08 SQL Basics

- **SQL Syntax**
- **Data Types in MySQL**
- **Basic SQL Operations:**
    - **SELECT, INSERT, UPDATE, DELETE**
- **Filtering Data:**
    - **WHERE, ORDER BY, DISTINCT**
- **Aggregate Functions:**
    - **COUNT, SUM, AVG, MIN, MAX**
- **Joins:**
    - **INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN**
- **Subqueries**
- **Aliases**

01 SQL Syntax

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

- **SELECT:** Specifies the columns to retrieve
- **FROM:** Specifies the table to query
- **WHERE:** Filters records based on a condition

02 Data Types in MySQL

Data types define the type of data that can be stored in a column

Common Data Types:

- **INT:** Integer values
- **VARCHAR(size):** Variable-length string with a maximum length defined by `size`
- **DATE:** Date in the format `YYYY-MM-DD`
- **FLOAT:** Floating-point numbers

Example:

```sql
CREATE TABLE Products (
    ProductID INT,
    ProductName VARCHAR(100),
    Price FLOAT,
    ManufactureDate DATE
);
```

- **Explanation:** This creates a `Products` table with four columns: an integer `ProductID`, a string `ProductName`, a floating-point `Price`, and a date `ManufactureDate`

03 Basic SQL Operations

SELECT

Retrieves data from a table

Example:

```sql
SELECT ProductName, Price
FROM Products;
```

- **Explanation:** Retrieves the `ProductName` and `Price` columns from the `Products` table

INSERT

Inserts new data into a table

Example:

```sql
INSERT INTO Products (ProductID, ProductName, Price, ManufactureDate)
VALUES (1, 'Laptop', 800.00, '2023-08-01');
```

- **Explanation:** Inserts a new product with an ID of 1, a name of 'Laptop', a price of 800.00, and a manufacture date of August 1, 2023

UPDATE

Updates existing data in a table

Example:

```
UPDATE Products
SET Price = 850.00
WHERE ProductID = 1;
```

- **Explanation:** Updates the price of the product with `ProductID` 1 to 850.00

DELETE

Deletes data from a table

Example:

```
DELETE FROM Products
WHERE ProductID = 1;
```

- **Explanation:** Deletes the record with `ProductID` 1 from the `Products` table

04 Filtering Data

These operations are used to filter and sort the data retrieved from a table

WHERE

Filters records based on a specific condition

Example:

```
SELECT *
FROM Products
WHERE Price > 500;
```

- **Explanation:** Retrieves all products with a price greater than 500

ORDER BY

Sorts the result set by one or more columns

Example:

```
SELECT *
FROM Products
```

```
ORDER BY Price DESC;
```

- **Explanation:** Retrieves all products sorted by price in descending order

GROUP BY

Groups rows that have the same values in specified columns and allows aggregate functions (like `COUNT`, `SUM`, `AVG`, etc.) to be applied to each group

Example:

```
SELECT Department, COUNT(EmployeeID) AS NumberOfEmployees
FROM Employees
GROUP BY Department;
```

- **Explanation:** Groups employees by department and counts the number of employees in each department. The result will show the department name and the corresponding number of employees in that department

DISTINCT

Retrieves unique values from a column

Example:

```
SELECT DISTINCT ProductName
FROM Products;
```

- **Explanation:** Retrieves unique product names from the `Products` table

05 Aggregate Functions

These functions perform calculations on multiple rows of a table and return a single value

COUNT

Counts the number of rows

Example:

```
SELECT COUNT(*)
```

```
FROM Products;
```

- **Explanation:** Returns the total number of products

SUM

Calculates the total sum of a numeric column

Example:

```
SELECT SUM(Price)
FROM Products;
```

- **Explanation:** Returns the total sum of all product prices

AVG

Calculates the average value of a numeric column

Example:

```
SELECT AVG(Price)
FROM Products;
```

- **Explanation:** Returns the average price of all products

MIN

Finds the minimum value in a column

Example:

```
SELECT MIN(Price)
FROM Products;
```

- **Explanation:** Returns the lowest price among all products

MAX

Finds the maximum value in a column

Example:

```sql
SELECT MAX(Price)
FROM Products;
```

- **Explanation:** Returns the highest price among all products

06 Joins

Joins are used to retrieve data from multiple tables based on a related column

INNER JOIN

Returns records that have matching values in both tables

Example:

```sql
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **Explanation:** Retrieves order IDs and customer names for orders where there is a matching customer

LEFT JOIN (or LEFT OUTER JOIN)

Returns all records from the left table, and the matched records from the right table. Unmatched records from the right table will return `NULL`

Example:

```sql
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

- **Explanation:** Retrieves all customers and their order IDs. If a customer has no orders, `NULL` is returned for the order ID

RIGHT JOIN (or RIGHT OUTER JOIN)

Returns all records from the right table, and the matched records from the left table. Unmatched records from the left table will return `NULL`

Example:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **Explanation:** Retrieves all orders and their associated customer names. If an order has no matching customer, `NULL` is returned for the customer name

FULL JOIN (or FULL OUTER JOIN)

Returns all records when there is a match in either left or right table. Unmatched records from either table will return `NULL`

Note: MySQL doesn't directly support `FULL JOIN`. You can simulate it using `UNION`

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
UNION
SELECT Customers.CustomerName, Orders.OrderID
FROM Orders
RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

- **Explanation:** Retrieves all customers and their order IDs, including those customers with no orders and orders with no matching customer

07 Subqueries

A subquery is a query within another query. It provides a way to retrieve data to be used in the main query

Example:

```
SELECT ProductName
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

- **Explanation:** Retrieves product names where the price is greater than the average price of all products

08 Aliases

Aliases provide a temporary name for a table or column. It is useful for renaming columns or tables to make the query more readable

Example:

```
SELECT ProductName AS Name, Price AS Cost
FROM Products;
```

- **Explanation:** Renames the `ProductName` column as `Name` and `Price` as `Cost` in the result set