SQL Aggregate Function

SQL is excellent at aggregating data the way you might in a pivot table in Excel. You will use aggregate functions all the time, so it's important to get comfortable with them. The functions themselves are the same ones you will find in Excel or any other analytics program.

- COUNT counts how many rows are in a particular column.
- SUM adds together all the values in a particular column.
- MIN and MAX return the lowest and highest values in a particular column, respectively.
- AVG Calculates the average of a group of selected values.

Example:-
```
SELECT COUNT(*)
  FROM Sales ;
```

Example :-
```
SELECT COUNT (column_name)
FROM table_name
WHERE condition ;
```

Example :-
```
SELECT SUM (column_name)
FROM table_name
WHERE condition ;
```

Example :-
```
SELECT MIN (column_name)
FROM table_name
WHERE condition;
```

Example :-
```
SELECT MAX (column_name)
FROM table_name
WHERE condition ;
```

Example :-
```
SELECT AVG (column_name)
FROM table_name
WHERE condition ;
```

The SQL GROUP BY clause

GROUP BY allows you to separate data into groups, which can be aggregated independently of one another.

```
SELECT year,
       COUNT (*) AS count
FROM sales
GROUP BY year ;
```

Multiple column

```
SELECT year,
       month,
       COUNT (*) AS count
FROM sales
GROUP BY year, month;
```

GROUP BY Column numbers

```
SELECT year,
       month,
       COUNT(*) AS count
FROM sales
GROUP BY 1, 2 ;
```

Using GROUP BY with ORDER BY

```
SELECT year,
       month,
       COUNT(*) AS count
FROM sales
GROUP BY year, month
ORDER BY month, year ;
```

Using GROUP BY with LIMIT

```
SELECT column_name,
FROM table_name
WHERE condition
GROUP BY column_name
LIMIT number ;
```

HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Example :-
```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

- ```
  SELECT year,
         month,
         MAX(high) AS month_high
  FROM Sales
  GROUP BY year, month
  HAVING MAX(high) > 400
  ORDER BY year, month ;
  ```

The SQL CASE statement

The CASE statement is SQL's way of handling if/then logic. The CASE statement is followed by at least one pair of WHEN and THEN statements - SQL's equivalent of IF/THEN in Excel. Because of this pairing, you might be tempted to call this SQL CASE WHEN, but CASE is the accepted term.

Every CASE statement must end with the END statement. The ELSE statement is optional, and provides a way to capture values not specified in the WHEN/THEN statement. CASE is easiest to understand in the context of an example.
  ```

Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Example:-
```
SELECT orderID, Quantity,
CASE
    WHEN Quantity > 30 THEN "The quantity is greater than 30"
    WHEN Quantity = 30 THEN "The quantity is 30"
    ELSE "The quantity is under 30"
END AS QuantityText
FROM Sales;
```

SQL DISTINCT

You'll occasionally want to look at only the unique values in a particular column. You can do this using SELECT DISTINCT syntax.

Example:-

- ```
  SELECT DISTINCT month
  FROM Sales;
  ```

- ```
  SELECT DISTINCT year, month
  FROM Sales;
  ```

Using DISTINCT in aggregations

```
SELECT COUNT(DISTINCT month) AS unique_months
  FROM Sales;
```

MySQL JOINS

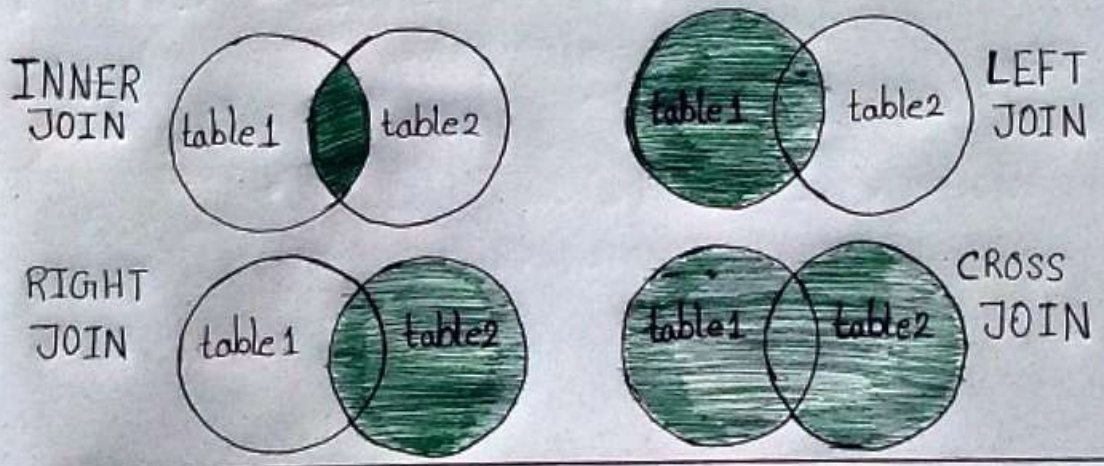
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Example:-

```
SELECT *
  FROM benn.college_football_players players
  JOIN benn.college_football_teams teams
    ON teams.school_name = players.school_name
```

Supported Types of JOINS in MySQL

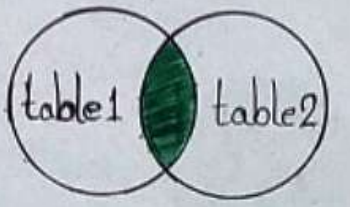
- INNER JOIN: Returns records that have matching values in both tables.
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table.
- RIGHT JOIN: Returns all records from the right table, and the matched records from the left table.
- CROSS JOIN: Returns all records from both tables.



INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN



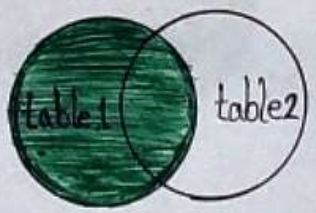
Example :-

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).

LEFT JOIN



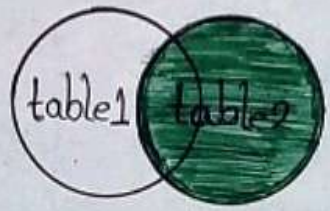
Example :-

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table 2), and the matching records (if any) from the left table (table 1).

RIGHT JOIN



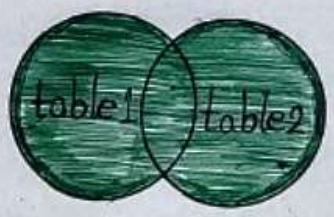
Example :-

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

CROSS JOIN

The CROSS JOIN keyword returns all records from both tables (table 1 and table 2).

CROSS JOIN



Example :-

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

SELF JOIN

A self Join is a regular join, but the table is joined with itself.

Example :-

```
SELECT column_name(s)
FROM table1 T1 , table1.T2
WHERE condition ;
```

UNION Operator

SQL joins allow you to combine two datasets side-by-side, but UNION allows you to stack one dataset on top of the other. Put differently, UNION allows you to write two separate SELECT statements, and to have the results of one statement display in the same table as the results from the other statement.

Example :-

- ```
  SELECT column_name(s) FROM table1
  UNION
  SELECT column_name(s) FROM table2 ;
  ```

- ```
  SELECT column_name(s) FROM table1
  UNION ALL
  SELECT column_name(s) FROM table2 ;
  ```

IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.
The IN operator is a shorthand for multiple OR conditions.

Example :-

- SELECT * FROM Sales
  WHERE country IN ("India", "Nepal", "UK");
- SELECT * FROM sales
  WHERE country NOT IN ("India", "Nepal", "UK");
- SELECT * FROM Sales
  WHERE country IN (SELECT country FROM suppliers);

EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

Example :-

```
SELECT column_name (s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

ANY and ALL Operator

The ANY and ALL operator allow you to perform a comparison between a single column value and a range of other values.

ANY Operator

- It returns a boolean value as a result.
- It returns TRUE if ANY of the subquery values meet the condition.

ANY means that the condition will be true if the operation is true for any of the values in the range.

Example :-

```
SELECT ProductName FROM Sales
WHERE ProductID = ANY
  (SELECT ProductID FROM OrderDetails
   WHERE Quantity > 99) ;
```

ALL Operator

- It returns a boolean value as a result.
- It returns TRUE if ALL of the subquery values meet the condition.
- It is used with SELECT, WHERE and HAVING statements.

ALL means that the condition will be true only if the operation is true for all values in the range.

Example :-

-
```
SELECT ALL ProductName
FROM Sales
WHERE TRUE ;
```

- SELECT ProductName FROM sales
  WHERE ProductID = ALL
  (SELECT ProductID FROM OrderDetails
  WHERE Quantity = 10) ;

INSERT INTO SELECT

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
The INSERT INTO SELECT statement requires that the data types in source and target tables matches.

The existing records in the target table are unaffected.

Example:-
- INSERT INTO table2
  SELECT * FROM table1
  WHERE condition ;

- INSERT INTO table2 (column1, column2, column3, ...)
  SELECT column1, column2, column3, ...
  FROM table1
  WHERE condition ;

INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

It is possible to write the INSERT INTO statement in two ways.

- Specify both the column names and the values to be inserted.

```
INSERT INTO table_name(column1, column2, column3, ...)
VALUES (value1, value2, value3, ...) ;
```

- If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...) ;
```

IFNULL() Function

IFNULL() function lets you return an alternative value if an expression is NULL.
The example below returns 0 if the value is NULL.

- ```
  SELECT contactname,
    IFNULL (bizphone, homephone) AS phone
  FROM contacts ;
  ```

- ```
  SELECT name,
    IFNULL (officephone, mobilephone) AS contact
  FROM employee ;
  ```