1. Calculate the total amount spent by each customer.

 Select customerID,CustomerName,sum(amount) as totalAmount from group by customerID

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

Select customerId,customerName

from customers

group by customerId

having sum(amount)>10000

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 SupplierId,SupplierName,count(ProductId) from supplier s

join products p on s.supplierID=p.supplierID

group by SupplierID,productid

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

 Select customerID,customerName,productid,sum(amount)

from customers c join products p On p.productID=c.ProductID

group by CustomerID,c.productID

Stored Procedures:

* Executed only once and get fetched from cache
* Can combine more than one queries
* Used for data abstraction as the people don’t know what are the queries present inside a stored procedure

1. Stored Procedure with Insert Operation

create procedure c2

@Id int,

@pname varchar(20),

@category varchar(20),

@price int,

@stock int

AS

begin

insert into products values(@Id,@pname,@category,@price,@stock);

end;

2. Stored Procedure with Update Operation

create procedure c3

@exi int,

@ans int

AS

begin

update products set Price=@ans where Price=@exi;

end;

3. Stored Procedure with Delete Operation

Create procedure DeleteOrderByID

@OrderID

AS

Begin

Delete from orders where orderID=@OrderID

END;

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=’Electronic’ 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 Total\_Quantity 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 revenue from orders group by productid having count(productID)>1

This query gives the total revenue generated by a product which is ordered more than one time

The order of execution is:

a. The sql engine will group the products based on their ID’s

b. The sql engine will check for product ordered more than once from the groups made

c. now the sql engine will display the product Id and total revenue generated

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 sum(price),avg(price) from products;

Thie query uses aggregate functions without any grouping

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 \* from customers from orders o

join customers c on c.customerId=o.customerId

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 Order where OrderID=@OrderID;

END;

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),

@Price Int

AS

Begin

Select \* from products where category=@category and price=@price;

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

@productID int,

@ProductName varchar(20),

@Category varchar(20),

@Price int,

@Stock int

AS

Begin

Insert into products values(@productID int,,@ProductName, @Category, @Price, @Stock)

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;

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

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(20),

@Total INT OUTPUT

AS

BEGIN

Select @Total =count(\*) from products where Category= @category;

END;