Introduction to SQL

- SQL (Structured Query Language) is a computer language aimed to store, manipulate and retrieve data stored in relational databases.
- SQL language has several parts:

1) DDL - Data Definition Language
2) DML - Data Manipulation Language
3) View Definition
4) Transaction Control

1. Data Definition Language:

- DDL statements are used to define the database structure or schema. Examples :

i. CREATE
ii. ALTER
iii. DROP
iv. RENAME

2. Data Manipulation Language:

- DML statements are used for managing data within schema objects. Examples :

i. SELECT
ii. INSERT
iii. UPDATE
iv. DELETE

Basic Structure of SQL Query:

- General Structure - SELECT, ALL/ DISTINCT, *, AS, FROM, WHERE
- Comparison - IN, BETWEEN, LIKE, ILIKE
- Grouping - GROUP BY, HAVING, COUNT(), SUM(), AVG(), MAX(), MIN()
- Display Order - ORDER BY, ASC/ DESC
- Logical Operators - AND, OR, NOT
- Output - INTO TABLE/ CURSOR, TO SCREEN
- Union - UNION

COUNT Statement:

- COUNT returns the number of input rows that match a specific condition of a query

- Syntax:

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

- COUNT can be applied on a specific column or just passed as COUNT(*), both giving same results

WHERE Statement:

- WHERE statement allows us to specify the conditions on columns for the rows to be returned.

OPERATOR	DESCRIPTION
=	EQUAL
>	GREATER THAN
<	LESS THAN
>=	GREATER THAN OR EQUAL TO
<=	LESS THAN OR EQUAL TO
<> OR !=	NOT EQUAL TO
AND	BOTH CONDITIONS TRUE
OR	EITHER OF THE CONDITIONS TRUE
NOT	OFFSET OF SPECIFIED CONDITION

- The WHERE clause is used to filter records.
- Syntax:

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

LIMIT Statement:

- The LIMIT command allows us to limit the number of rows returned for a query.
- Useful for not wanting to return every single row in a table, but only view the top few rows to get an idea of the table layout
- LIMIT also becomes useful in combination with ORDER BY
- LIMIT goes at the very end of a query request and is the last command to be executed

BETWEEN Statement:

- The BETWEEN operator can be used to match a value against a range of values
- Syntax:

```
value BETWEEN low AND high
```

- Can be combined with NOT operator
- Can be also used with dates in the format : YYYY-MM-DD

IN Statement:

- The IN operator can be used to create a condition that checks to see if a value is included in a list of multiple options for example if a user's name shows up in a list of know names
- Syntax:

```
value IN ( option 1, option 2, option 3,....., option N
```

LIKE & ILIKE Statement:

- In order to match a string against a general pattern we use LIKE and ILIKE for example:

All emails ending with '@gmail.com'

- LIKE & ILIKE allows us to perform pattern matching against string data with the use of wildcard characters

● Percent % - Matches any sequence of characters

● Underscore _ - Matches any single character

- LIKE is case-sensitive whereas ILIKE is case-insensitive

- All names that begin with ‘A’

```
WHERE name LIKE ‘A%’
```

- All names that end with ‘a’

```
WHERE name LIKE ‘%a’
```

- Get all Mission Impossible Films

```
WHERE name LIKE ‘Mission Impossible _ _’
```

- Combination of Wildcards

```
WHERE name LIKE ‘_ her%’
```

● Cheryl

● Theresa

● Sherri

Aggregate Functions:

- The main idea behind aggregate function is to take multiple inputs and return a single output

i. AVG() - Returns floating point values.
ii. ROUND() can be used to specify precision after the decimal
iii. COUNT()
iv. MAX()
v. MIN()
vi. SUM()

The AVG() function:

- It returns the average value of a numeric column.
- Syntax:

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

The SUM() function :

- It returns the total sum of a numeric column.
- Syntax:

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

The MIN() function:

- It returns the smallest value of the selected column.
- MIN() Syntax:

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

The MAX() function:

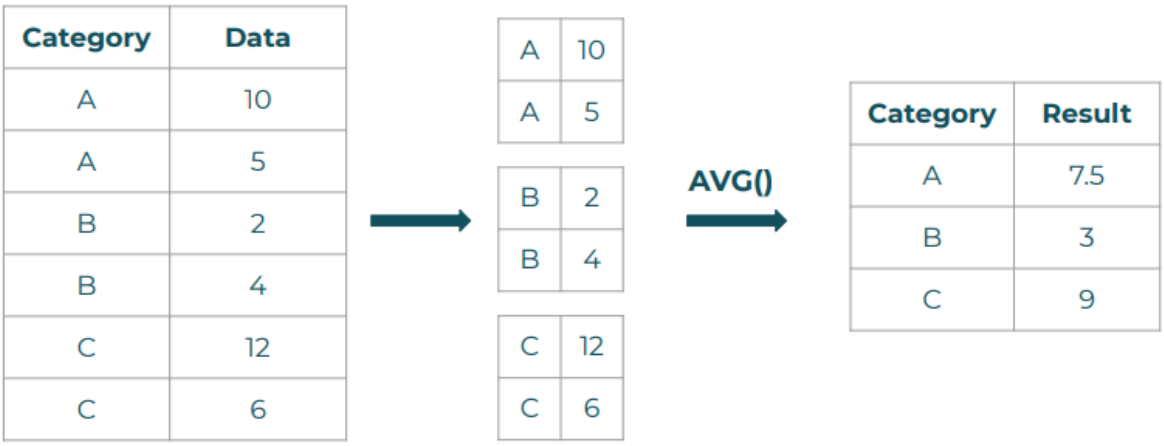
- It returns the largest value of the selected column.
- MAX() Syntax:

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

GROUP BY Statement:

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- It is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.
- Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```



HAVING BY Statement:

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

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

SQL 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.

- Syntax:

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

SQL ANY Operator:

- ANY means that the condition will be true if the operation is true for any of the values in the range.
- Returns a boolean value as a result
- Returns TRUE if ANY of the subquery values meet the condition
- Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

SQL SELECT INTO Statement:

- The SELECT INTO statement copies data from one table into a new table.
- Syntax:
  Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

- Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

SQL INSERT INTO SELECT Statement:

- It copies data from one table and inserts it into another table.
- It requires that the data types in source and target tables match.
- The existing records in the target table are unaffected.
- Syntax:
  Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

- Copy only some columns from one table into another table:

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

SQL CASE Expression:

- The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.
- Syntax:

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

Operations in MySQL:

SELECT & DISTINCT Statement:.

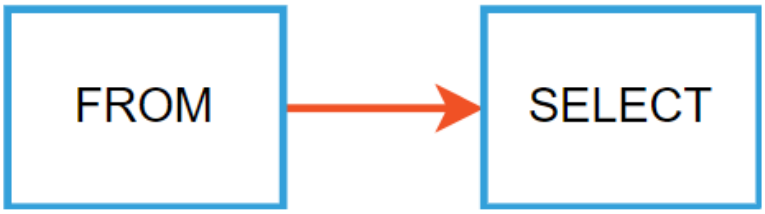
- The SELECT statement allows you to select data from one or more tables. To write a SELECT statement in MySQL, you use this syntax:

```
SELECT select_list
FROM table_name;
```

In this syntax:

- First, specify one or more columns from which you want to select data after the SELECT keyword. If the select_list has multiple columns, you need to separate them by a comma (,).
- Second, specify the name of the table from which you want to select data after the FROM keyword.
- The semicolon (;) is optional. It denotes the end of a statement. If you have two or more statements, you need to use the semicolon (;) to separate them so that MySQL will execute each statement individually.

- When executing the SELECT statement, MySQL evaluates the FROM clause before the SELECT clause:



- Sometimes a table contains a column that has duplicate values and in order to return only the unique values, DISTINCT statement is used in combination with SELECT

ORDER BY statement:

- When you use the SELECT statement to query data from a table, the order of rows in the result set is unspecified. To sort the rows in the result set, you add the ORDER BY clause to the SELECT statement.
- Syntax of the ORDER BY clause:

```
SELECT
  select_list
FROM
  table_name
ORDER BY
  column1 [ASC|DESC],
  column2 [ASC|DESC],
  ...;
```

- In this syntax, you specify the one or more columns that you want to sort after the ORDER BY clause.
- The ASC stands for ascending and the DESC stands for descending.
- ASC to sort the result set in ascending order and DESC to sort the result set in descending order respectively.

INSERT statement:

- The INSERT statement allows you to insert one or more rows into a table. The following illustrates the syntax of the INSERT statement:

```
INSERT INTO table(c1,c2,...)
VALUES (v1,v2,...);
```

In this syntax,

- First, specify the table name and a list of comma-separated columns inside parentheses after the INSERT INTO clause.
- Then, put a comma-separated list of values of the corresponding columns inside the parentheses following the VALUES keyword.
- The number of columns and values must be the same. In addition, the positions of columns must be corresponding with the positions of their values.

Syntax to insert multiple rows into a table using a single INSERT statement:

```
INSERT INTO table(c1,c2,...)
VALUES
  (v11,v12,...),
  (v21,v22,...),
   ...
  (vnn,vn2,...);
```

- In this syntax, rows are separated by commas in the VALUES clause.
- Consider the example shown below:
- We have created a tasks table, so now let's insert a row in that table using INSERT :

```
INSERT INTO tasks(title,priority)
VALUES('Learn MySQL INSERT Statement',1);
```

- MySQL returns the following message:

```
1 row(s) affected
```

- It means that one row has been inserted into the tasks table successfully.

UPDATE statement:

- It updates data in a table. It allows you to change the values in one or more columns of a single row or multiple rows.
- Syntax of the UPDATE statement:

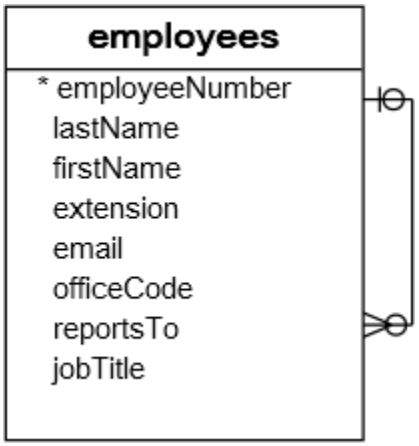
```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
   column_name1 = expr1,
   column_name2 = expr2,
   ...
[WHERE
   condition];
```

In this syntax:

- First, specify the name of the table that you want to update data after the UPDATE keyword.
- Second, specify which column you want to update and the new value in the SET clause. To update values in multiple columns, you use a list of comma-separated assignments by supplying a value in each column's assignment in the form of a literal value, an expression, or a subquery.
- Third, specify which rows to be updated using a condition in the WHERE clause. The WHERE clause is optional. If you omit it, the UPDATE statement will modify all rows in the table.
- Note that the WHERE clause is so important that you should not forget.
- MySQL supports two modifiers in the UPDATE statement.

1. The LOW_PRIORITY modifier instructs the UPDATE statement to delay the update until there is no connection reading data from the table
2. The IGNORE modifier enables the UPDATE statement to continue updating rows even if errors occurred. The rows that cause errors such as duplicate-key conflicts are not updated.

- E.g.: See the following employees table.



- In this example, we will update the email of Mary Patterson to the new email mary.patterso@classicmodelcars.com**.**
- First, find Mary's email from the employees table using the following SELECT statement:

```
SELECT
   firstname,
   lastname,
   email
FROM
   employees
WHERE
   employeeNumber = 1056;
```
Code language: SQL (Structured Query Language) (sql)

	firstname	lastname	email
▶	Mary	Patterson	mpatterso@classicmodelcars.com

- Second, update the email address of Mary to the new email mary.patterson@classicmodelcars.com :

```
UPDATE employees
SET
   email = 'mary.patterson@classicmodelcars.com'
WHERE
   employeeNumber = 1056;
```

- MySQL issued the number of rows affected:

```
1 row(s) affected
```

DELETE statement:

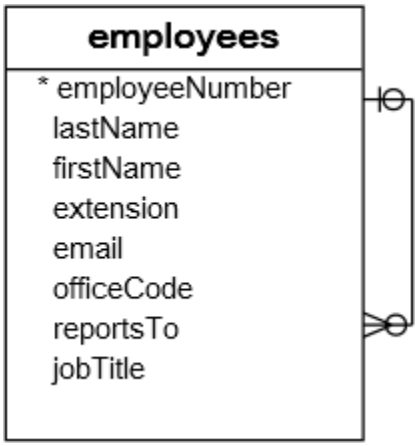
To delete data from a table, you use the MySQL DELETE statement. The following illustrates the syntax of the DELETE statement:

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

- In this statement:

1. First, specify the table from which you delete data.
2. Second, use a condition to specify which rows to delete in the WHERE clause. The DELETE statement will delete rows that match the condition,

- Note that the WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table.
- Besides deleting data from a table, the DELETE statement returns the number of deleted rows.
- Let’s understand this with the example given below, we will use the employees table.



- Note that once you delete data, it is gone. Later, you will learn how to put the DELETE statement in a transaction so that you can roll it back.
- Suppose you want to delete employees whose the officeNumber is 4, you use the DELETE statement with the WHERE clause as shown in the following query:

```
DELETE FROM employees
WHERE officeCode = 4;
```

- To delete all rows from the employees table, you use the DELETE statement without the WHERE clause as follows:

```
DELETE FROM employees;
```

- All rows in the employees table deleted.

Conditional Statements and Operators:

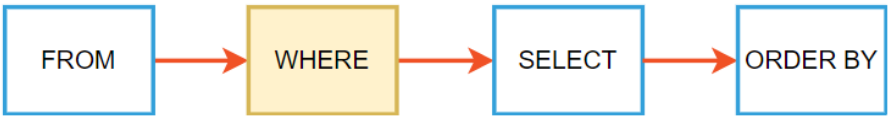
WHERE clause:

The WHERE clause allows you to specify a search condition for the rows returned by a query. The following shows the syntax of the WHERE clause:

```
SELECT
   select_list
FROM
   table_name
WHERE
   search_condition;
```

- The search_condition is a combination of one or more expressions using the logical operator AND, OR and NOT.

- In MySQL, a predicate is a Boolean expression that evaluates to TRUE, FALSE, or UNKNOWN.
- The SELECT statement will include any row that satisfies the search_condition in the result set.
- Besides the SELECT statement, you can use the WHERE clause in the UPDATE or DELETE statement to specify which rows to update or delete.
- When executing a SELECT statement with a WHERE clause, MySQL evaluates the WHERE clause after the FROM clause and before the SELECT and ORDER BY clauses:



AND Operator:

- MySQL doesn't have a built-in Boolean type. Instead, it uses the number zero as FALSE and non-zero values as TRUE.
- The AND operator is a logical operator that combines two or more Boolean expressions and returns 1, 0, or NULL:

A AND B

- In this expression, A and B are called operands. They can be literal values or expressions.
- The logical AND operator returns 1 if both A and B are non-zero and not NULL. It returns 0 if either operand is zero; otherwise, it returns NULL.
- The logical AND operator returns 1 if both A and B are non-zero and NOT NULL.

OR Operator:

- The MySQL OR operator is a logical operator that combines two Boolean expressions.

A OR B

- If both A and B are not NULL, the OR operator returns 1 (true) if either A or B is non-zero.

IN Operator:

- It allows you to determine if a value matches any value in a list of values. Here's the syntax of the IN operator:

value IN (value1, value2, value3,...)\

- The IN operator returns 1 (true) if the value equals any value in the list (value1, value2, value3,…). Otherwise, it returns 0.
- In this syntax:
- First, specify the value to test on the left side of the IN operator. The value can be a column or an expression.
- Second, specify a comma-separated list of values to match in the parentheses.

BETWEEN Operator:

- It is a logical operator that specifies whether a value is in a range or not. Here's the syntax of the BETWEEN operator:

```
value BETWEEN low AND high;
```

- The BETWEEN operator returns 1 if: value >= low AND value <= high
- Otherwise, it returns 0.
- If the value, low, or high is NULL, the BETWEEN operator returns NULL .
- For example, the following statement returns 1 because 15 is between 10 and 20:

```
SELECT 15 BETWEEN 10 AND 20;
```

- The following example returns 0 because 15 is not between 20 and 30:

```
SELECT 15 BETWEEN 20 AND 30;
```

IS NULL Operator:

To test whether a value is NULL or not, you use the IS NULL operator. Here's the basic syntax of the IS NULL operator:

```
value IS NULL
```

- If the value is NULL, the expression returns true. Otherwise, it returns false.
- Note that MySQL does not have a built-in BOOLEAN type. It uses the TINYINT(1) to represent the BOOLEAN values i.e., true means 1 and false means 0.
- Because the IS NULL is a comparison operator, you can use it anywhere that an operator can be used e.g., in the SELECT or WHERE clause.
- See the following example:

```
SELECT 1 IS NULL,  -- 0
    0 IS NULL,  -- 0
    NULL IS NULL; -- 1
```

- To check if a value is not NULL, you use IS NOT NULL operator:

```
value IS NOT NULL
```

- This expression returns true (1) if the value is not NULL. Otherwise, it returns false (0).