-- Morning Exercises

-- Calculate the total amount spent by each customer

SELECT c.customer\_name, SUM(o.price) AS TotalSpent FROM Orders o JOIN Customers c ON

o.customer\_id = c.customer\_id GROUP BY c.customer\_name;

-- Find the customers who have spent more that 1000 in total

SELECT c.customer\_name, SUM(o.price) AS TotalSpent FROM Orders o JOIN Customers c ON

o.customer\_id = c.customer\_id GROUP BY c.customer\_name HAVING SUM(o.price) > 1000;

-- Find Product categories with more than 5 products

SELECT Category, COUNT(\*) AS ProductCount FROM Products

GROUP BY Category HAVING COUNT(\*) > 5;

-- Calculate the total number of products for each category and supplier combination

SELECT Category, Supplier, COUNT(\*) AS ProductCount FROM Products

GROUP BY Category, Supplier;

-- 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

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

FROM Orders;

-- Afternoon Stored Procedure Exercise

-- Stored Procedure with Insert

CREATE PROCEDURE InsertProductRecord

@product\_id INT,

@product\_name VARCHAR(50),

@price DECIMAL(10,2)

AS

BEGIN

INSERT INTO Products (product\_id,product\_name,price)

values (@product\_id,@product\_name,@price);

END;

exec InsertProductRecord @product\_id = 7, @product\_name = 'Tablet',

@price = 500.00;

-- Stored Procedure with Update

CREATE PROCEDURE UpdateProductRecord

@product\_id INT,

@product\_name VARCHAR(50),

@price DECIMAL(10,2)

AS

BEGIN

UPDATE Products

SET product\_name = @product\_name, price = @price

WHERE product\_id = @product\_id

END;

exec UpdateProductRecord @product\_id = 2, @product\_name = 'Tablet',

@price = 100.00;

-- Stored Procedure with Delete

CREATE PROCEDURE DeleteProductRecord

@product\_id INT

AS

BEGIN

DELETE FROM Products WHERE product\_id = @product\_id

END;

exec DeleteProductRecord @product\_id = 1

-- 1. Hands-on Exercise: 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 ProductName FROM Products WHERE Category = 'Electronics'

and Price > 500;

-- 2. Hands-on Exercise: Total Aggregations using SQL Queries

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

SELECT SUM(Quantity) AS TotalQuantity FROM Orders;

-- 3. Hands-on Exercise: Group By Aggregations using SQL Queries

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

SELECT p.ProductName, SUM(o.TotalAmount) AS TotalRevenue from Orders O

join Products p on o.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(o.TotalAmount) as TotalRevenue from Orders o

join Products p on o.ProductID = p.ProductID WHERE o.OrderDate > '2024-08-01'

GROUP BY p.Category HAVING SUM(o.TotalAmount) > 20000

ORDER BY TotalRevenue DESC;

-- Initially all the query execution starts from the 'FROM' which finds the tables

-- Then we will use 'WHERE' to specify the condition to filter

-- Once we have completed filtering we can start grouping by 'GROUP BY', we did group by the category

-- 'HAVING' should be applied after grouping to filter more with aggregation, here we are checking for total amount > 20000

-- 'ORDER BY' is given finally to sort the result in descending

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

SELECT p.ProductName, SUM(o.Quantity) AS TotalQuantity from Orders o

join Products p on o.ProductID = p.ProductID

GROUP BY p.ProductName

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

SELECT c.FirstName, COUNT(o.OrderID) as NoOfOrders from

Customers c join Orders o on c.CustomerID = o.CustomerID

GROUP BY c.FirstName HAVING COUNT(o.OrderID) > 5;

-- 1. Create a stored procedure named GetAllCustomers that retrieves all customer details from the Customers table.

CREATE PROCEDURE GetAllCustomers

AS

BEGIN

SELECT \* FROM Customers;

END;

exec GetAllCustomers

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

CREATE PROCEDURE GetOrderDetailsByID

@OrderID INT

AS

BEGIN

SELECT \* FROM Orders

WHERE OrderID = @OrderID

END;

exec GetOrderDetailsByID @OrderID = 1

-- 3. 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;

exec GetProductsByCategoryAndPrice @Category = 'Electronics' ,

@MinPrice = 500.00;

-- 4. 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(50),

@Category VARCHAR(50),

@Price DECIMAL(10,2),

@StockQuantity INT

AS

BEGIN

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

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

END;

EXEC InsertNewProduct @ProductName = 'Headphone', @Category = 'Electronics',

@Price = 500.00, @StockQuantity = 15;

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

select \* from Customers

CREATE PROCEDURE UpdateCustomerEmail

@CustomerID INT,

@NewEmail VARCHAR(50)

AS

BEGIN

UPDATE Customers

SET Email = @NewEmail

WHERE CustomerID = @CustomerID;

END;

EXEC UpdateCustomerEmail @CustomerID = 1, @NewEmail = 'abc@gmail.com';

--6. 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;

EXEC DeleteOrderByID @OrderID = 1

--7. 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(50),

@TotalProducts INT OUTPUT

AS

BEGIN

SELECT @TotalProducts = COUNT(\*) FROM Products

where Category = @Category;

END;

DECLARE @TOTAL INT;

EXEC GetTotalProductsInCategory @Category = 'Electronics', @TotalProducts = @Total OUTPUT

SELECT @Total AS TotalProductsInCategory