EXERCISE-1

1. Calculate total amount spent by each customer.

Query: SELECT CustomerID,CustomerName, SUM(TotalAmount) as TotalSpent
FROM Customers
GROUP BY CustomerID;

2. Find customers who spent more than 1000$ in total.

Query: SELECT CustomerID,CustomerName ,SUM(TotalAmount) as TotalSpent
FROM Customers
GROUP BY CustomerID
HAVING SUM(TotalAmount) > 1000;

Query: SELECT CustomerID,CustomerName ,SUM(TotalAmount) as TotalSpent
FROM Customers
WHERE TotalAmount > 1000;

3. Find product categories with more than 5 products.

Query: SELECT Category, COUNT(ProductID) as NumberOfProducts
FROM Products
GROUP BY Category
HAVING COUNT(ProductID) > 5**;**

4. Calculate the total no of products for each category and supplier combination.

Query: SELECT Category, SupplierID, COUNT(*) as NumberOfProducts
FROM Products
GROUP BY Category, SupplierID;

Query: SELECT Category,SupplierName,COUNT(ProductId) as NumberOfProducts
FROM Suppliers
JOIN Products p on s.SupplierID=p.SupplierID

GROUP BY SupplierID,ProductID

5. Summarize total sales by product and customer, also provide an overall cost.

Query: SELECT CustomerID, CustomerName, ProductID, SUM(amount) as OverallCost

FROM Customers c

JOIN Products p On p.ProductID=c.ProductID

GROUP BY CustomerID,ProductID

STORED PROCEDURE:

1. Stored Procedure with Insert Operation.

```
CREATE PROCEDURE InsertProduct
    @p_product_id INT,
    @p_product_name VARCHAR(50),
    @p_price INT
AS
BEGIN
    INSERT INTO Products (product_id, product_name, price)
    VALUES (@p_product_id, @p_product_name,@p_price);
END;
EXEC InsertProduct @p_product_id=2, @p_product_id='TV', @p_price=5000;
```

2. Stored Procedure with Delete Operation.

```
CREATE PROCEDURE DeleteProduct
    @p_product_id INT
AS
BEGIN
    DELETE FROM Products
    WHERE product_id =@p_product_id;
END ;
EXEC DeleteProduct  @p_product_id=1;
```

3. Stored Procedure with Update Operation.

```
CREATE PROCEDURE UpdateProduct
    @p_product_id INT,
    @p_product_name VARCHAR(50),
    @p_price INT
AS
BEGIN
    UPDATE Products
    SET product_name = @p_product_name, price = @p_price
    WHERE product_id =@ p_product_id;
END ;
EXEC UpdateProduct @p_product_id=1, @p_product_name='Laptop' , @p_price=7000;
```

CODING CHALLENGE:

HANDS ON EXERCISE:

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

Query:
```
SELECT * FROM Products
    WHERE Category = 'Electronics' AND Price > 500;
```

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

Query:
```
SELECT SUM(Quantity) AS TotalQuantitySold
    FROM Orders;
```

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

Query:
```
SELECT ProductID, SUM(Quantity*TotalAmount) AS TotalRevenue
    FROM Orders
    GROUP BY ProductID;
```

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

Query: SELECT ProductID, SUM(TotalAmount) AS TotalRevenue

```
FROM Orders
WHERE OrderDate >= '2024-08-01'
GROUP BY ProductID
HAVING SUM(TotalAmount) > 50000
ORDER BY TotalRevenue DESC;
```

Order of Execution:

1. **FROM**: The table is selected (Orders).
2. **WHERE**: Rows are filtered based on the WHERE condition (OrderDate >= '2024-08-01').
3. **GROUP BY**: The remaining rows are grouped by ProductID.
4. **HAVING**: Groups are filtered based on the HAVING condition (SUM(TotalAmount) > 50000).
5. **SELECT**: The selected columns (ProductID, SUM(TotalAmount) AS TotalRevenue) are retrieved.
6. **ORDER BY**: The final result set is sorted according to TotalRevenue in descending order.

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

Query: Example of Incorrect Query:

```
SELECT ProductID, ProductName, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY ProductID;
```

*ProductName is neither aggregated nor included in the GROUP BY clause, which violates SQL's rules for using aggregate functions.

Corrected Query:

```
SELECT ProductID, ProductName, SUM(Quantity) AS TotalQuantity
FROM Orders
GROUP BY ProductID, ProductName;
```

*This query groups by both ProductID and ProductName, ensuring that every column in the SELECT clause is either aggregated or part of the GROUP BY clause.

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

Query: SELECT CustomerID, COUNT(OrderID) AS OrderCount

FROM Orders

GROUP BY CustomerID

HAVING COUNT(OrderID) > 5;

STORED PROCEDURE:

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;

Exec GetAllCustomers;
```

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(20),
@MinPrice Int
AS
Begin
    Select * from products
    Where Category=@Category and Price=@MinPrice;
END;
Exec GetProductsByCategoryAndPrice @Category = 'Electronics', @MinPrice = 1000;
```

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
@ProductID int,
@ProductName varchar(20),
@Category varchar(20),
@Price int,
@Stock int
AS
Begin
    INSERT INTO Products (ProductID, ProductName, Category, Price, Stock)
    Values (@productID,@ProductName, @Category, @Price, @Stock)
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 varchar(20),

@NewEMail varchar(20)

AS

Begin

        Update Customers

        Set Email=@NewEMail

        Where CustomerID=@CustomerID;

END;

Exec UpdateCustomerEmail  @CustomerID = 2, @NewEMail = 'amit.sharma@example.com';
```

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;

Exec DeleteOrderByID @OrderID = 1;
```

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(20),
@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;
```