Aggregation Function : Summarization fucntion

Step 1: Create Table

```
sql
Copy code
CREATE TABLE employees1 (
    emp_no INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    gender CHAR(1),
    birth_date DATE,
    hire_date DATE,
    salary DECIMAL(10, 2)
);
```

Step 2: Insert Data

```
sql
Copy code
INSERT INTO employees1 (emp_no, first_name, last_name, gende
r, birth_date, hire_date, salary)
VALUES
(1001, 'John', 'Doe', 'M', '1990-05-15', '2020-01-10', 50000.
75),
(1002, 'Alice', 'Smith', 'F', '1985-03-25', '2023-07-19', 600
00.50),
(1003, 'Bob', 'Johnson', 'M', '1993-09-12', '2022-11-30', 550
```

```
00.25),
(1004, 'Emily', 'Davis', 'F', '1988-02-20', '2021-06-15', 580
00.80),
(1005, 'Michael', 'Williams', 'M', '1982-12-05', '2017-11-2
2', 62000.40);
```

Step 3: Use Aggregate Functions

1. **COUNT** – Count total rows in the table.

```
sql
Copy code
SELECT COUNT(emp_no) AS Total_Employees FROM employees1;
```

1. **SUM** – Find the total salary of all employees.

```
SELECT SUM(salary) AS Total_Salary FROM employees1;
```

1. **AVG** – Find the average salary.

```
SELECT AVG(salary) AS Average_Salary FROM employees1;
```

1. **MAX** – Find the highest salary.

```
SELECT MAX(salary) AS Max_Salary FROM employees1;
```

1. **MIN** – Find the lowest salary.

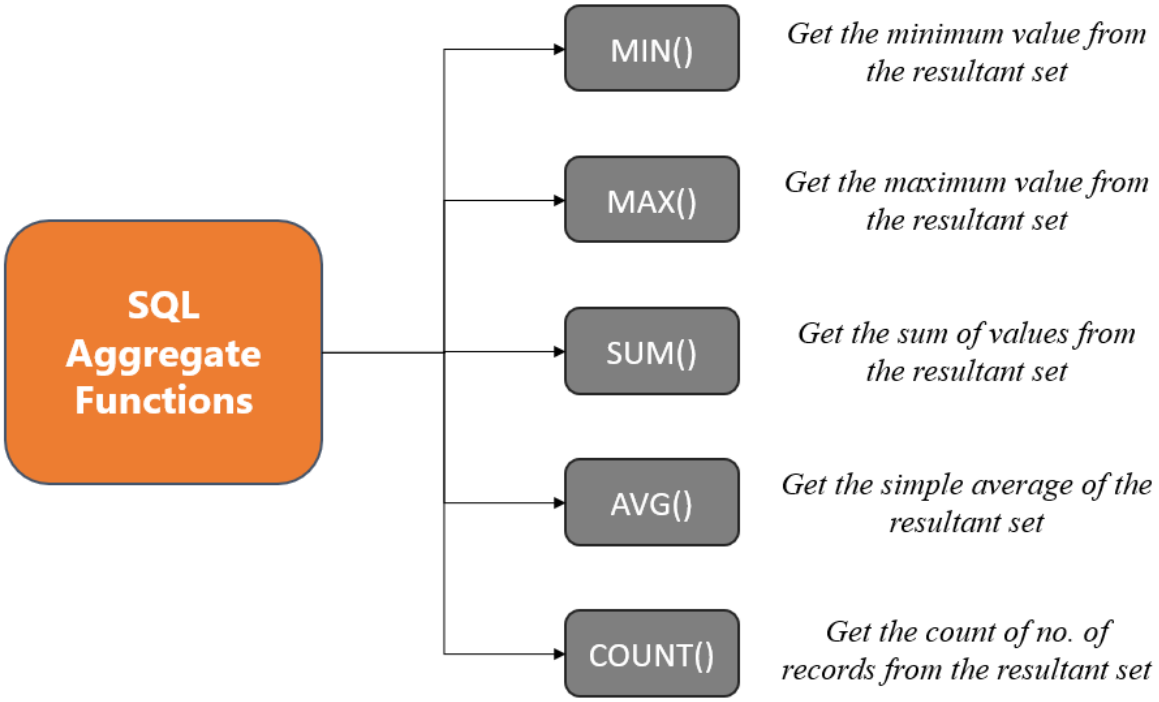
```
SELECT MIN(salary) AS Min_Salary FROM employees1;
```

1. **GROUP BY** – Find the total salary grouped by gender.

```
SELECT gender, SUM(salary) AS Total_Salary_By_Gender
FROM employees1
GROUP BY gender;
```

Why Aggregate Functions are Called Aggregate Functions in SQL:

- **Definition**: Aggregate functions are called "aggregate" because they perform a calculation on a set of values and return a single value, summarizing the data.
- **Purpose**: They help in analyzing and summarizing large volumes of data. Rather than returning individual data points, these functions provide meaningful insights at a higher level, like totals, averages, and counts.



`COUNT()`

- **Purpose**: Counts the number of rows in a set.
- **Usage**: Useful for counting rows, including non-null values.
- **Syntax**:

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

- **Example**:

```
SELECT COUNT(*) FROM employees;  -- Counts all rows
```

`COUNT()`

- **Description**: Counts the number of rows or non-`NULL` values in a specified column.
- **Important Features**:
  - Ignores `NULL` values in the column when counting.
  - `COUNT(*)` counts all rows, including `NULL` values.
  - Used for counting records in a dataset.
- **Example**:

```
SELECT COUNT(*) FROM employees;  -- Counts all rows in the
employees table
```

SUM :

`SUM()`

- **Purpose**: Returns the total sum of a numeric column.
- **Usage**: Typically used for adding up numerical data, such as salaries or sales amounts.
- **Syntax**:

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

- **Example**:

```
SELECT SUM(salary) FROM employees;  -- Total salary of all
employees
```

`SUM()`

- **Description**: Returns the total sum of a numeric column.
- **Important Features**:
  - Only sums up numeric values; ignores `NULL`.
  - Can be used with `GROUP BY` for summing data per group.
  - Doesn't work on non-numeric columns.
- **Example**:

```
SELECT SUM(salary) FROM employees;  -- Total salary of all
employees
```

`AVG()`

- **Description**: Calculates the average (mean) value of a numeric column.
- **Features**:
  - Ignores `NULL` values in the calculation.
  - Returns a floating-point result, even if the column contains integers.
  - Can be used with `GROUP BY` to compute the average per group.
- **Example**:

```
SELECT AVG(salary) FROM employees;  -- Average salary of a
ll employees
```

```
SELECT department_id, AVG(salary) FROM employees GROUP BY
department_id;  -- Average salary per department
```

---

4. `MIN()`

- **Description**: Returns the smallest value in a column.
- **Features**:
  - Works with numeric, text, and date data types.
  - Ignores `NULL` values.
  - Returns the minimum value from a dataset or within groups (if `GROUP BY` is used).
- **Example**:

```
SELECT MIN(salary) FROM employees;  -- Minimum salary of a
ll employees
```

```
SELECT department_id, MIN(salary) FROM employees GROUP BY
department_id;  -- Minimum salary per department
```

---

5. `MAX()`

- **Description**: Returns the largest value in a column.
- **Features**:
  - Works with numeric, text, and date data types.
  - Ignores `NULL` values.

- Returns the maximum value from a dataset or within groups (if `GROUP BY` is used).

- **Example**:

```
SELECT MAX(salary) FROM employees;  -- Maximum salary of a
ll employees
```

```
SELECT department_id, MAX(salary) FROM employees GROUP BY
department_id;  -- Maximum salary per department
```

1. `ROUND()`

- **Description**: Rounds a numeric value to the nearest integer or to a specified number of decimal places.
- **Features**:
  - Can round to a specific number of decimal places.
  - Rounds up or down based on the value of the next digit.
  - Can be used with both numeric and decimal data types.
  - The default rounding is to 0 decimal places (rounds to the nearest whole number).
- **Syntax**:

```
ROUND(number, decimals)
```

- `number`: The numeric value to be rounded.
- `decimals`: The number of decimal places to round to (optional, default is 0).

- **Example**:

```
SELECT ROUND(123.456, 2);  -- Rounds 123.456 to 2 decimal
places, result: 123.46
```

```
SELECT ROUND(123.456);  -- Rounds 123.456 to the nearest w
hole number, result: 123
```

```
SELECT ROUND(123.456, 1);  -- Rounds 123.456 to 1 decimal
place, result: 123.5
```

---

2. `CEILING()`

- **Description**: Rounds a number up to the nearest integer greater than or equal to the given number.
- **Features**:
  - Always rounds up, regardless of the fractional part.
  - Useful for scenarios where you need to round up to the next integer.
  - Works with both positive and negative numbers.
- **Syntax**:

```
CEILING(number)
```

```

```

- `number`: The numeric value to round up.

- **Example**:

```
SELECT CEILING(123.456);  -- Rounds 123.456 up to the next
integer, result: 124
```

```
SELECT CEILING(-123.456);  -- Rounds -123.456 up to the ne
xt integer (less negative), result: -123
```

---

3. `FLOOR()`

- **Description**: Rounds a number down to the nearest integer less than or equal to the given number.
- **Features**:
  - Always rounds down, regardless of the fractional part.
  - Works with both positive and negative numbers.
  - Useful when you need to round down to the nearest whole number.
- **Syntax**:

```
FLOOR(number)
```

  - `number`: The numeric value to round down.

- **Example**:

```
SELECT FLOOR(123.456);  -- Rounds 123.456 down to the next
integer, result: 123
```

```
SELECT FLOOR(-123.456);  -- Rounds -123.456 down to the ne
xt integer (more negative), result: -124
```

---

4. `TRUNCATE()`

- **Description**: Removes the decimal part of a number without rounding it.
- **Features**:
  - Does not round, simply removes any digits after the decimal point.
  - Works similarly to `ROUND()` with 0 decimal places, but it does not round the result.
  - Can be used to truncate a number to a specific number of decimal places.
- **Syntax**:

```
sql
Copy code
TRUNCATE(number, decimals)
```

  - `number`: The numeric value to truncate.
  - `decimals`: The number of decimal places to keep.
- **Example**:

```
sql
Copy code
```

```
SELECT TRUNCATE(123.456, 2);  -- Truncates 123.456 to 2 de
cimal places, result: 123.45
```

```
sql
Copy code
SELECT TRUNCATE(123.456);  -- Truncates 123.456 to the nea
rest integer, result: 123
```

General Features of Rounding Functions:

- **Handling Negative Numbers**: Functions like `CEILING()` and `FLOOR()` behave differently for negative numbers, rounding in opposite directions.
- **Use Cases**:
  - `ROUND()`: When you need to round numbers based on typical rounding rules.
  - `CEILING()`: When you need to round up (e.g., pricing, when you want to round to the nearest upper value).
  - `FLOOR()`: When you need to round down (e.g., rounding down percentages or measurements).
  - `TRUNCATE()`: When you need to discard decimals without rounding.

class code “

USE Employees;

CREATE TABLE employees2 (
emp_no INT PRIMARY KEY,
first_name VARCHAR(50),

```
last_name VARCHAR(50),
salary DECIMAL(10, 2)
);

INSERT INTO employees2 (emp_no, first_name, last_name, salary)
VALUES
(1001, 'John', 'Doe', 50000.75),
(1002, 'Alice', 'Smith', 60000.55),
(1003, 'Bob', 'Johnson', 55000.25),
(1004, 'Emily', 'Davis', 58000.80),
(1005, 'Michael', 'Williams', 62000.99);

SELECT emp_no, first_name, salary, ROUND(salary, 0) AS Rounded_Salary
FROM employees2;

SELECT emp_no, first_name, salary, FLOOR(salary) AS Floored_Salary
FROM employees2;

SELECT emp_no, first_name, salary, CEIL(salary) AS Ceil_Salary
FROM employees2;

SELECT emp_no, first_name, salary, POWER(salary, 2) AS Salary_Squared
FROM employees2;

SELECT emp_no, first_name, salary, SQRT(salary) AS Salary_Square_Root
FROM employees2;
```