EXERCISES (23.08.2024)

Select Customers, SUM(Price) AS TotalAmountSpent from Products

group by Customers;

Select Customers, SUM(Price) AS TotalAmount from Products

group by Customers

having SUM(Price)>1000;

Select Category,COUNT(ProductId) As NumofProducts from Products

Group By Category

Having count(ProductId)>5;

Select category, COUNT(ProductId) AS TotalProducts from Products

Group by category;

Select p.Customers ,p.ProductId, SUM(o.Quantity\*p.Price) AS TotalSales

From Products p

JOIN Orders o ON p.ProductId=o.ProductId

Group By p.Customers,p.ProductId

**Stored Procedure with Insert Operation**

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**

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**

CREATE PROCEDURE DeleteCustomer

@CustomerID INT

AS

BEGIN

DELETE FROM Customers

WHERE CustomerID = @CustomerID;

END;

**HANDS ON EXERCISES:**

**1**. **Filtering Data using SQL Queries**

**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. Total Aggregations using SQL Queries**

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

SELECT SUM(Quantity) AS TotalQuantity from Orders;

**3.Group By Aggregations using SQL Queries**

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

SELECT p.ProductName, p.ProductId, SUM(o.Quantity\*p.Price) AS TotalRevenue from Orders o

JOIN Products p ON o.ProductId=p.ProductId

Group by p.ProductId,p.ProductName;

4. **Order of Execution of SQL Queries**

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

SELECT

Category,

COUNT(ProductId) AS ProductCount,

AVG(Price) AS AveragePrice

FROM

Products

WHERE

Price > 100

GROUP BY

Category

HAVING

COUNT(ProductId) > 0

ORDER BY

AveragePrice;

**5. Rules and Restrictions to Group and Filter Data in SQL Queries**

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

SELECT

ProductId,

ProductName,

Category,

SUM(Price) AS TotalPrice

FROM

Products

GROUP BY

ProductId,

ProductName,

Category;

**6. Filter Data based on Aggregated Results using Group By and Having**

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

SELECT

CustomerId,

COUNT(OrderID) AS OrderCount

FROM

Orders

GROUP BY

CustomerId

HAVING

COUNT(OrderID) > 5;

**STORED PROCEDURES**

**1. Basic Stored Procedure**

**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 \* FROM Orders

WHERE OrderID = @OrderID;

END;

EXEC GetOrderDetailsByOrderID @OrderID = 1;

**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 \* 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(100),

@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(25)

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 DeleteOrderByID

@OrderID INT

AS

BEGIN

DELETE FROM Orders

WHERE OrderID = @OrderID;

END;

**7.Stored Procedure with Output Parameter**

**Create a stored procedure named GetTotalProductsInCategory that accepts a Category parameter and returns the total number of products in that category using an output parameter.**

CREATE PROCEDURE GetTotalProductsInCategory

@Category VARCHAR(100),

@TotalProducts INT OUTPUT

AS

BEGIN

SELECT @TotalProducts = COUNT(\*)

FROM Products

WHERE Category = @Category;

END;