**By-- AJAY R**

**8825963258**

**Customer Relationship Management (CRM) System**

## Abstract

The CRM system is designed to help businesses efficiently manage customer interactions, sales opportunities, and marketing campaigns. It provides a structured database to store customer details, track leads, manage support tickets, and analyze sales performance. This project implements an SQL-based relational database to ensure smooth customer management processes.

## Introduction

Customer Relationship Management (CRM) is an essential business strategy for improving customer interactions and streamlining operations. This project aims to develop an SQL-based CRM system that enables organizations to track and manage customers, sales, leads, support tickets, and marketing campaigns.

The system links customers to orders, sales opportunities, and support tickets, ensuring a comprehensive view of customer interactions. Employees handle assigned sales and support cases, improving service quality.

By integrating structured data relationships, the CRM enhances customer engagement, optimizes sales processes, and provides insights into marketing performance, driving business growth

## System Architecture

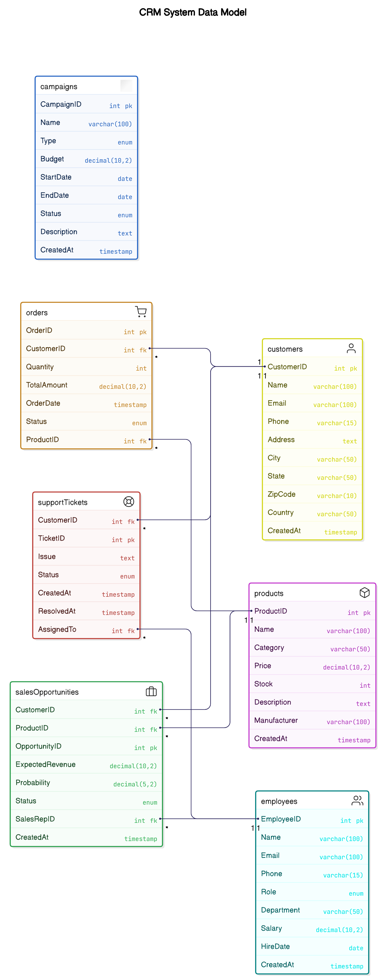
5.1 Database Design

The database follows a **relational model** with tables for Customers, Leads, Sales Opportunities, Support Tickets, and Campaigns.

5.2 Database Schema

**Entities & Relationships**

1. **Customers** (One-to-Many) → **SalesOpportunities**
2. **Customers** (One-to-Many) → **SupportTickets**
3. **Customers** (One-to-Many) → **Orders**
4. **Orders** (Many-to-One) → **Products**
5. **SalesOpportunities** (Many-to-One) → **Products**
6. **SupportTickets** (Many-to-One) → **Employees** (Assigned Support Rep)
7. **SalesOpportunities** (Many-to-One) → **Employees** (Assigned Sales Rep)
8. **Campaigns** (Independent, tracks marketing activities)

**ER Diagram (Entity-Relationship Model)**

## 6. Implementation

6.1 Technologies Used

* **Database:** MySQL
* **Query Language:** SQL
* **Development Tools:** MySQL Workbench
* **Chatgpt**: For generating a large dataset

6.2 Creating the Database

CREATE DATABASE CRM\_System;

USE CRM\_System;

6.3 Table Creation

Table 1 Customers

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

Phone VARCHAR(15),

Address TEXT,

City VARCHAR(50),

State VARCHAR(50),

ZipCode VARCHAR(10),

Country VARCHAR(50),

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Table 2 Employees

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE NOT NULL,

Phone VARCHAR(15),

Role ENUM('Sales', 'Support', 'Marketing'),

Department VARCHAR(50),

Salary DECIMAL(10,2),

HireDate DATE,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Table 3 Products

CREATE TABLE Products (

ProductID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Category VARCHAR(50),

Price DECIMAL(10,2),

Stock INT,

Description TEXT,

Manufacturer VARCHAR(100),

WarrantyPeriod INT, -- In months

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Table 4 SalesOpportunities

CREATE TABLE SalesOpportunities (

OpportunityID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

ProductID INT,

ExpectedRevenue DECIMAL(10,2),

Probability DECIMAL(5,2),

Status ENUM('New', 'Negotiation', 'Closed-Won', 'Closed-Lost'),

SalesRepID INT,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE SET NULL,

FOREIGN KEY (SalesRepID) REFERENCES Employees(EmployeeID) ON DELETE SET NULL

);

Table 5 SupportTickets

CREATE TABLE SupportTickets (

TicketID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT,

Issue TEXT,

Status ENUM('Open', 'In Progress', 'Resolved', 'Closed'),

AssignedTo INT,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

ResolvedAt TIMESTAMP NULL,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,

FOREIGN KEY (AssignedTo) REFERENCES Employees(EmployeeID) ON DELETE SET NULL

);

Table 6 Products Campaigns

CREATE TABLE Campaigns (

CampaignID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Type ENUM('Email', 'Social Media', 'Advertisement', 'Referral'),

Budget DECIMAL(10,2),

StartDate DATE,

EndDate DATE,

Status ENUM('Planned', 'Active', 'Completed'),

Description TEXT,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Table 7 orders

CREATE TABLE Orders (  
 OrderID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT, ProductID INT, Quantity INT,

TotalAmount DECIMAL(10,2),

OrderDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

Status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled'),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE SET NULL

);

6.4 Sample Data Insertion

To insert values into the tables, you need to use the INSERT INTO SQL statement for each table.

Table 1 Customers

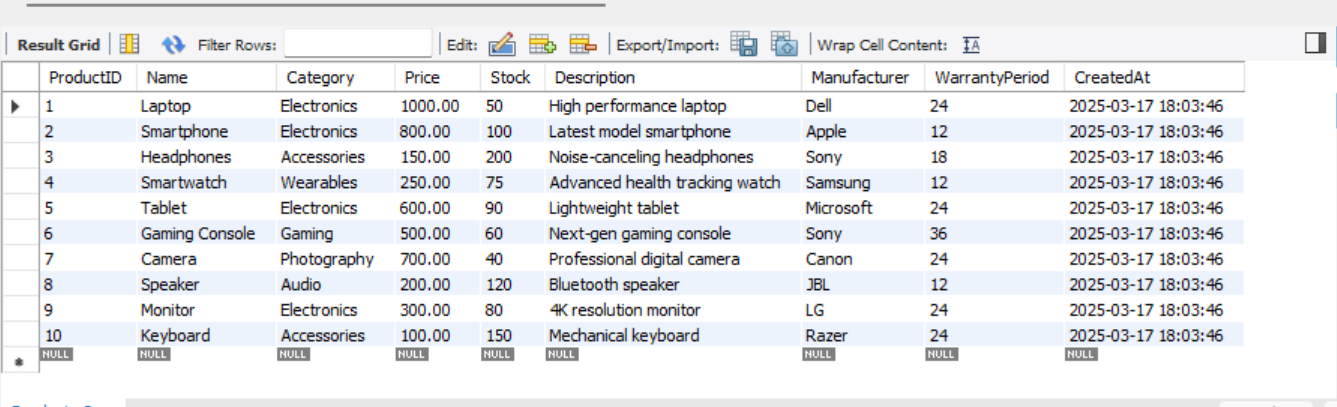


Table 2 Employees

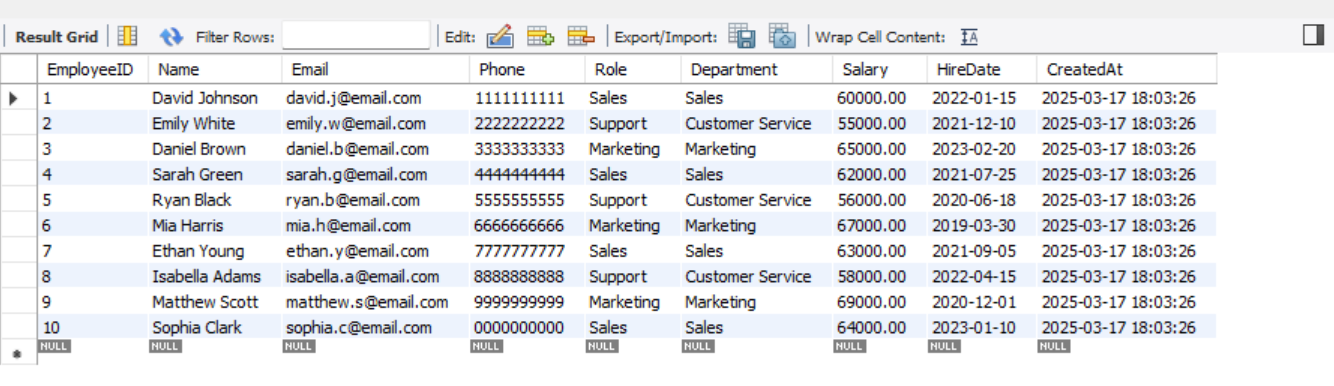


Table 3 Products

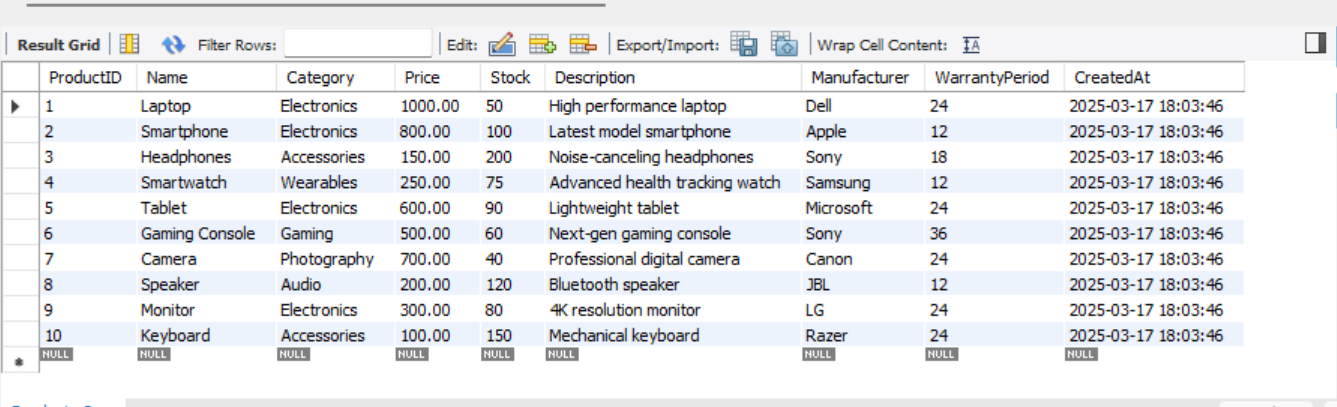


Table 4 SalesOpportunities

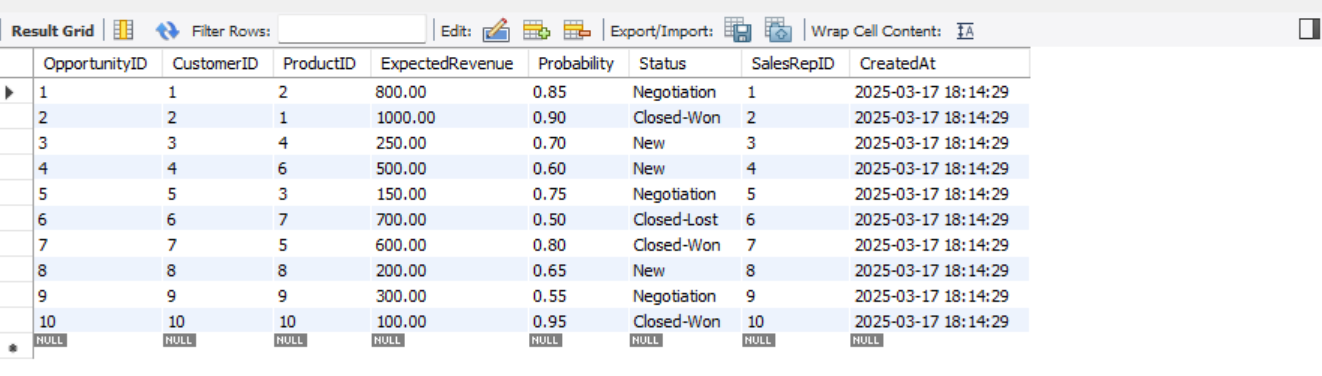


Table 5 SupportTickets

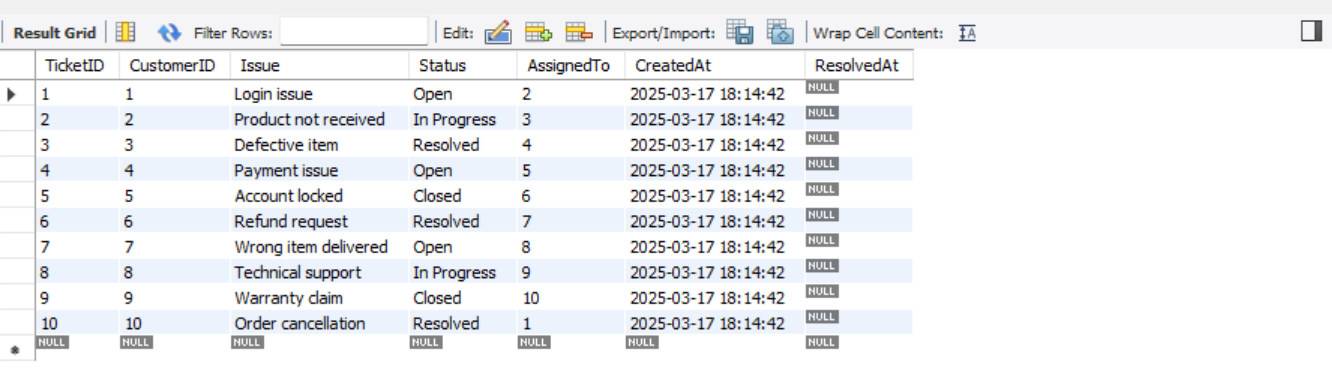


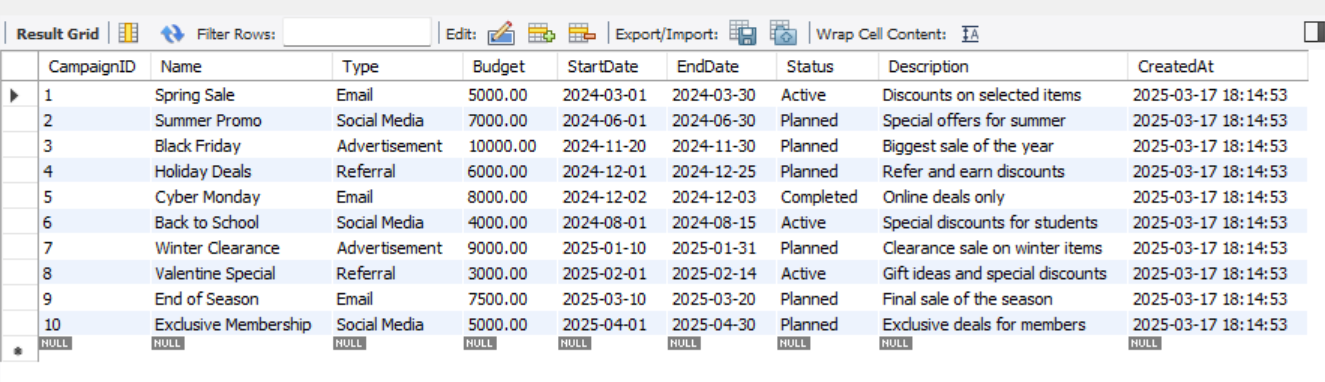
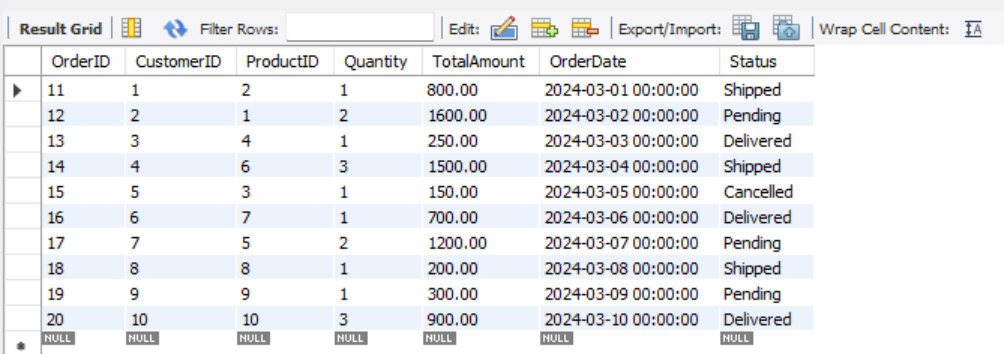
Table 6 Products Campaigns

Table 7 orders



6.5 Query Examples

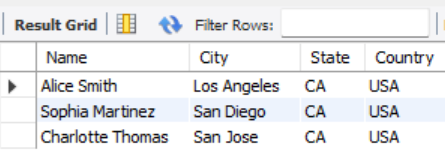
-- 1. WHERE Clause (Filter customers in California)

SELECT Name, City, State, Country

FROM Customers

WHERE State = 'CA';

Output:



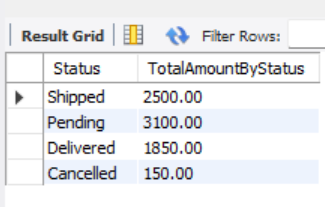
-- 2. GROUP BY Clause (group orders by their Status and get the total amount for each status:)

SELECT Status, SUM(TotalAmount) AS TotalAmountByStatus

FROM Orders

GROUP BY Status;

Output:



--3. HAVING Clause (Filter orders with a total amount greater than 1000)

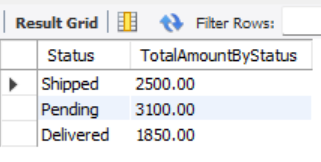
SELECT Status, SUM(TotalAmount) AS TotalAmountByStatus

FROM Orders

GROUP BY Status

HAVING SUM(TotalAmount) > 1000;

Output:

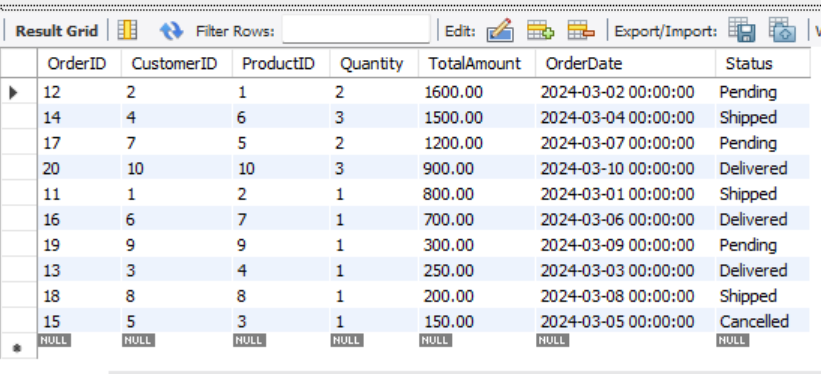


-- 4. ORDER BY Clause (Sort orders by the total amount in descending order)

SELECT \* FROM Orders

ORDER BY TotalAmount DESC;

Output:

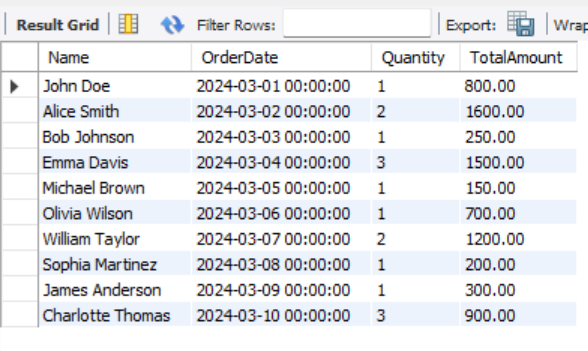


-- 5. INNER JOIN (Get details of customers with their orders)

SELECT Customers.Name, Orders.OrderDate, Orders.Quantity, Orders.TotalAmount

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;



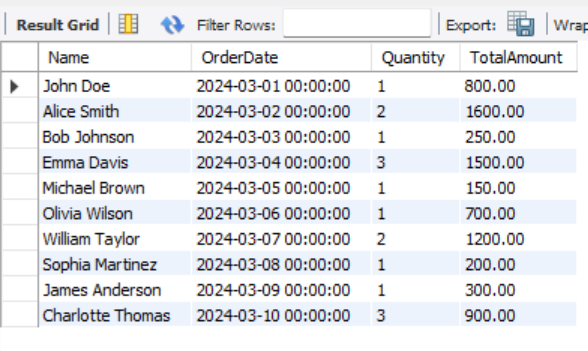
-- 6. LEFT OUTER JOIN (Get all customers and their orders, even if they haven’t placed any)

SELECT Customers.Name, Orders.OrderDate, Orders.Quantity, Orders.TotalAmount

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

Ouput:



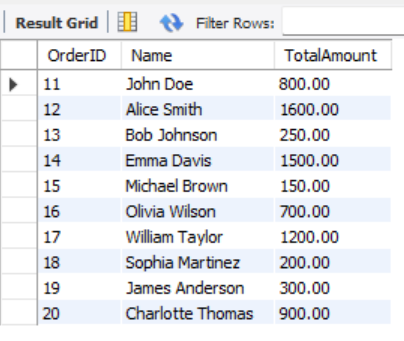
-- 7. RIGHT OUTER JOIN (Get all orders and their respective customers, even if no customer data exists)

SELECT Orders.OrderID, Customers.Name, Orders.TotalAmount

FROM Orders

RIGHT JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

Output:



-- 8. CROSS JOIN (Get all combinations of products and orders) with limit

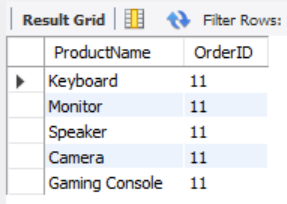
SELECT Products.Name AS ProductName, Orders.OrderID

FROM Products

CROSS JOIN Orders

LIMIT 5;

Output:



-- 9. Subquery (Non-correlated) (Get customers who have placed orders for products costing more than $500)

SELECT Name

FROM Customers

WHERE CustomerID IN (

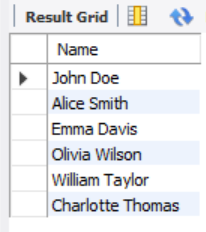
SELECT CustomerID

FROM Orders

WHERE TotalAmount > 500

);

Output:



-- 10. Subquery (Correlated) the orders with a TotalAmount greater than the average for that specific customer:

SELECT OrderID, CustomerID, TotalAmount

FROM Orders O

WHERE TotalAmount > (

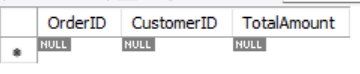
SELECT AVG(TotalAmount)

FROM Orders

WHERE CustomerID = O.CustomerID

);

Output:

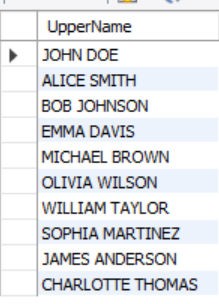


-- 11. Single Row Function (UPPER) (Get all customers' names in uppercase)

SELECT UPPER(Name) AS UpperName

FROM Customers;

Output:

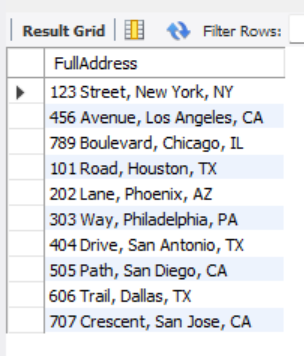


-- 12. Single Row Function (CONCAT) (Get the full address of customers by combining the street, city, and state)

SELECT CONCAT(Address, ', ', City, ', ', State) AS FullAddress

FROM Customers;

Output:



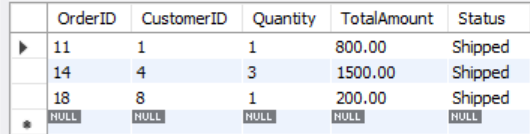
-- 13. to find the orders whose Status contains a specific pattern, for example, all orders where the Status contains the word "Shipped".

SELECT OrderID, CustomerID, Quantity, TotalAmount, Status

FROM Orders

WHERE Status LIKE '%Shipped%';

Output:

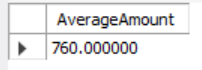


-- 14. Multi-row Function (AVG) (Find the average total amount spent on orders)

SELECT AVG(TotalAmount) AS AverageAmount

FROM Orders;

Output:



-- 15. Complex Query (Join + Aggregate + HAVING) (Find sales opportunities with an expected revenue above $500, grouped by product)

SELECT Products.Name, SUM(SalesOpportunities.ExpectedRevenue) AS TotalRevenue

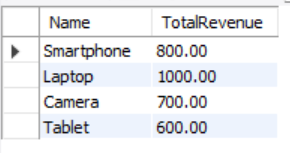
FROM SalesOpportunities

INNER JOIN Products ON SalesOpportunities.ProductID = Products.ProductID

GROUP BY Products.Name

HAVING SUM(SalesOpportunities.ExpectedRevenue) > 500;

Output:



## 7. Results & Discussion

The Customer Relationship Management (CRM) system was successfully implemented with seven interrelated tables, enabling efficient data storage and retrieval. The **Customers** table ensures seamless tracking of customer details, while the **SalesOpportunities** table dynamically monitors and updates potential sales prospects. The integration of **SupportTickets** allows for effective resolution of customer complaints, improving overall service quality.

Furthermore, the **Campaigns** module offers valuable insights into customer engagement and marketing effectiveness. The project also includes 15 SQL queries covering key **clauses** (WHERE, GROUP BY, HAVING, ORDER BY), **joins** (INNER JOIN, OUTER JOIN, CROSS JOIN), **subqueries** (correlated and non-correlated), and **functions** (single-row and multi-row) to extract meaningful insights from the data.