**1. Calculate the total amount spent by each customer.**

select o.CustomerID, SUM(o.TotalAmount) as TotalSpent

from Orders as o

group by o.CustomerID

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

select o.CustomerID

from Orders as o

group by o.CustomerID

having SUM(o.TotalAmount) >= 1000

**3. Find Product Categories with More Than 5 Products**

select Category from Products

group by Category

having count(ProductID)>5

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

select Category,supplier,count(\*)

from Products

group by Category,supplier

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

select ProductID, CustomerID, SUM(TotalAmount) as TotalSales

from Orders as o join Customers as c on o.OrderID=c.OrderID

group by ProductID, CustomerID

**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 \* 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 ProductID, 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(TotalAmount) 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(TotalAmount) AS TotalRevenue from Orders

where OrderDate >= '2024-08-01' group by ProductID

having SUM(TotalAmount) > 500 order by TotalRevenue DESC;

**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.**

select AVG(price) as AveragePrice,SUM(StockQuantity) as TotalStockQuantity from Products group by Category

**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(OrderID) from orders

group by CustomerId

having COUNT(OrderID)>5

**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(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. 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(50),@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=7, @productName='Watch' , @category='Electronics', @price=2000.85 ,@stockQuantity=100

**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 UpdateCustomerEmail1

@customerID int,@email varchar(50)

as

begin

update Customers set Email=@email

where CustomerID=@customerID

end

exec UpdateCustomerEmail1 @customerID=1,@email='johnsnow@gmail.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=2

**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