:
  + % → any sequence of characters.
  + \_ → exactly one character.
* Theory: Used when you don’t know the exact value but know a **pattern** (e.g., names starting with "A").

Code :

Code:

-- Names starting with 'A'

SELECT name

FROM employees

WHERE name LIKE 'A%';

-- Names ending with 'n'

SELECT name

FROM employees

WHERE name LIKE '%n';

-- Names with 5 letters only

SELECT name

FROM employees

WHERE name LIKE '\_\_\_\_\_';

**(e) BETWEEN**

* Selects values within a **range**.
* Includes both boundary values.
* Theory: A convenient way to filter numbers, dates, or other ranges instead of using >= and <=.

Code :

Code :

-- Employees with salary between 40,000 and 60,000

SELECT name, salary

FROM employees

WHERE salary BETWEEN 40000 AND 60000;

**3. Sort with ORDER BY**

* The ORDER BY clause is used to **arrange rows** in a specific order.
* Default is **ascending (ASC)**; use DESC for descending.
* You can sort by one column or multiple columns.

**Theory:**

* **Ascending (ASC)** → smallest to largest, alphabetically A–Z.
* **Descending (DESC)** → largest to smallest, alphabetically Z–A.
* Useful for ranking results (e.g., highest salaries, oldest dates).

Code :

-- Sort employees by salary (lowest to highest)

SELECT name, salary

FROM employees

ORDER BY salary ASC;

-- Sort employees by salary (highest to lowest)

SELECT name, salary

FROM employees

ORDER BY salary DESC;

-- Sort by department (A→Z) and then salary (highest first)

SELECT name, department, salary

FROM employees

ORDER BY department ASC, salary DESC;

**✅ Summary (Theory Only)**

1. **SELECT \* vs. Specific Columns**
   * \* shows everything; specifying columns gives only what you need.
2. **Filtering with WHERE**
   * WHERE applies conditions.
   * AND = both must be true.
   * OR = at least one must be true.
   * LIKE = pattern-based search.
   * BETWEEN = range filtering.
3. **Sorting with ORDER BY**
   * Arranges results in ascending or descending order.
   * Can sort by multiple columns.