SQL

• What is SQL ?

SQL (Structured Query Language) is a programming language designed for managing data in relational database. SQL has a variety of functions that allow its users to read, manipulate, and change data. Though SQL is commonly used by engineers in software devolopment, it's also popular with data analysts for afew reason:

- It's semantically easy to understand and learn.
- Because it can be used to access large amounts of data directly where it's stored, analysts don't have to copy data into other applications.
- Compared to spereadsheet tools, data analysis done in SQL is easy to audit and replicate. For analysts, this means no more looking for the cell with the typo in the formula.

• SELECT * Example

The following SQL statement selects all the columns from the "Sales" table:

Example - **SELECT * FROM Sales ;**

- Select columns wise

Example-
```
SELECT year,
       month,
       west
FROM Sales
```

- Rename Columns

Example -
```
SELECT west AS "west Region"
  FROM Sales
```

- LIMIT Clause

The LIMIT clause is used to specify the number of records to return

Example -
```
SELECT *
  FROM Sales
LIMIT 100
```

- WHERE Clause

The WHERE clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

Example-
```
SELECT *
   FROM sales
WHERE country = "Canada";
```

- **Comparison operators on numerical data**

The most basic way to filter data is using Comparison operators. The easiest way to understand them is to start by looking at a list of them :

Equal to	=
Not equal to	<> or !=
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=

Example-

```
• SELECT *
    FROM Sales
  WHERE city = " kolkata";

• SELECT *
    FROM Sales
  WHERE city != "Kolkata" ;

• SELECT *
    FROM Sales
  WHERE Month > " January" ;

• SELECT *
    FROM Sales
  WHERE Sale_amount <  50000
```

- ## Arithmetic in SQL

You can perform arithmetic in SQL using the Same operators you would in Excel: +, -, *, /. However in SQL you can only perform arithmetic across columns on values in a given row. To clarify, you can only add values in multiple columns from the Same row together using + — if you want to add values across multiple rows, you'll need to use aggregate functions.

Example -
```
SELECT year,
       month,
       west,
       South,
       west + south AS South_pluswest
FROM sales ;
```

Example -
```
    SELECT year,
           month,
           west,
           south,
           west + south - 4 * year AS new_column
FROM Sales ;
```

Example -
```
SELECT year,
       month,
       west,
       south,
       (west + south)/2 AS South_west_avg
FROM Sales ;
```

- ## CREATE TABLE

The CREATE TABLE statement is used to create a new table in a database.

Example-
```
CREATE TABLE person(
        PersonID int,
        LastName varchar(255),
        FristName varchar(255),
        Address varchar(255),
        city varchar(255),
    );
```

- ## INSERT INTO

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

1. specify both the column names and the values to be inserted :

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

2. 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.

```
INSERT INTO table_name
VALUES (value1, Value2, value3, ....);
```

- **What is a NULL Value?**

A field with a NULL value is a field with no value.
If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

- **How to Test for NULL Values?**

It is not possible to test for NULL values with comparison operators, such as =,<, or<>.
we will have to use the IS NULL and IS NOT NULL operators instead.

- **The IS NULL Operator**

The IS NULL operator is used to test for empty values (NULL values).

Example-
```
SELECT customerName,contactName,Address
FROM Sales
WHERE Address IS NULL ;
```

- **The IS NOT NULL Operator**

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).
Example -
```
SELECT customerName, ContactName, Address
FROM Sales
WHERE Address IS NOT NULL ;
```

UPDATE Statement

The UPDATE statement is used to modify the existing records in a tabel.

Example-

```
UPDATE Sales
SET contactName =" Alan", city =" Goa"
WHERE Customer ID = 1 ;
```

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

Example-

```
UPDATE Sales
SET PostalCode = 00000
WHERE Country = " India" ;
```

Notes :-

Be carefull when updateing records. If you omit the WHERE clause, ALL records will be updated!

DELETE Statement

The DELETE statement is used to delete existing records in a table.

Example- `DELETE FROM Sales WHERE CustomerName =" Bob";`

[illegible]

• Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Example -

```
DELETE FROM table_name ;
```

• Aliases

Aliases are used to give a table, or a column in a table, a tempoary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

• Alias Column Example

```
SELECT column_name AS alias_name
FROM table_name ;
```

• Alias Table Example

```
SELECT column_name (s)
FROM table_name AS alias_name ;
```

- ## SQL Logical operators

Logical operators allow you to use multiple Comparison operators in one query.
Each logical operator is a special snowflake, so we'll go through them individually in the following lessons.

- LIKE allows you to match similar values, instead of exact values.
- IN allows you to specify a list of values you'd like to include.
- BETWEEN allows you to select only rows within a certain range.
- IS NULL allows you to select rows that contain no data in a given column.
- AND allows you to select only rows that satisfy two conditions.
- OR allows you to select rows that satisfy either of two conditions.
- NOT allows you to select rows that do not match a certain condition.

- ## LIKE Operator

```
SELECT *
   FROM Sales
WHERE "group" LIKE 'New%';
```

- **IN Operator**

```
SELECT*
   FROM Songs
WHERE artist IN ('Taylor swift', 'Usher');
```

- **BETWEEN Operator**

```
SELECT*
   FROM Songs
WHERE year_rank BETWEEN 5 AND 10;
```

- **AND Operator**

```
SELECT *
   FROM Songs
WHERE year = 2012 AND year_rank <= 10;
```

- **OR Operator**

```
SELECT*
   FROM Songs
WHERE year_rank = 5 OR artist = "Sonu";
```

- **NOT Operator**

```
SELECT*
   FROM Sales
WHERE NOT Country = 'Japan";
```

- Combining AND, OR and NOT

```
SELECT * FROM Sales
WHERE country='Japan' AND (city='GOa' OR city='Puri')
```

- ORDER BY

```
SELECT *
   FROM Sales
ORDER BY country, CustomerName ;

SELECT * FROM Sales
ORDER BY Country ASC, CustomerName DESC ;
```

- Using Comments (How to use comments)

```
• SELECT *   -- This is select command
    FROM sales
  WHERE year = 2020 ;

• /* Here's a comment so long and descriptive that
it could only fit on multiple lines. Fortunately,
it, too, will not affect how this code runs. */
SELECT *
   FROM Sales
WHERE year = 2015 ;
```

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 result 1
            WHEN condition 2  THEN result2
            WHEN conditionN  THEN resultN
            ELSE result
      END ;
```

Example :-

```
SELECT orderID , Quantity,
   CASE
      WHEN Quantity > 30 THEN "The quantity is greater than
      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.

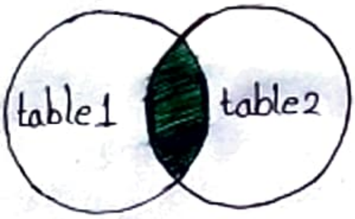
Example :-

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

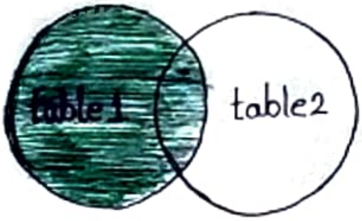
Supported Types of JOINS in MySQL

- 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.

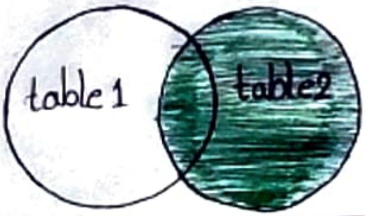
INNER JOIN



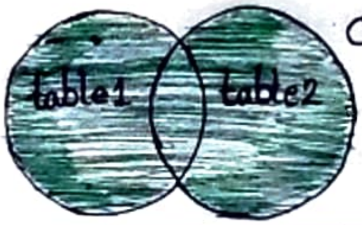
LEFT JOIN



RIGHT JOIN



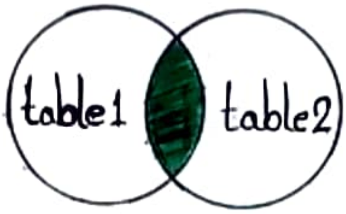
CROSS JOIN



INNER JOIN

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

INNER JOIN



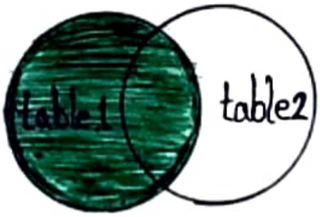
Example :-

```
SELECT column_name(s)
FROM table 1
INNER JOIN table 2
ON table1. column_name = table2.column_name;
```

LEFT JOIN

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

LEFT JOIN



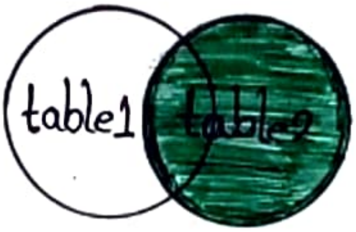
Example :-

```
SELECT column_name(s)
FROM table 1
LEFT JOIN table 2
ON table1.column_name = table2. column_name;
```

RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table 1).

RIGHT JOIN



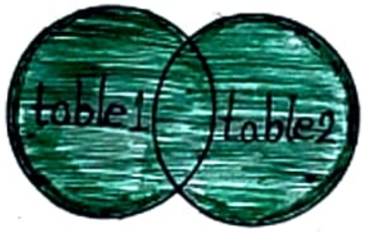
Example :-

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table 2
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 table 1
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 anothertable.

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 ;
  ```

-
  ```
  INSERTINTO 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 ;
  ```