SQL: Interview Preparation Questions

1. **What do you mean by data definition language?**

   Data definition language or DDL allows the execution of queries like CREATE, DROP, and ALTER. That is those queries that define the data.

2. **What do you mean by data manipulation language?**

   Data manipulation Language or DML is used to access or manipulate data in the database. It allows us to perform the below-listed functions:

   - Insert data or rows in a database
   - Delete data from the database
   - Retrieve or fetch data
   - Update data in a database.

3. **What is the view in SQL?**

   Views in SQL are a kind of virtual table. A view also has rows and columns as they are on a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions.

   The CREATE VIEW statement of SQL is used for creating views.

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;

view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows
```

4. **What do you mean by foreign key?**

   A Foreign key is a field that can uniquely identify each row in another table. This constraint is used to specify a field as a Foreign key. That is this field points to the primary key of another table. This usually creates a kind of link between the two tables.

5. **What is the primary key?**
   A Primary Key is one of the candidate keys. One of the candidate keys is selected as the most important and becomes the primary key. There cannot be more than one primary key in a

table. A Primary key is unique to ensure that each record in the table is distinct and easily identifiable.

6. **What is a stored procedure?**

   Stored Procedures are created to perform one or more DML operations on databases. It is nothing but a group of SQL statements that accepts some input in the form of parameters, performs some task, and may or may not return a value.

7. **What are aggregate and scalar functions?**

   For doing operations on data SQL has many built-in functions, they are categorized into two categories and further sub-categorized into seven different functions under each category. The categories are:

   Aggregate functions: These functions are used to do operations from the values of the column and a single value is returned.
   Example: MAX(), MIN(), SUM(), AVG(), COUNT(), etc.

   Scalar functions: These functions are based on user input, these too return a single value.
   Example: ABS(), ROUND(), etc.

8. **What is an ALIAS command?**

   Aliases are the temporary names given to a table or column for the purpose of a particular SQL query. It is used when the name of a column or table is used other than its original name, but the modified name is only temporary.

   Aliases are created to make table or column names more readable.
   The renaming is just a temporary change and the table name does not change in the original database.

   Aliases are useful when table or column names are big or not very readable.
   These are preferred when there is more than one table involved in a query.

9. **What is the difference between BETWEEN and IN operators in SQL?**

   BETWEEN: The BETWEEN operator is used to fetch rows based on a range of values.

   For example,

```
SELECT * FROM Students
WHERE ROLL_NO BETWEEN 20 AND 30;
```

   This query will select all those rows from the table. Students where the value of the field ROLL_NO lies between 20 and 30.

IN: The IN operator is used to check for values contained in specific sets.

For example,

```
SELECT * FROM Students
WHERE ROLL_NO IN (20,21,23);
```

This query will select all those rows from the table Students where the value of the field ROLL_NO is either 20 or 21 or 23.

10. Describe the difference between WHERE and HAVING in SQL.

The WHERE clause is employed to restrict individual rows before they are grouped, such as when filtering rows prior to a GROUP BY operation. Conversely, the HAVING clause is utilized to filter groups of rows after they have been grouped, like filtering groups based on aggregate values.

11. Write an SQL query to extract the first three characters from a column named ProductName in the Products table.

Variations:

1. Write an SQL query to retrieve the last four characters from the ProductCode column in the Products table.
2. Write an SQL query to extract a substring starting from the 2nd character to the 5th character in the Description column of the Products table.
3. Write an SQL query to extract the first five characters from the Category column in the Products table.

```
-- Original Question Solution
SELECT LEFT(ProductName, 3) AS ShortName FROM Products;

-- Variation 1 Solution
SELECT RIGHT(ProductCode, 4) AS ShortCode FROM Products;

-- Variation 2 Solution
SELECT SUBSTRING(Description, 2, 4) AS SubDescription FROM Products;

-- Variation 3 Solution
SELECT LEFT(Category, 5) AS ShortCategory FROM Products;
```

12. Write an SQL query to replace all occurrences of the word "Basic" with "Premium" in the PlanType column of the Subscriptions table.

Variations:

1. Write an SQL query to replace all spaces with hyphens (-) in the Address column of the

Customers table.
2. Write an SQL query to replace any occurrence of the substring "Temp" with "Permanent" in the JobStatus column in the Employees table.
3. Write an SQL query to replace all instances of the letter "A" with "E" in the ProductName column of the Products table.

```sql
-- Original Question Solution
SELECT REPLACE(PlanType, 'Basic', 'Premium') AS UpdatedPlan FROM Subscriptions;

-- Variation 1 Solution
SELECT REPLACE(Address, ' ', '-') AS HyphenatedAddress FROM Customers;

-- Variation 2 Solution
SELECT REPLACE(JobStatus, 'Temp', 'Permanent') AS UpdatedStatus FROM Employees;

-- Variation 3 Solution
SELECT REPLACE(ProductName, 'A', 'E') AS ModifiedProductName FROM Products;
```

13. Write an SQL query to extract the year from the OrderDate column in the Orders table.

Variations:

1. Write an SQL query to retrieve the month name (e.g., January, February) from the OrderDate column in the Orders table.
2. Write an SQL query to extract the day of the week from the OrderDate column in the Orders table.
3. Write an SQL query to retrieve the day number (e.g., 01, 02) of the month from the OrderDate column in the Orders table.

```sql
-- Original Question Solution
SELECT YEAR(OrderDate) AS OrderYear FROM Orders;

-- Variation 1 Solution
SELECT MONTHNAME(OrderDate) AS OrderMonth FROM Orders;

-- Variation 2 Solution
SELECT DAYNAME(OrderDate) AS OrderDayOfWeek FROM Orders;

-- Variation 3 Solution
SELECT LPAD(DAY(OrderDate), 2, '0') AS DayOfMonth FROM Orders;
```

14. Write an SQL query to convert the date in the OrderDate column of the Orders table to

the format DD-MM-YYYY.

Variations:

1. Write an SQL query to convert the OrderDate column to the format Month DD, YYYY (e.g., January 01, 2024).
2. Write an SQL query to display the OrderDate column in the format YYYY/MM/DD.
3. Write an SQL query to display only the time (HH:MM
) from the OrderDate column if it has datetime values.

```
-- Original Question Solution
SELECT DATE_FORMAT(OrderDate, '%d-%m-%Y') AS FormattedOrderDate FROM Orders;

-- Variation 1 Solution
SELECT DATE_FORMAT(OrderDate, '%M %d, %Y') AS FormattedOrderDate FROM Orders;

-- Variation 2 Solution
SELECT DATE_FORMAT(OrderDate, '%Y/%m/%d') AS FormattedOrderDate FROM Orders;

-- Variation 3 Solution
SELECT DATE_FORMAT(OrderDate, '%H:%i:%s') AS OrderTime FROM Orders;
```

15. Write an SQL query to convert a string column OrderDateStr in the Orders table (formatted as YYYYMMDD) to a date.

Variations:

1. Write an SQL query to convert a string OrderDateStr (formatted as DD-MM-YYYY) to a date in the Orders table.
2. Write an SQL query to convert a string OrderDateStr (formatted as MM/DD/YYYY) to a date in the Orders table.
3. Write an SQL query to convert a string OrderDateStr (formatted as YYYY Month DD like 2023 January 15) to a date in the Orders table.

```
-- Original Question Solution
SELECT STR_TO_DATE(OrderDateStr, '%Y%m%d') AS ConvertedOrderDate FROM Orders;

-- Variation 1 Solution
SELECT STR_TO_DATE(OrderDateStr, '%d-%m-%Y') AS ConvertedOrderDate FROM Orders;

-- Variation 2 Solution
SELECT STR_TO_DATE(OrderDateStr, '%m/%d/%Y') AS ConvertedOrderDate FROM Orders;

-- Variation 3 Solution
SELECT STR_TO_DATE(OrderDateStr, '%Y %M %d') AS ConvertedOrderDate FROM Orders;
```

16. Write an SQL query to add 7 days to the OrderDate column in the Orders table and display it as NewDeliveryDate.

Variations:

1. Write an SQL query to subtract 1 month from the OrderDate column in the Orders table and display it as PreviousMonthOrderDate.
2. Write an SQL query to add 3 years to the OrderDate column in the Orders table and display it as FutureOrderDate.
3. Write an SQL query to find the difference in days between DeliveryDate and OrderDate in the Orders table.

```sql
-- Original Question Solution
SELECT OrderDate, DATE_ADD(OrderDate, INTERVAL 7 DAY) AS NewDeliveryDate FROM
Orders;

-- Variation 1 Solution
SELECT OrderDate, DATE_SUB(OrderDate, INTERVAL 1 MONTH) AS
PreviousMonthOrderDate FROM Orders;

-- Variation 2 Solution
SELECT OrderDate, DATE_ADD(OrderDate, INTERVAL 3 YEAR) AS FutureOrderDate FROM
Orders;

-- Variation 3 Solution
SELECT OrderDate, DeliveryDate, DATEDIFF(DeliveryDate, OrderDate) AS DaysDifference
FROM Orders;
```

17. Write an SQL query to select all records from the Employees table where the DepartmentID is either 1, 2, or 3.

Variations:

1. Write an SQL query to select all records from the Employees table where the Salary is between 50000 and 100000.
2. Write an SQL query to select all records from the Products table where the ProductID is either 101, 103, or 107.
3. Write an SQL query to retrieve all records from the Orders table where the OrderDate is between '2023-01-01' and '2023-12-31'.

```sql
-- Original Question Solution using IN
```

```
SELECT * FROM Employees WHERE DepartmentID IN (1, 2, 3);

-- Variation 1 Solution using BETWEEN
SELECT * FROM Employees WHERE Salary BETWEEN 50000 AND 100000;

-- Variation 2 Solution using IN
SELECT * FROM Products WHERE ProductID IN (101, 103, 107);

-- Variation 3 Solution using BETWEEN for dates
SELECT * FROM Orders WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31';
```

Data for Questions 18-19:

Table: Employees

EmployeeID	Name	DepartmentID	Salary
1	Alice	1	50000
2	Bob	2	60000
3	Charlie	1	55000
4	David	3	70000
5	Eve	2	48000

Table: Departments

DepartmentID	DepartmentName
1	HR
2	Finance
3	IT

Projects

ProjectID	ProjectName	DepartmentID
101	Alpha	1
102	Beta	2
103	Gamma	3
104	Delta	1

18. **Write an SQL query to retrieve the names of employees along with their department names.**

Variations:

1. **Write an SQL query to retrieve the names of employees and the names of the departments they work in, but only for employees in the "Finance" department.**
2. **Write an SQL query to retrieve the department names and the total salary for each department, using a join between Employees and Departments.**
3. **Write an SQL query to get a list of all departments and the employees who work in them. If a department has no employees, it should still appear in the result.**

```
-- Original Question Solution (Inner Join)
SELECT Employees.Name AS EmployeeName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;

-- Variation 1 Solution (Inner Join with WHERE condition)
SELECT Employees.Name AS EmployeeName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
WHERE Departments.DepartmentName = 'Finance';

-- Variation 2 Solution (Inner Join with GROUP BY)
SELECT Departments.DepartmentName, SUM(Employees.Salary) AS TotalSalary
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
GROUP BY Departments.DepartmentName;

-- Variation 3 Solution (Left Join to include departments without employees)
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID;
```

19. **Write an SQL query to retrieve pairs of employees from the same department. Show both employee names in each pair.**

Variations:

1. **Write an SQL query to retrieve pairs of employees from the "HR" department.**
2. **Write an SQL query to find pairs of employees where the first employee has a higher salary than the second employee, within the same department.**
3. **Write an SQL query to find pairs of employees who are from the same department and have different salaries.**

```
-- Original Question Solution (Self Join)
SELECT e1.Name AS Employee1, e2.Name AS Employee2, d.DepartmentName
FROM Employees e1
INNER JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.EmployeeID <
e2.EmployeeID
INNER JOIN Departments d ON e1.DepartmentID = d.DepartmentID;

-- Variation 1 Solution (Self Join with WHERE condition)
SELECT e1.Name AS Employee1, e2.Name AS Employee2, d.DepartmentName
FROM Employees e1
INNER JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.EmployeeID <
e2.EmployeeID
INNER JOIN Departments d ON e1.DepartmentID = d.DepartmentID
WHERE d.DepartmentName = 'HR';

-- Variation 2 Solution (Self Join with Salary comparison)
SELECT e1.Name AS HigherSalaryEmployee, e2.Name AS LowerSalaryEmployee,
d.DepartmentName
FROM Employees e1
INNER JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.Salary >
e2.Salary
INNER JOIN Departments d ON e1.DepartmentID = d.DepartmentID;

-- Variation 3 Solution (Self Join with different salaries)
SELECT e1.Name AS Employee1, e2.Name AS Employee2, d.DepartmentName
FROM Employees e1
INNER JOIN Employees e2 ON e1.DepartmentID = e2.DepartmentID AND e1.Salary <>
e2.Salary
INNER JOIN Departments d ON e1.DepartmentID = d.DepartmentID;
```

20. Write an SQL query to retrieve the names of employees along with the projects they are assigned to, only for employees who have been assigned a project.

Variations:

1. **Write an SQL query to retrieve the names of employees and the projects they are assigned to, only for employees in the "IT" department.**
2. **Write an SQL query to display the department name, employee name, and project name for all employees who are assigned to a project.**
3. **Write an SQL query to retrieve the names of employees and the names of the departments they work in, only for employees with a salary above 55000.**

```
-- Original Question Solution (Inner Join between Employees and Projects)
```

```sql
SELECT Employees.Name AS EmployeeName, Projects.ProjectName
FROM Employees
INNER JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID;

-- Variation 1 Solution (Inner Join with WHERE condition)
SELECT Employees.Name AS EmployeeName, Projects.ProjectName
FROM Employees
INNER JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID
WHERE Employees.DepartmentID = (SELECT DepartmentID FROM Departments
WHERE DepartmentName = 'IT');

-- Variation 2 Solution (Inner Join with three tables)
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName,
Projects.ProjectName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
INNER JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID;

-- Variation 3 Solution (Inner Join with Salary condition)
SELECT Employees.Name AS EmployeeName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
WHERE Employees.Salary > 55000;
```

21. Write an SQL query to retrieve the names of all departments along with the employees who work in them. Show all departments, even if they don't have employees.

Variations:

1. **Write an SQL query to retrieve the names of all employees and the projects they are assigned to, showing all employees even if they are not assigned a project.**
2. **Write an SQL query to list all departments along with the total number of employees in each department. Include departments that have no employees.**
3. **Write an SQL query to retrieve all employees along with their department names. Show all employees, even if they are not assigned to a department.**

```sql
-- Original Question Solution (Left Join between Departments and Employees)
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID;

-- Variation 1 Solution (Left Join between Employees and Projects)
SELECT Employees.Name AS EmployeeName, Projects.ProjectName
FROM Employees
LEFT JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID;
```

```
-- Variation 2 Solution (Left Join with COUNT to include departments without employees)
SELECT Departments.DepartmentName, COUNT(Employees.EmployeeID) AS
EmployeeCount
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID
GROUP BY Departments.DepartmentName;

-- Variation 3 Solution (Left Join to include employees without departments)
SELECT Employees.Name AS EmployeeName, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID;
```

22. **Write an SQL query to retrieve all employees and projects, showing the department name, employee name, and project name. Include departments that have no employees or projects.**

Variations:

1. **Write an SQL query to retrieve all employees and departments, displaying the department name and employee name. Show all departments even if they have no employees, and show all employees even if they are not assigned to a department.**
2. **Write an SQL query to retrieve all departments and projects, displaying department name and project name, including departments with no projects and projects with no departments.**
3. **Write an SQL query to list all employees and their assigned projects, showing all employees even if they are not assigned to a project, and all projects even if they have no assigned employees.**

Example Solution:

(Note: MySQL does not support FULL OUTER JOIN directly, but it can be achieved by combining LEFT JOIN and RIGHT JOIN with UNION)

```
-- Original Question Solution (Full Outer Join for Employees and Projects)
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName,
Projects.ProjectName
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID
LEFT JOIN Projects ON Departments.DepartmentID = Projects.DepartmentID
UNION
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName,
Projects.ProjectName
FROM Departments
RIGHT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID
RIGHT JOIN Projects ON Departments.DepartmentID = Projects.DepartmentID;
```

```sql
-- Variation 1 Solution (Full Outer Join for Employees and Departments)
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName
FROM Departments
LEFT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID
UNION
SELECT Departments.DepartmentName, Employees.Name AS EmployeeName
FROM Departments
RIGHT JOIN Employees ON Departments.DepartmentID = Employees.DepartmentID;

-- Variation 2 Solution (Full Outer Join for Departments and Projects)
SELECT Departments.DepartmentName, Projects.ProjectName
FROM Departments
LEFT JOIN Projects ON Departments.DepartmentID = Projects.DepartmentID
UNION
SELECT Departments.DepartmentName, Projects.ProjectName
FROM Departments
RIGHT JOIN Projects ON Departments.DepartmentID = Projects.DepartmentID;

-- Variation 3 Solution (Full Outer Join for Employees and Projects)
SELECT Employees.Name AS EmployeeName, Projects.ProjectName
FROM Employees
LEFT JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID
UNION
SELECT Employees.Name AS EmployeeName, Projects.ProjectName
FROM Employees
RIGHT JOIN Projects ON Employees.DepartmentID = Projects.DepartmentID;
```

Data for questions - 23-25

Sales

SaleID	EmployeeID	ProductID	SaleDate	Quantity	TotalAmount
1	1	101	15-01-2023	10	500
2	2	102	10-02-2023	5	250
3	3	101	05-03-2023	7	350
4	4	103	25-01-2023	15	750
5	2	104	20-02-2023	12	600
6	1	103	10-03-2023	9	450

7	3	104	18-01-2023	6	300
8	5	101	28-02-2023	20	1000

Departments

DepartmentID	DepartmentName
1	HR
2	Finance
3	IT

23. **Write an SQL query to calculate the total quantity of products sold by each employee.**

Variations:

1. **Write an SQL query to find the total sales amount (TotalAmount) for each product.**
2. **Write an SQL query to get the total quantity of each product sold, displaying only products with total sales above 20 units.**
3. **Write an SQL query to calculate the number of sales made by each employee.**

```
-- Original Question Solution
SELECT EmployeeID, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY EmployeeID;

-- Variation 1 Solution
SELECT ProductID, SUM(TotalAmount) AS TotalSalesAmount
FROM Sales
GROUP BY ProductID;

-- Variation 2 Solution (Using HAVING to filter totals)
SELECT ProductID, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY ProductID
HAVING SUM(Quantity) > 20;

-- Variation 3 Solution (Counting sales by employee)
SELECT EmployeeID, COUNT(SaleID) AS SalesCount
FROM Sales
GROUP BY EmployeeID;
```

24. **Write an SQL query to display each department's total sales amount by joining the**

Sales and Departments tables.

Variations:

1. Write an SQL query to show each department's total quantity of products sold, using a join between Sales and Departments.
2. Write an SQL query to display the name of each employee and their total sales amount, using a join between Sales and Employees.
3. Write an SQL query to show the total amount of sales for each product, but only include products with a total sales amount above 500.

```
-- Original Question Solution
SELECT Departments.DepartmentName, SUM(Sales.TotalAmount) AS TotalSalesAmount
FROM Sales
JOIN Employees ON Sales.EmployeeID = Employees.EmployeeID
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
GROUP BY Departments.DepartmentName;

-- Variation 1 Solution
SELECT Departments.DepartmentName, SUM(Sales.Quantity) AS TotalQuantitySold
FROM Sales
JOIN Employees ON Sales.EmployeeID = Employees.EmployeeID
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
GROUP BY Departments.DepartmentName;

-- Variation 2 Solution
SELECT Employees.Name AS EmployeeName, SUM(Sales.TotalAmount) AS
TotalSalesAmount
FROM Sales
JOIN Employees ON Sales.EmployeeID = Employees.EmployeeID
GROUP BY Employees.Name;

-- Variation 3 Solution (Using HAVING for filtering by total sales)
SELECT ProductID, SUM(TotalAmount) AS TotalSalesAmount
FROM Sales
GROUP BY ProductID
HAVING SUM(TotalAmou\nt) > 500;
```

25. Write an SQL query to find the total sales amount for each employee, but only include sales made in the year 2023 and only show employees with a total sales amount above 700.

Variations:

1. Write an SQL query to find the total quantity of each product sold in 2023, but only display products with total quantities sold above 15.

2. **Write an SQL query to calculate the total sales amount for each department in the "Finance" and "IT" departments, only including departments with sales above 800.**
3. **Write an SQL query to get the total quantity of products sold by each employee in January 2023, displaying only those employees who sold more than 10 units.**

```sql
-- Original Question Solution (Using WHERE for date and HAVING for sales amount)
SELECT EmployeeID, SUM(TotalAmount) AS TotalSalesAmount
FROM Sales
WHERE YEAR(SaleDate) = 2023
GROUP BY EmployeeID
HAVING SUM(TotalAmount) > 700;

-- Variation 1 Solution
SELECT ProductID, SUM(Quantity) AS TotalQuantitySold
FROM Sales
WHERE YEAR(SaleDate) = 2023
GROUP BY ProductID
HAVING SUM(Quantity) > 15;

-- Variation 2 Solution (Joining Departments with WHERE and HAVING)
SELECT Departments.DepartmentName, SUM(Sales.TotalAmount) AS TotalSalesAmount
FROM Sales
JOIN Employees ON Sales.EmployeeID = Employees.EmployeeID
JOIN Departments ON Employees.DepartmentID = Departments.DepartmentID
WHERE Departments.DepartmentName IN ('Finance', 'IT')
GROUP BY Departments.DepartmentName
HAVING SUM(Sales.TotalAmount) > 800;

-- Variation 3 Solution (Filtering by date and quantity)
SELECT EmployeeID, SUM(Quantity) AS TotalQuantitySold
FROM Sales
WHERE SaleDate BETWEEN '2023-01-01' AND '2023-01-31'
GROUP BY EmployeeID
HAVING SUM(Quantity) > 10;
```

Data for questions - 26-28

Orders

OrderID	CustomerID	OrderDate	TotalAmount	Status
1	101	15-01-2023	500	Shipped
2	102	10-02-2023	300	Processing
3	103	05-03-2023	150	Cancelled

4	104	25-01-2023	700	Shipped
5	101	20-02-2023	200	Shipped
6	105	10-03-2023	450	Processing
7	103	18-01-2023	600	Shipped
8	106	28-02-2023	350	Cancelled

26. Write an SQL query to categorize each order in the Orders table as "High" if the TotalAmount is greater than 500, and "Low" otherwise.

Variations:

1. **Write an SQL query to categorize orders into "Early Year" if the OrderDate is before March 1, 2023, and "Late Year" if the OrderDate is on or after March 1, 2023.**
2. **Write an SQL query to label each order as "Completed" if the Status is "Shipped," and "Pending" otherwise.**
3. **Write an SQL query to label each order as "High," "Medium," or "Low" based on TotalAmount, where "High" is above 500, "Medium" is between 300 and 500, and "Low" is below 300.**

```
-- Original Question Solution
SELECT OrderID, TotalAmount,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      ELSE 'Low'
   END AS OrderCategory
FROM Orders;

-- Variation 1 Solution
SELECT OrderID, OrderDate,
   CASE
      WHEN OrderDate < '2023-03-01' THEN 'Early Year'
      ELSE 'Late Year'
   END AS OrderPeriod
FROM Orders;

-- Variation 2 Solution
SELECT OrderID, Status,
   CASE
      WHEN Status = 'Shipped' THEN 'Completed'
      ELSE 'Pending'
   END AS OrderStatus
```

```sql
FROM Orders;

-- Variation 3 Solution
SELECT OrderID, TotalAmount,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      WHEN TotalAmount BETWEEN 300 AND 500 THEN 'Medium'
      ELSE 'Low'
   END AS OrderCategory
FROM Orders;
```

27. **Write an SQL query to count the number of "High" and "Low" orders by customer, using the condition that an order is "High" if TotalAmount is above 500, and "Low" otherwise.**

Variations:

1. **Write an SQL query to count the number of orders in each Status category (e.g., "Completed" for "Shipped" and "Pending" for other statuses) for each customer.**
2. **Write an SQL query to calculate the total sales amount by each customer and categorize each customer as "VIP" if their total spending is above 1000, and "Regular" otherwise.**
3. **Write an SQL query to group orders by "Early Year" and "Late Year" based on OrderDate and count how many orders fall into each category.**

```sql
-- Original Question Solution
SELECT CustomerID,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      ELSE 'Low'
   END AS OrderCategory,
   COUNT(*) AS OrderCount
FROM Orders
GROUP BY CustomerID,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      ELSE 'Low'
   END;

-- Variation 1 Solution
SELECT CustomerID,
   CASE
      WHEN Status = 'Shipped' THEN 'Completed'
      ELSE 'Pending'
   END AS OrderStatus,
   COUNT(*) AS StatusCount
```

```
FROM Orders
GROUP BY CustomerID,
   CASE
      WHEN Status = 'Shipped' THEN 'Completed'
      ELSE 'Pending'
   END;

-- Variation 2 Solution
SELECT CustomerID,
   CASE
      WHEN SUM(TotalAmount) > 1000 THEN 'VIP'
      ELSE 'Regular'
   END AS CustomerCategory,
   SUM(TotalAmount) AS TotalSpent
FROM Orders
GROUP BY CustomerID;

-- Variation 3 Solution
SELECT
   CASE
      WHEN OrderDate < '2023-03-01' THEN 'Early Year'
      ELSE 'Late Year'
   END AS OrderPeriod,
   COUNT(*) AS OrderCount
FROM Orders
GROUP BY
   CASE
      WHEN OrderDate < '2023-03-01' THEN 'Early Year'
      ELSE 'Late Year'
   END;
```

28. **Assume there is another table Customers with the following structure:**

CustomerID	CustomerName
101	John Doe
102	Jane Smith
103	Sam Wilson
104	Alice Brown
105	Tom Davis
106	Kate Evans

Write an SQL query to join Orders and Customers tables and retrieve each customer's name, along with the count of "High" and "Low" orders they have placed, where "High" orders have TotalAmount above 500.

Variations:

1. Write an SQL query to join Orders and Customers, displaying each customer’s name and the total sales they generated, and categorize them as "VIP" if total sales exceed 1000 and "Regular" otherwise.
2. Write an SQL query to join Orders and Customers, showing each customer’s name, and the total count of orders, and categorize them as "Frequent" if they placed more than 2 orders and "Occasional" otherwise.
3. Write an SQL query to join Orders and Customers, displaying each customer's name, and total amount spent on "Completed" orders (Status = "Shipped"), and label customers as "High Spender" if the total spent on "Completed" orders is above 500.

```sql
-- Original Question Solution
SELECT Customers.CustomerName,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      ELSE 'Low'
   END AS OrderCategory,
   COUNT(*) AS OrderCount
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
GROUP BY Customers.CustomerName,
   CASE
      WHEN TotalAmount > 500 THEN 'High'
      ELSE 'Low'
   END;

-- Variation 1 Solution
SELECT Customers.CustomerName,
   CASE
      WHEN SUM(TotalAmount) > 1000 THEN 'VIP'
      ELSE 'Regular'
   END AS CustomerCategory,
   SUM(TotalAmount) AS TotalSpent
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
GROUP BY Customers.CustomerName;

-- Variation 2 Solution
SELECT Customers.CustomerName,
   COUNT(OrderID) AS OrderCount,
   CASE
      WHEN COUNT(OrderID) > 2 THEN 'Frequent'
```

```
        ELSE 'Occasional'
    END AS CustomerType
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
GROUP BY Customers.CustomerName;

-- Variation 3 Solution
SELECT Customers.CustomerName,
    SUM(CASE WHEN Status = 'Shipped' THEN TotalAmount ELSE 0 END) AS
CompletedTotal,
    CASE
        WHEN SUM(CASE WHEN Status = 'Shipped' THEN TotalAmount ELSE 0 END) > 500
THEN 'High Spender'
        ELSE 'Regular Spender'
    END AS SpendingCategory
FROM Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
GROUP BY Customers.CustomerName;
```

Data for questions 29 - 33

Table: Orders

OrderID	CustomerID	OrderDate	ProductID	Quantity	TotalAmount
1	201	15-01-2023	301	3	150
2	202	10-02-2023	302	5	300
3	203	05-03-2023	301	2	100
4	204	25-01-2023	303	4	200
5	201	20-02-2023	302	6	360
6	205	10-03-2023	304	1	90
7	203	18-01-2023	301	8	400
8	206	28-02-2023	302	2	120

Table: Customers

CustomerID	CustomerName	City
201	John Doe	New York
202	Jane Smith	Los Angeles

203	Sam Wilson	New York
204	Alice Brown	Chicago
205	Tom Davis	Miami
206	Kate Evans	Chicago

Table: Products

ProductID	ProductName	Price
301	Widget A	50
302	Widget B	60
303	Widget C	50
304	Widget D	90

29. Write a query to find the total sales amount for each customer, along with the number of unique products they ordered.

```
WITH CustomerSales AS (
  SELECT
    Orders.CustomerID,
    Customers.CustomerName,
    SUM(Orders.TotalAmount) AS TotalSales,
    COUNT(DISTINCT Orders.ProductID) AS UniqueProductsOrdered
  FROM Orders
  JOIN Customers ON Orders.CustomerID = Customers.CustomerID
  GROUP BY Orders.CustomerID, Customers.CustomerName
)
SELECT * FROM CustomerSales;
```

30. Write a query to categorize each customer as a "VIP" if their total spending is above 400, and "Regular" otherwise, including the customer's city.

```
WITH CustomerSpending AS (
  SELECT
    Orders.CustomerID,
    Customers.CustomerName,
    Customers.City,
    SUM(Orders.TotalAmount) AS TotalSpent
  FROM Orders
  JOIN Customers ON Orders.CustomerID = Customers.CustomerID
  GROUP BY Orders.CustomerID, Customers.CustomerName, Customers.City
)
SELECT
   CustomerName,
   City,
   TotalSpent,
   CASE
      WHEN TotalSpent > 400 THEN 'VIP'
      ELSE 'Regular'
   END AS CustomerCategory
FROM CustomerSpending;
```

31. **Write a query to find each product's name, and total quantity sold, and categorize it as "High Demand" if the total quantity sold is greater than 10, otherwise "Low Demand". Only include products with total sales amounting to more than 500.**

```
WITH ProductSales AS (
   SELECT
      Products.ProductName,
      Orders.ProductID,
      SUM(Orders.Quantity) AS TotalQuantitySold,
      SUM(Orders.TotalAmount) AS TotalSalesAmount
   FROM Orders
   JOIN Products ON Orders.ProductID = Products.ProductID
   GROUP BY Orders.ProductID, Products.ProductName
)
SELECT
   ProductName,
   TotalQuantitySold,
   CASE
      WHEN TotalQuantitySold > 10 THEN 'High Demand'
      ELSE 'Low Demand'
   END AS DemandCategory
FROM ProductSales
WHERE TotalSalesAmount > 500;
```

32. **Write a query to find the total sales amount for each city and categorize each city as "Key Market" if total sales exceed 500, and "Emerging Market" otherwise.**

```
WITH CitySales AS (
   SELECT
      Customers.City,
      SUM(Orders.TotalAmount) AS TotalCitySales
   FROM Orders
   JOIN Customers ON Orders.CustomerID = Customers.CustomerID
```

```
    GROUP BY Customers.City
)
SELECT
    City,
    TotalCitySales,
    CASE
        WHEN TotalCitySales > 500 THEN 'Key Market'
        ELSE 'Emerging Market'
    END AS MarketCategory
FROM CitySales;
```

33. **Write a query to calculate the number of "VIP" and "Regular" customers for each city, where a customer is considered "VIP" if their total spending exceeds 400.**

```
WITH CustomerSpending AS (
    SELECT
        Orders.CustomerID,
        Customers.CustomerName,
        Customers.City,
        SUM(Orders.TotalAmount) AS TotalSpent
    FROM Orders
    JOIN Customers ON Orders.CustomerID = Customers.CustomerID
    GROUP BY Orders.CustomerID, Customers.CustomerName, Customers.City
),
CustomerCategory AS (
    SELECT
        CustomerName,
        City,
        CASE
            WHEN TotalSpent > 400 THEN 'VIP'
            ELSE 'Regular'
        END AS CustomerCategory
    FROM CustomerSpending
)
SELECT
    City,
    CustomerCategory,
    COUNT(*) AS CustomerCount
FROM CustomerCategory
GROUP BY City, CustomerCategory;
```

Data for questions - 34-35

Table: Products

ProductID	ProductName	Category	Price
1	Laptop	Electronics	800
2	Smartphone	Electronics	600
3	Desk Chair	Furniture	120
4	Desk	Furniture	300
5	Headphones	Electronics	150
6	Bookshelf	Furniture	200
7	Microwave	Appliances	250
8	Blender	Appliances	100

Table: Sales

SaleID	ProductID	SaleDate	Quantity	TotalAmount
1	1	10-01-2023	5	4000
2	2	15-01-2023	3	1800
3	3	10-02-2023	4	480
4	4	20-02-2023	2	600
5	5	05-03-2023	6	900
6	6	10-03-2023	1	200
7	7	15-03-2023	3	750
8	8	20-03-2023	5	500

34. Write a query to find all products in the Electronics category whose price is higher than the average price of all Furniture products.

Variations:

1. **Write a query to find all furniture products whose prices are lower than the average price of all electronics products.**
2. **Write a query to find all products in the Appliances category whose price is higher than the average price of all products across all categories.**

3. **Write a query to find all products in the Furniture category whose price is higher than the average price of all products within the Furniture category itself.**

```
-- Original Question Solution
SELECT ProductName, Price
FROM Products
WHERE Category = 'Electronics' AND Price > (
    SELECT AVG(Price)
    FROM Products
    WHERE Category = 'Furniture'
);

-- Variation 1 Solution
SELECT ProductName, Price
FROM Products
WHERE Category = 'Furniture' AND Price < (
    SELECT AVG(Price)
    FROM Products
    WHERE Category = 'Electronics'
);

-- Variation 2 Solution
SELECT ProductName, Price
FROM Products
WHERE Category = 'Appliances' AND Price > (
    SELECT AVG(Price)
    FROM Products
);

-- Variation 3 Solution
SELECT ProductName, Price
FROM Products
WHERE Category = 'Furniture' AND Price > (
    SELECT AVG(Price)
    FROM Products
    WHERE Category = 'Furniture'
);
```

35. **Write a query to retrieve each sale's SaleID, ProductID, and TotalAmount, and mark it as "Above Average" if its TotalAmount is above the average TotalAmount for that product, or "Below Average" otherwise.**

Variations:

1. **Write a query to retrieve each sale's SaleID, ProductID, and Quantity, and label it as "High Volume" if the quantity sold is above the average quantity for that product, and**

"Low Volume" otherwise.

2. **Write a query to retrieve each sale's SaleID, ProductID, and TotalAmount, and label it as "High Sale" if its TotalAmount is above the average TotalAmount for all sales of Electronics products, and "Low Sale" otherwise.**
3. **Write a query to retrieve each sale's SaleID, ProductID, and TotalAmount, and label it as "High Sale" if its TotalAmount is above the average TotalAmount for sales made after 2023-02-01, and "Low Sale" otherwise.**

```sql
-- Original Question Solution (Correlated Subquery)
SELECT SaleID, ProductID, TotalAmount,
  CASE
    WHEN TotalAmount > (
      SELECT AVG(TotalAmount)
      FROM Sales AS S2
      WHERE S2.ProductID = Sales.ProductID
    ) THEN 'Above Average'
    ELSE 'Below Average'
  END AS SaleCategory
FROM Sales;

-- Variation 1 Solution (Correlated Subquery with Quantity comparison)
SELECT SaleID, ProductID, Quantity,
  CASE
    WHEN Quantity > (
      SELECT AVG(Quantity)
      FROM Sales AS S2
      WHERE S2.ProductID = Sales.ProductID
    ) THEN 'High Volume'
    ELSE 'Low Volume'
  END AS VolumeCategory
FROM Sales;

-- Variation 2 Solution (Correlated Subquery with Category filter)
SELECT SaleID, ProductID, TotalAmount,
  CASE
    WHEN TotalAmount > (
      SELECT AVG(TotalAmount)
      FROM Sales AS S2
      JOIN Products ON S2.ProductID = Products.ProductID
      WHERE Products.Category = 'Electronics'
    ) THEN 'High Sale'
    ELSE 'Low Sale'
  END AS SaleCategory
FROM Sales;

-- Variation 3 Solution (Correlated Subquery with Date filter)
SELECT SaleID, ProductID, TotalAmount,
  CASE
    WHEN TotalAmount > (
      SELECT AVG(TotalAmount)
      FROM Sales AS S2
      WHERE S2.SaleDate > '2023-02-01'
    ) THEN 'High Sale'
    ELSE 'Low Sale'
  END AS SaleCategory
FROM Sales;
```

Data for questions -

Table: Orders

Column	Data Type
OrderID	INT
CustomerID	INT
ProductID	INT
OrderDate	DATE
Quantity	INT
TotalAmount	DECIMAL(10, 2)
Status	VARCHAR(20)

Table: Customers

Column	Data Type
CustomerID	INT
CustomerName	VARCHAR(50)
City	VARCHAR(50)

36. Create a stored procedure named GetCustomerOrders that accepts a CustomerID parameter and returns all orders for that customer.

```
CREATE PROCEDURE GetCustomerOrders(IN custID INT)
BEGIN
  SELECT OrderID, ProductID, Quantity, TotalAmount, Status
```

```
   FROM Orders
   WHERE CustomerID = custID;
END;
```

Variations:

1. **Modify the GetCustomerOrders procedure to include the customer's name from the Customers table.**

```
CREATE PROCEDURE GetCustomerOrdersWithName(IN custID INT)
BEGIN
   SELECT Orders.OrderID, Customers.CustomerName, Orders.ProductID,
Orders.Quantity, Orders.TotalAmount, Orders.Status
   FROM Orders
   JOIN Customers ON Orders.CustomerID = Customers.CustomerID
   WHERE Orders.CustomerID = custID;
END;
```

2. **Update the stored procedure to accept an additional ProductID parameter so that it returns only orders for that customer and product.**

```
CREATE PROCEDURE GetCustomerOrdersForProduct(IN custID INT, IN prodID INT)
BEGIN
   SELECT OrderID, ProductID, Quantity, TotalAmount, Status
   FROM Orders
   WHERE CustomerID = custID AND ProductID = prodID;
END;
```

3. **Modify the stored procedure to return only orders with a TotalAmount greater than 200.**

```
CREATE PROCEDURE GetCustomerOrdersAboveAmount(IN custID INT, IN minAmount
DECIMAL)
BEGIN
   SELECT OrderID, ProductID, Quantity, TotalAmount, Status
   FROM Orders
   WHERE CustomerID = custID AND TotalAmount > minAmount;
END;
```

Dataset based questions

Marketing Campaign Effectiveness:

This dataset will help you analyze the effectiveness of marketing campaigns in driving customer engagement and purchases.

Instructions: The dataset needs to be imported into your local MySQL Workbench and then you have to perform all the questions mentioned below.

Schema:

Customers Table

- **customer_id (Primary Key):** Unique identifier for the customer.
- **signup_date:** Date the customer signed up.
- **region:** Geographic region of the customer.
- **age_group:** Age group of the customer (e.g., 18-25, 26-35).

Campaigns Table

- **campaign_id (Primary Key):** Unique identifier for the campaign.
- **campaign_type:** Type of campaign (e.g., Email, Social Media, TV Ad).
- **start_date:** Start date of the campaign.
- **end_date:** End date of the campaign.

Engagements Table

- **engagement_id (Primary Key):** Unique identifier for the engagement.
- **customer_id (Foreign Key):** Links to the Customers table.
- **campaign_id (Foreign Key):** Links to the Campaigns table.
- **engagement_type:** Type of engagement (e.g., Click, Purchase).
- **engagement_date:** Date of the engagement.

Questions:

1. **Find Campaigns with the Highest "Click" Engagements**
   Write a query to find the top 5 campaigns with the most "Click" engagements. Use the engagement_type column to filter the results.

Solution:

```
SELECT campaign_id, COUNT(*) AS click_count
FROM engagements
WHERE engagement_type = 'Click'
GROUP BY campaign_id
ORDER BY click_count DESC
LIMIT 5;
```

2. **Calculate the Total Number of Customers per Age Group**
   Write a query to calculate the number of customers in each age group.

Solution:

```
SELECT age_group, COUNT(*) AS total_customers
FROM customers
GROUP BY age_group;
```

3. **Find Campaigns That Ended Before a Specific Date**
   Write a query to find all campaigns that ended before '2023-06-30'.

Solution:

```
SELECT campaign_id, campaign_type, end_date
FROM campaigns
WHERE end_date < '2023-06-30';
```

4. **Merge Data to Find Engagement Counts by Campaign Type**
   Write a query to calculate the total number of engagements for each campaign_type by joining the Campaigns and Engagements tables.

Solution:

```
SELECT c.campaign_type, COUNT(e.engagement_id) AS total_engagements
FROM campaigns c
JOIN engagements e ON c.campaign_id = e.campaign_id
GROUP BY c.campaign_type
ORDER BY total_engagements DESC;
```

5. **Identify Regions with No Engagements**
   Write a query to find regions from the Customers Table where no engagements have been recorded.

Solution:

```
SELECT DISTINCT c.region
FROM customers c
LEFT JOIN engagements e ON c.customer_id = e.customer_id
WHERE e.customer_id IS NULL;
```

6. **Impute Missing Campaign End Dates**
   Write a query to update all missing end_date values in the Campaigns Table to 30 days

after the start_date.

Solution:

```
SELECT DISTINCT c.region
FROM customers c
LEFT JOIN engagements e ON c.customer_id = e.customer_id
WHERE e.customer_id IS NULL;
```

7. **Top Customers by Total Engagements**
   Write a query to identify the top 3 customers with the highest number of engagements.

Solution:

```
SELECT customer_id, COUNT(*) AS total_engagements
FROM engagements
GROUP BY customer_id
ORDER BY total_engagements DESC
LIMIT 3;
```

8. **Calculate Monthly Signups**
   Write a query to count the number of customer signups grouped by month.

Solution:

```
SELECT MONTH(signup_date) AS signup_month, COUNT(*) AS total_signups
FROM customers
GROUP BY signup_month
ORDER BY signup_month;
```

9. **Identify Campaigns with No Engagements**
   Write a query to find all campaigns that have no recorded engagements in the Engagements Table.

Solution:

```
SELECT c.campaign_id, c.campaign_type
FROM campaigns c
LEFT JOIN engagements e ON c.campaign_id = e.campaign_id
WHERE e.campaign_id IS NULL;
```

10. **Revenue Contribution by Campaign**
    Assume each engagement contributes $10 in revenue. Write a query to calculate the

total revenue generated by each campaign.

Solution:

```
SELECT e.campaign_id, COUNT(*) * 10 AS total_revenue
FROM engagements e
GROUP BY e.campaign_id
ORDER BY total_revenue DESC;
```