**CODING ASSESSMENT**

**BY AMITA C**

**Week1**

**Exercises (Morning Session)**

**--1. Calculate the Total Amount Spent by Each Customer**

SELECT c.CustomerName, SUM(o.TotalAmount) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerName;

**--Find Customers Who Have Spent More Than $1,000 in Total**

SELECT c.CustomerName, SUM(o.TotalAmount) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerName

HAVING SUM(o.TotalAmount) > 1000;

**--Find Product Categories with More Than 5 Products**

SELECT Category, COUNT(\*) AS ProductCount

FROM Products

GROUP BY Category

HAVING COUNT(\*) > 1;

**--Calculate the Total Number of Products for Each Category and Supplier Combination**

SELECT P.CategoryId, s. SupplierId, s. SupplierName,

COUNT (p.ProductId) AS TotalProducts

FROM Products p

JOIN Supplier s

ON p. Supplierlds. SupplierId

GROUP BY p. CategoryId, s. SupplierId, s. SupplierName;

**-- Summarize total sales by product and customer, and also provide an overall total**

SELECT product\_id, customer\_id, SUM(price) AS TotalSales FROM Orders

GROUP BY product\_id, customer\_id;

UNION ALL

SELECT NULL AS product\_id, NULL AS customer\_id, SUM(price) AS TotalSales

FROM Orders;

**Exercises ( Afternoon Session )**

**--Stored Procedure with Insert Operation**

**-- SQL Server / MySQL**

CREATE PROCEDURE InsertCustomer

@CustomerID INT,

@CustomerName VARCHAR(100),

@BirthDate DATE

AS

BEGIN

INSERT INTO Customers (CustomerID, CustomerName, BirthDate)

VALUES (@CustomerID, @CustomerName, @BirthDate);

END;

**--Stored Procedure with Update Operation**

**-- SQL Server / MySQL**

CREATE PROCEDURE UpdateCustomer

@CustomerID INT,

@CustomerName VARCHAR(100),

@BirthDate DATE

AS

BEGIN

UPDATE Customers

SET CustomerName = @CustomerName,

BirthDate = @BirthDate

WHERE CustomerID = @CustomerID;

END;

**--Stored Procedure with Delete Operation**

**-- SQL Server / MySQL**

CREATE PROCEDURE DeleteCustomer

@CustomerID INT

AS

BEGIN

DELETE FROM Customers

WHERE CustomerID = @CustomerID;

END;

**HANDS – ON EXCERSISES**

CREATE DATABASE CODING

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

BirthDate DATE

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10, 2),

StockQuantity INT

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

INSERT INTO Customers (CustomerID, CustomerName, BirthDate)

VALUES (1, 'John Doe', '1985-06-15'),

(2, 'Jane Smith', '1990-08-22'),

(3, 'Alice Johnson', '1975-12-05');

INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity)

VALUES (1, 'Smartphone', 'Electronics', 800.00, 50),

(2, 'Laptop', 'Electronics', 1200.00, 30),

(3, 'Headphones', 'Accessories', 150.00, 100),

(4, 'Tablet', 'Electronics', 600.00, 40),

(5, 'Desk', 'Furniture', 300.00, 20);

INSERT INTO Orders (OrderID, OrderDate, CustomerID, TotalAmount)

VALUES (1, '2024-08-01', 1, 1000.00),

(2, '2024-08-15', 2, 1200.00),

(3, '2024-08-20', 3, 600.00);

INSERT INTO OrderDetails (OrderDetailID, OrderID, ProductID, Quantity)

VALUES (1, 1, 1, 1),

(2, 1, 3, 2),

(3, 2, 2, 1),

(4, 3, 4, 1);

**--1. Retrieve all products from the Products table that belong to the category 'Electronics' and have a price greater than 500.**

SELECT \* FROM Products

WHERE Category = 'Electronics' AND Price > 500;

**--2. Calculate the total quantity of products sold from the Orders table**

SELECT SUM(Quantity) AS TotalQuantitySold

FROM OrderDetails;

**--3. Calculate the total revenue generated for each product in the Orders table**.

SELECT p.ProductName, SUM(od.Quantity \* p.Price) AS TotalRevenue

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

GROUP BY p.ProductName;

**--4. Write a query that uses WHERE, GROUP BY, HAVING, and ORDER BY clauses and explain the order of execution.**

SELECT p.Category, SUM(od.Quantity \* p.Price) AS TotalRevenue

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

WHERE p.Price > 500

GROUP BY p.Category

HAVING SUM(od.Quantity \* p.Price) > 1000

ORDER BY TotalRevenue DESC;

**--5. Write a query that corrects a violation of using non-aggregated columns without grouping them.**

SELECT Category, COUNT(\*) AS ProductCount

FROM Products

GROUP BY Category;

**--6. Retrieve all customers who have placed more than 5 orders using GROUP BY and HAVING clauses.**

SELECT c.CustomerName, COUNT(o.OrderID) AS OrderCount

FROM Orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

GROUP BY c.CustomerName

HAVING COUNT(o.OrderID) > 1;

**--STORED PROCEDURES**

**--Basic stored procedures : Create a stored procedure named GetAllCustomers that retrieves all customer details from the Customers table.**

CREATE PROCEDURE GetAllCustomers

AS

BEGIN

SELECT \* FROM Customers;

END;

**--2. Stored Procedure with Input Parameter**

**--Create a stored procedure named GetOrderDetailsByOrderID that accepts an OrderID as a parameter and retrieves the order details for that specific order.**

CREATE PROCEDURE GetOrderDetailsByOrderID

@OrderID INT

AS

BEGIN

SELECT od.OrderDetailID, p.ProductName, od.Quantity, p.Price

FROM OrderDetails od

JOIN Products p ON od.ProductID = p.ProductID

WHERE od.OrderID = @OrderID;

END;

**--3. Stored Procedure with Multiple Input Parameters**

**--Create a stored procedure named GetProductsByCategoryAndPrice that accepts a product Category and a minimum Price as input parameters and retrieves all products that meet the criteria.**

CREATE PROCEDURE GetProductsByCategoryAndPrice

@Category VARCHAR(50),

@MinPrice DECIMAL(10, 2)

AS

BEGIN

SELECT ProductID, ProductName, Price, StockQuantity

FROM Products

WHERE Category = @Category AND Price >= @MinPrice;

END;

**--4. Stored Procedure with Insert Operation**

**--Create a stored procedure named InsertNewProduct that accepts parameters for ProductName, Category, Price, and StockQuantity and inserts a new product into the Products table.**

CREATE PROCEDURE InsertNewProduct

@ProductName VARCHAR(100),

@Category VARCHAR(50),

@Price DECIMAL(10, 2),

@StockQuantity INT

AS

BEGIN

INSERT INTO Products (ProductName, Category, Price, StockQuantity)

VALUES (@ProductName, @Category, @Price, @StockQuantity);

END;

**--5. Stored Procedure with Update Operation**

**--Create a stored procedure named UpdateCustomerEmail that accepts a CustomerID and a NewEmail parameter and updates the email address for the specified customer.**

CREATE PROCEDURE UpdateCustomerEmail

@CustomerID INT,

@NewEmail VARCHAR(100)

AS

BEGIN

UPDATE Customers

SET Email = @NewEmail

WHERE CustomerID = @CustomerID;

END;

**--6. Stored Procedure with Delete Operation**

**--Create a stored procedure named DeleteOrderByID that accepts an OrderID as a parameter and deletes the corresponding order from the Orders table.**

CREATE PROCEDURE GetTotalProductsInCategory

@Category VARCHAR(50),

@TotalProducts INT OUTPUT

AS

BEGIN

SELECT @TotalProducts = COUNT(\*)

FROM Products

WHERE Category = @Category;

END;

**--7. Stored Procedure with Output Parameter**

**--Create a stored procedure named GetTotalProductsInCategory that accepts a Category parameter and returns the total**

CREATE PROCEDURE GetTotalProductsInCategory

@Category VARCHAR(50),

@TotalProducts INT OUTPUT

AS

BEGIN

SELECT @TotalProducts = COUNT(\*)

FROM Products

WHERE Category = @Category;

END;