***DBMS MINI PROJECT***

***GROUP 2***

**Title:** Online Shopping System

**PROBLEM STATEMENT**: **Develop a database for managing products, customers, and purchases.**

**DESCRIPTION: The system should store product listings, customer details, and purchase transactions**.

**GitHub Repository Link:** https://github.com/ayush2005soumya/Online-Shopping-System/edit/main/Online\_Shopping\_System

**Group Members:**

|  |  |
| --- | --- |
| **NAME** | **ROLL NO.** |
| CHINTHAN N | 23CS8006 |
| AYUSH CHATTERJEE | 23CS8007 |
| PILLI RAJESH | 23CS8008 |
| DAWIPAYAN DAS | 23CS8009 |
| BILL SAHA | 23CS8010 |

**Table Creation**

CREATE TABLE Products (

    ProductID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(100),

    Price DECIMAL(10, 2),

    Stock INT,

    CategoryID INT

);

CREATE TABLE Customers (

    CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(100),

    Email VARCHAR(100),

    Address VARCHAR(255),

    Phone VARCHAR(20)

);

CREATE TABLE Orders (

    OrderID INT AUTO\_INCREMENT PRIMARY KEY,

    CustomerID INT,

    ProductID INT,

    OrderDate DATE,

    Quantity INT,

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

    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

commit;

**Insertion of Data**

INSERT INTO Products (Name, Price, Stock, CategoryID) VALUES

('Samsung Galaxy S24', 79999.00, 50, 1),

('Apple iPhone 15 Pro', 134999.00, 40, 1),

('HP Pavilion Gaming Laptop', 74999.00, 30, 2),

('Dell XPS 13', 129999.00, 25, 2),

('Asus ROG Strix G15', 94999.00, 20, 2),

('Sony Bravia 55-inch 4K TV', 82999.00, 15, 3),

('LG OLED 65-inch TV', 144999.00, 10, 3),

('Samsung 1.5 Ton Split AC', 45999.00, 35, 4),

('Voltas 1 Ton Window AC', 27999.00, 40, 4),

('Blue Star 1.2 Ton Inverter AC', 52999.00, 25, 4),

('Sony WH-1000XM5 Headphones', 34999.00, 50, 5),

('JBL Boombox 3 Speaker', 31999.00, 40, 5),

('Apple AirPods Pro', 24999.00, 60, 5),

('Samsung Galaxy Buds2 Pro', 17999.00, 55, 5),

('Boat Rockerz 450 Headphones', 2499.00, 100, 5),

('Canon EOS R5 Camera', 319999.00, 5, 6),

('Nikon Z7 Mirrorless Camera', 259999.00, 8, 6),

('GoPro Hero 12', 49999.00, 20, 6),

('DJI Mini 3 Drone', 74999.00, 15, 6),

('Sony Alpha 7 IV Camera', 249999.00, 7, 6);

INSERT INTO Customers (Name, Email, Address, Phone) VALUES

('Arjun Mehta', 'arjun@mehta.com', 'Andheri West, Mumbai', '9998887777'),

('Rohit Sharma', 'rohit@sharma.com', 'Banjara Hills, Hyderabad', '9997776666'),

('Aman Verma', 'aman@verma.com', 'Lajpat Nagar, Delhi', '8887776665'),

('Karan Singh', 'karan@singh.com', 'Shastri Nagar, Lucknow', '7776665554'),

('Vikram Rao', 'vikram@rao.com', 'BTM Layout, Bangalore', '6665554443'),

('Rajat Malhotra', 'rajat@malhotra.com', 'Salt Lake City, Kolkata', '6664443332'),

('Nikhil Joshi', 'nikhil@joshi.com', 'Sadar Bazar, Pune', '5554443331'),

('Priya Iyer', 'priya@iyer.com', 'Anna Nagar, Chennai', '7778889999'),

('Ankit Mishra', 'ankit@mishra.com', 'Aliganj, Lucknow', '1234567890'),

('Raghav Kapoor', 'raghav@kapoor.com', 'Sector 18, Noida', '2223334445'),

('Devansh Rao', 'devansh@rao.com', 'Rajajinagar, Bangalore', '3334445556'),

('Kunal Desai', 'kunal@desai.com', 'Wakad, Pune', '1112223334'),

('Harshvardhan Reddy', 'harsh@reddy.com', 'Jubilee Hills, Hyderabad', '9996663331'),

('Aditya Chauhan', 'aditya@chauhan.com', 'Vasant Kunj, Delhi', '5557778889'),

('Manav Pillai', 'manav@pillai.com', 'Panampilly Nagar, Kochi', '7778881112'),

('Rohan Shetty', 'rohan@shetty.com', 'Santacruz, Mumbai', '6667778883'),

('Kabir Khanna', 'kabir@khanna.com', 'Sector 22, Chandigarh', '2224446668'),

('Arvind Nair', 'arvind@nair.com', 'Technopark, Trivandrum', '1115559992'),

('Siddharth Das', 'siddharth@das.com', 'Lake Town, Kolkata', '8885554441'),

('Naveen Iqbal', 'naveen@iqbal.com', 'Gachibowli, Hyderabad', '5465545555'),

('Mehul Shah', 'mehul@shah.com', 'Bodakdev, Ahmedabad', '1122334455'),

('Rahul Bansal', 'rahul@bansal.com', 'Vaishali Nagar, Jaipur', '9988776655'),

('Tejas Patil', 'tejas@patil.com', 'Aundh, Pune', '4433221100'),

('Varun Sinha', 'varun@sinha.com', 'Dhanbad Main Road, Jharkhand', '1029384756'),

('Neeraj Kapoor', 'neeraj@kapoor.com', 'Model Town, Delhi', '9990001112'),

('Yash Tiwari', 'yash@tiwari.com', 'Ashok Nagar, Kanpur', '5551234567'),

('Deepak Rana', 'deepak@rana.com', 'Sector 21, Chandigarh', '6662345678'),

('Saurav Mehra', 'saurav@mehra.com', 'Durgakund, Varanasi', '7773456789'),

('Tarun Sharma', 'tarun@sharma.com', 'Mansarovar, Jaipur', '8884567890'),

('Akhil Nanda', 'akhil@nanda.com', 'Koramangala, Bangalore', '9995678901'),

('Yuvraj Singh', 'yuvraj@singh.com', 'Sahibzada Ajit Singh Nagar, Punjab', '1116789012'),

('Ishita Menon', 'ishita@menon.com', 'Edapally, Kochi', '2227890123'),

('Vikrant Chauhan', 'vikrant@chauhan.com', 'Civil Lines, Delhi', '3338901234'),

('Arjun Reddy', 'arjun@reddy.com', 'HiTech City, Hyderabad', '4449012345'),

('Kishore Kumar', 'kishore@kumar.com', 'Dilsukhnagar, Hyderabad', '5550123456');

INSERT INTO Orders (CustomerID, ProductID, OrderDate, Quantity) VALUES

(23, 18, '2024-08-30', 1),

(9, 7, '2024-07-10', 1),

(8, 1, '2024-01-20', 1),

(16, 11, '2024-12-25', 1),

(4, 12, '2024-09-09', 1),

(23, 14, '2024-03-05', 2),

(7, 7, '2024-10-15', 1),

(14, 11, '2024-12-30', 1),

(6, 9, '2025-02-11', 1),

(22, 1, '2024-03-04', 2),

(10, 13, '2024-01-31', 1),

(5, 20, '2025-01-25', 2),

(15, 3, '2025-02-13', 1),

(12, 17, '2025-03-02', 1),

(27, 5, '2024-08-18', 3),

(33, 12, '2024-11-13', 2),

(25, 16, '2024-01-17', 1),

(14, 16, '2025-04-23', 1),

(22, 19, '2025-03-30', 1),

(7, 7, '2024-06-07', 2),

(18, 5, '2024-09-10', 2),

(3, 5, '2024-09-28', 6),

(3, 7, '2024-05-24', 1),

(6, 6, '2024-08-01', 6),

(18, 11, '2025-03-13', 1),

(12, 12, '2024-03-10', 1),

(30, 8, '2024-03-16', 1),

(17, 17, '2024-08-27', 1),

(25, 10, '2024-11-09', 1),

(19, 3, '2024-03-03', 1),

(4, 5, '2024-11-13', 6),

(2, 13, '2024-10-29', 2),

(8, 4, '2024-07-23', 1),

(23, 6, '2025-03-12', 1),

(26, 15, '2024-04-10', 6),

(10, 1, '2024-11-10', 1),

(12, 3, '2024-01-07', 1),

(26, 7, '2024-09-21', 1),

(24, 5, '2025-03-18', 1),

(26, 6, '2024-09-03', 1),

(2, 6, '2024-04-20', 1),

(18, 11, '2024-04-24', 1),

(35, 20, '2024-07-01', 1),

(27, 19, '2024-09-04', 2),

(19, 5, '2024-10-21', 1),

(31, 2, '2024-01-23', 1),

(35, 17, '2025-04-19', 1),

(31, 20, '2024-02-16', 2),

(28, 9, '2024-09-08', 1),

(9, 5, '2024-07-10', 1),

(9, 7, '2024-07-10', 4),

(9, 16, '2024-07-10', 3);

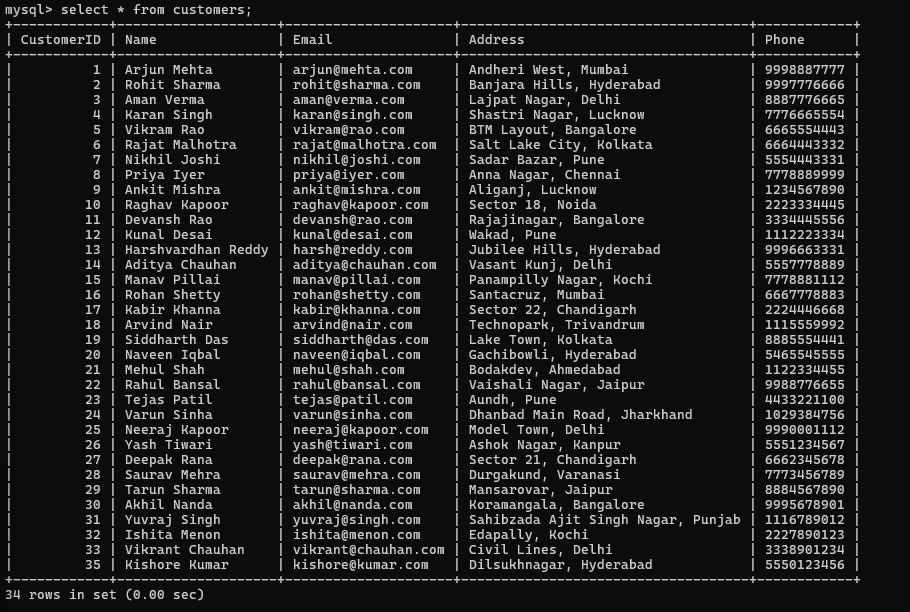
commit;

**Tables after data insertion**

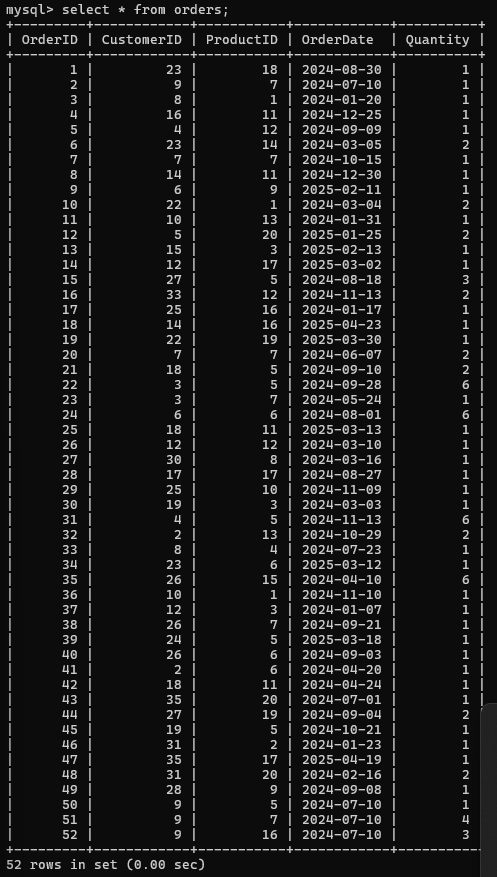
**Products:**



**Customers:**



**Orders:**



**Stored Procedures**

-- Stored procedure for query 1

DELIMITER //

CREATE PROCEDURE ListAvailableProducts()

BEGIN

    SELECT \* FROM Products WHERE Stock > 0;

END //

DELIMITER ;

-- Stored procedure for query 2

DELIMITER //

CREATE PROCEDURE GetCustomerPurchaseHistory(IN p\_CustomerID INT)

BEGIN

    SELECT o.OrderID, p.Name AS ProductName, o.OrderDate, o.Quantity, p.Price, (o.Quantity \* p.Price) AS TotalCost

    FROM Orders o

    JOIN Products p ON o.ProductID = p.ProductID

    WHERE o.CustomerID = p\_CustomerID

    ORDER BY o.OrderDate DESC;

END //

DELIMITER ;

-- Stored procedure for query 3

DELIMITER //

CREATE PROCEDURE GetMostPurchasedProduct()

BEGIN

    SELECT p.ProductID, p.Name, SUM(o.Quantity) AS TotalPurchased

    FROM Orders o

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY p.ProductID, p.Name

    ORDER BY TotalPurchased DESC

    LIMIT 1;

END //

DELIMITER ;

-- Stored procedure for query 5

DELIMITER //

CREATE PROCEDURE DeleteCustomer(IN p\_CustomerID INT)

BEGIN

    DELETE FROM Customers WHERE CustomerID = p\_CustomerID;

END //

DELIMITER ;

-- Stored procedure for query 6

DELIMITER //

CREATE PROCEDURE GenerateSalesReport()

BEGIN

    SELECT o.OrderID, c.Name AS CustomerName, p.Name AS ProductName, o.OrderDate, o.Quantity, p.Price, (o.Quantity \* p.Price) AS TotalCost

    FROM Orders o

    JOIN Customers c ON o.CustomerID = c.CustomerID

    JOIN Products p ON o.ProductID = p.ProductID

    ORDER BY o.OrderDate DESC;

END //

DELIMITER ;

-- Stored procedure for query 7

DELIMITER //

CREATE PROCEDURE GetRecentCustomers()

BEGIN

    SELECT DISTINCT c.CustomerID, c.Name, c.Email

    FROM Customers c

    JOIN Orders o ON c.CustomerID = o.CustomerID

    WHERE o.OrderDate >= DATE\_SUB(CURDATE(), INTERVAL 3 MONTH);

END //

DELIMITER ;

-- Stored procedure for query 8

DELIMITER //

CREATE PROCEDURE GetTotalRevenueByCategory()

BEGIN

    SELECT p.CategoryID, SUM(o.Quantity \* p.Price) AS TotalRevenue

    FROM Orders o

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY p.CategoryID;

END //

DELIMITER ;

-- Stored procedure for query 9

DELIMITER //

CREATE PROCEDURE GetTop5Customers()

BEGIN

    SELECT c.CustomerID, c.Name, SUM(o.Quantity \* p.Price) AS TotalSpent

    FROM Orders o

    JOIN Customers c ON o.CustomerID = c.CustomerID

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY c.CustomerID, c.Name

    ORDER BY TotalSpent DESC

    LIMIT 5;

END //

DELIMITER ;

-- Stored procedure for query 10

DELIMITER //

CREATE PROCEDURE GetOutOfStockProducts()

BEGIN

    SELECT \* FROM Products WHERE Stock = 0;

END //

DELIMITER ;

-- Stored procedure for query 11

DELIMITER //

CREATE PROCEDURE GetMostFrequentCategory()

BEGIN

    SELECT p.CategoryID, COUNT(o.OrderID) AS TotalOrders

    FROM Orders o

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY p.CategoryID

    ORDER BY TotalOrders DESC

    LIMIT 1;

END //

DELIMITER ;

-- Stored procedure for query 12

DELIMITER //

CREATE PROCEDURE GetOrdersByMonth(IN p\_Year INT, IN p\_Month INT)

BEGIN

    SELECT \* FROM Orders WHERE YEAR(OrderDate) = p\_Year AND MONTH(OrderDate) = p\_Month;

END //

DELIMITER ;

-- Stored procedure for query 13

DELIMITER //

CREATE PROCEDURE GetAverageOrderValue()

BEGIN

    SELECT c.CustomerID, c.Name, AVG(o.Quantity \* p.Price) AS AvgOrderValue

    FROM Orders o

    JOIN Customers c ON o.CustomerID = c.CustomerID

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY c.CustomerID, c.Name

    ORDER BY AvgOrderValue DESC;

END //

DELIMITER ;

-- Stored procedure for query 14

DELIMITER //

CREATE PROCEDURE GetLargeOrders()

BEGIN

    SELECT \* FROM Orders WHERE Quantity > 5;

END //

DELIMITER ;

-- Stored procedure for query 15

DELIMITER //

CREATE PROCEDURE GetCustomersWithoutOrders()

BEGIN

    SELECT \* FROM Customers WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Orders);

END //

DELIMITER ;

-- Stored procedure for query 16

DELIMITER //

CREATE PROCEDURE GetOrdersByTotalValue()

BEGIN

    SELECT o.OrderID, c.Name AS CustomerName, SUM(o.Quantity \* p.Price) AS TotalValue

    FROM Orders o

    JOIN Customers c ON o.CustomerID = c.CustomerID

    JOIN Products p ON o.ProductID = p.ProductID

    GROUP BY o.OrderID, c.Name

    ORDER BY TotalValue DESC;

END //

DELIMITER ;

**Triggers**

-- Trigger for query 4

DELIMITER //

CREATE TRIGGER UpdateStockAfterOrder

BEFORE INSERT ON Orders

FOR EACH ROW

BEGIN

    DECLARE current\_stock INT;

    -- Get current stock for the product

    SELECT Stock INTO current\_stock

    FROM Products

    WHERE ProductID = NEW.ProductID;

    -- Check if stock is sufficient

    IF current\_stock < NEW.Quantity THEN

        SIGNAL SQLSTATE '45000'

        SET MESSAGE\_TEXT = 'Not enough stock available for the product.';

    END IF;

    -- If enough stock, update it

    UPDATE Products

    SET Stock = Stock - NEW.Quantity

    WHERE ProductID = NEW.ProductID;

END;

//

DELIMITER ;

**Queries and Outputs**

1. **List all available products in stock.**

**Source code:**

**CALL ListAvailableProducts();**

**Output:**

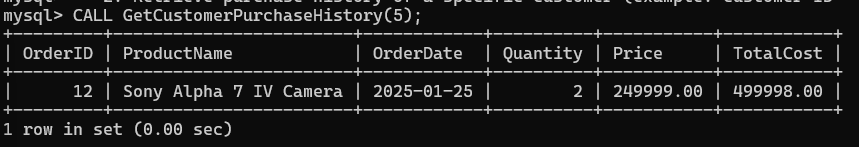


1. **Retrieve purchase history of a specific customer.**

**Source code:**

**CALL GetCustomerPurchaseHistory(5);**

**Output:**

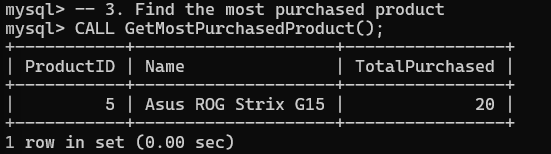


1. **Find the most purchased product.**

**Source code:**

**CALL GetMostPurchasedProduct();**

**Output:**



1. **Update stock quantity after a purchase.**

**Source code:**

-- Trigger for query 4

DELIMITER //

CREATE TRIGGER UpdateStockAfterOrder

BEFORE INSERT ON Orders

FOR EACH ROW

BEGIN

    DECLARE current\_stock INT;

    -- Get current stock for the product

    SELECT Stock INTO current\_stock

    FROM Products

    WHERE ProductID = NEW.ProductID;

    -- Check if stock is sufficient

    IF current\_stock < NEW.Quantity THEN

        SIGNAL SQLSTATE '45000'

        SET MESSAGE\_TEXT = 'Not enough stock available for the product.';

    END IF;

    -- If enough stock, update it

    UPDATE Products

    SET Stock = Stock - NEW.Quantity

    WHERE ProductID = NEW.ProductID;

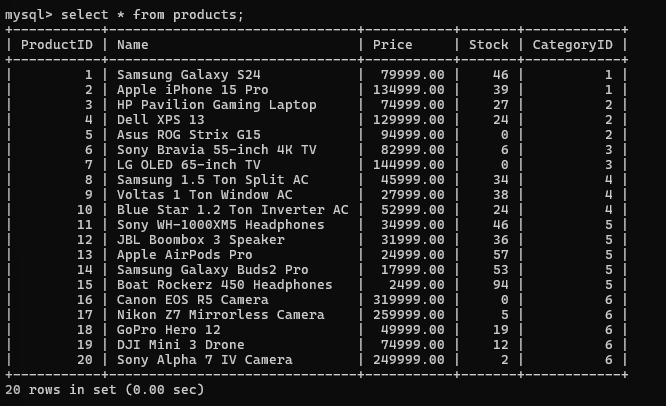
END;

//

DELIMITER ;

**Output:**

Final table after all orders ,changes are facilitated by the trigger.

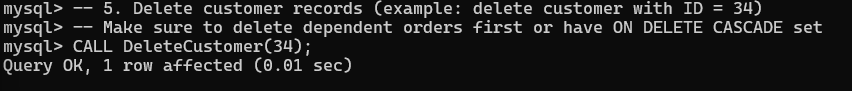


1. **Delete customer records.**

**Source code:**

CALL DeleteCustomer(34);

**Output:**



1. **Generate a sales report.**

**Source code:**

CALL GenerateSalesReport();

**Output:**

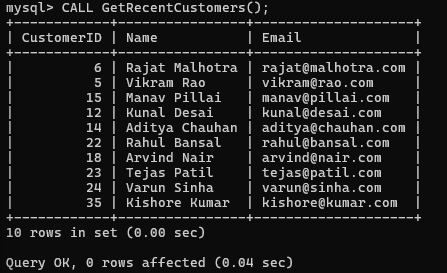


1. **Retrieve customers who placed orders in the last 3 months.**

**Source code:**

**CALL GetRecentCustomers();**

**Output:**

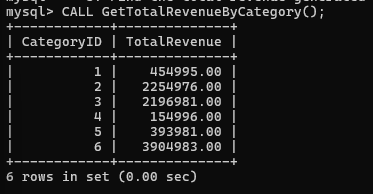


1. **Find the total revenue generated by each category.**

**Source code:**

**CALL GetRecentCustomers();**

**Output:**

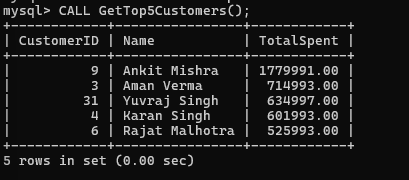


1. **Retrieve the top 5 customers with the highest total purchase value.**

**Source code:**

**CALL GetTop5Customers();**

**Output:**

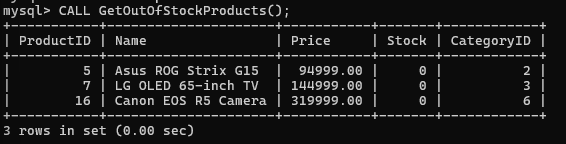


1. **List products that are out of stock.**

**Source code:**

**CALL GetOutOfStockProducts();**

**Output:**

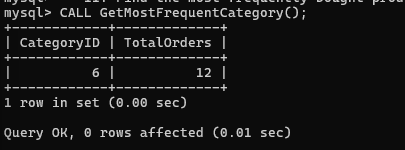


1. **Find the most frequently bought product category.**

**Source code:**

**CALL GetMostFrequentCategory();**

**Output:**

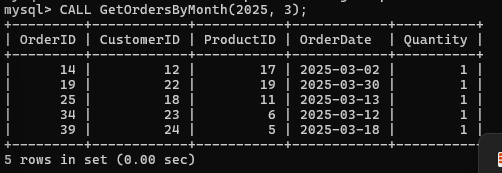


1. **Retrieve orders placed during a specific month.**

**Source code:**

**CALL GetOrdersByMonth(2025, 3);**

**Output:**

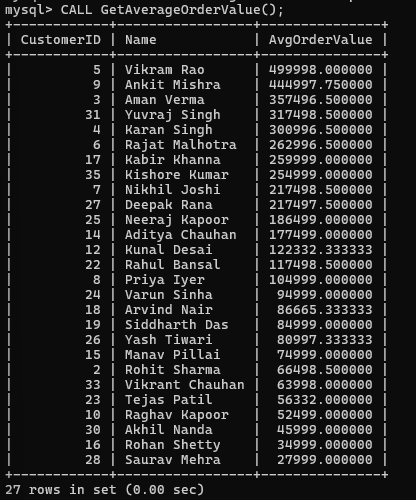


1. **Find the average order value per customer.**

**Source code:**

**CALL GetAverageOrderValue();**

**Output:**

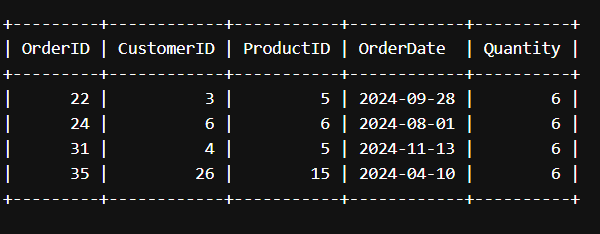


1. **Retrieve all orders with more than 5 items.**

**Source code:**

**CALL GetLargeOrders();**

**Output:**

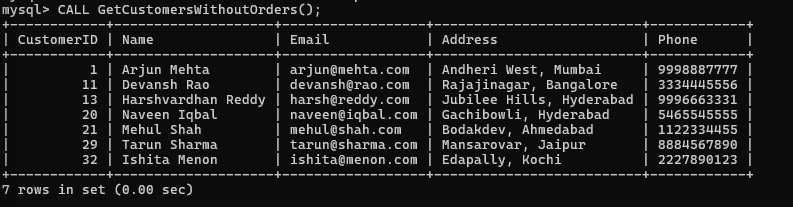
****

1. **Find customers who have never placed an order.**

**Source code:**

**CALL GetCustomersWithoutOrders();**

**Output:**



1. **Retrieve orders sorted by highest total value.**

**Source code:**

**CALL GetOrdersByTotalValue();**

**Output:**

