EXERCISE

1. Calculate the total amount spent by each customer.

```
select CustomerID ,SUM(Amount) AS TotalAmount
from Orders
GROUP BY CustomerID;
```

2. Find customers who have spent more than $1,000 in total.

```
select CustomerID ,SUM(Amount) AS TotalAmount
from Orders
GROUP BY CustomerID
HAVING SUM(Amount) > 1000;
```

3. Find Product Categories with More Than 5 Products

```
select CategoryID,COUNT(Productid) AS ProductCount
from Products
GROUP BY CategoryID
HAVING COUNT(ProductID) >5;
```

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

```
select CategoryID, SupplierID, COUNT(ProductID) AS ProductCount
from Products
GROUP BY CategoryID, SupplierID;
```

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

```
SELECT CustomerID, ProductID, SUM(Amount) AS TotalSales
FROM Orders
GROUP BY ROLLUP(CustomerID, ProductID);
```

6.Stored Procedure with Insert Operation

```
create procedure InsertProduct
@ProductID int ,
@ProductName varchar(100) ,
@Category varchar(50) ,
@Price DECIMAL(10,2) ,
@StockQuantity int
AS
```

```
BEGIN
insert into Products (ProductID, ProductName, Category, Price, StockQuantity)
values (@ProductID, @ProductName, @Category, @Price, @StockQuantity);
END;

EXEC InsertProduct
@ProductID=105,
@ProductName = 'Mouse',
@Category = 'Electronics',
@Price = 99.99,
@StockQuantity = 5;
```

7. Stored Procedure with Update Operation

```
create procedure UpdateProduct
@ProductID INT,
@Price DECIMAL(10, 2) = NULL
AS
BEGIN
UPDATE Products
SET Price = ISNULL(@Price, Price)
WHERE ProductID = @ProductID;
END;

exec UpdateProduct
@ProductID = 1,
@Price = 89.99;
```

8. Stored Procedure with Delete Operation

```
create procedure DeleteProduct
@ProductID INT
AS
BEGIN
DELETE FROM Products
```

```
WHERE ProductID = @ProductID;
END;

exec DeleteProduct
@ProductID = 3;
```

Hands-on

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 ProductID 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 ProductID , SUM(Quantity * Amount) AS TotalRevenue
from Orders
group by ProductID;
```

4. Hands-on Exercise: 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 ProductID, SUM(Quantity * Amount) AS TotalRevenue
FROM Orders
WHERE OrderDate >= '2024-08-20'
GROUP BY ProductID
HAVING SUM(Quantity * Amount) > 100
```

ORDER BY TotalRevenue DESC;

Order of Execution:

FROM: The query identifies the data source (the Orders table).

WHERE: filters the rows in table on the condition (OrderDate >= '2024-01-01')

GROUP BY: groups the remaining rows by ProductID.

HAVING: filters the groups and only keep groups where the SUM(Quantity * Price) is greater than 1000.

SELECT: selects the columns specified from the groups that passed the HAVING filter.

ORDER BY: orders the resulting rows by TotalRevenue in descending order.

5. Hands-on Exercise: 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.

--Incorrect Query :

SELECT ProductID, SUM(Quantity) AS TotalQuantity

FROM Orders;

--Correctd Query :

SELECT ProductID, SUM(Quantity) AS TotalQuantity

FROM Orders

GROUP BY ProductID;

6. Hands-on Exercise: 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(*) AS OrderCount

FROM Orders

GROUP BY CustomerID

HAVING COUNT(*) > 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 Customer;

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 = 1001;
```

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 GetProductsByCategoryAndPrice2

@Category VARCHAR(50),
```

```
@MinPrice DECIMAL(10, 2)
AS
BEGIN
SELECT *
FROM Products
WHERE Category = @Category AND Price >= @MinPrice;
END;

EXEC GetProductsByCategoryAndPrice2
@Category = 'Electronics',
@MinPrice = 500;
```

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(100),
@Category VARCHAR(50),
@Price DECIMAL(10, 2),
@StockQuantity INT
AS
BEGIN
INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity)
VALUES (@ProductID, @ProductName, @Category, @Price, @StockQuantity);
END;

EXEC InsertNewProduct
@ProductID = 106,
@ProductName = 'Monitor',
```

@Category = 'Electronics',

@Price = 99.99,

@StockQuantity = 10;

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(255)

AS

BEGIN

UPDATE Customer

SET Email = @NewEmail

WHERE CustomerID = @CustomerID;

END;

EXEC UpdateCustomerEmail

@CustomerID = 1,

@NewEmail = 'varsh@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 GetTotalProductsInCategory2
@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 TotalProductsInCategory2;
```