Queries

Subquery with JOIN and GROUP BY

```
-- Find the average order total for each customer who has made more than 5 orders
SELECT c.CustomerID, c.FirstName, c.LastName, AVG(o.TotalDue) AS AverageOrderTotal
FROM SalesLT.Customer c
JOIN SalesLT.SalesOrderHeader o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName
HAVING COUNT(o.SalesOrderID) > 5
ORDER BY AverageOrderTotal DESC;
```

Subquery in WHERE Clause with Aggregate Function

```
-- Find customers who have placed orders with a total quantity greater than the average quantity
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer
WHERE CustomerID IN (
   SELECT CustomerID
   FROM SalesLT.SalesOrderHeader
   GROUP BY CustomerID
   HAVING SUM(OrderQty) > (SELECT AVG(OrderQty) FROM SalesLT.SalesOrderHeader)
);
```

Correlated Subquery with ORDER BY

```
-- Retrieve products with a list price greater than the average list price of their category
SELECT ProductID, Name, ListPrice, ProductCategoryID
FROM SalesLT.Product p
WHERE ListPrice > (
   SELECT AVG(ListPrice)
   FROM SalesLT.Product
```

```
    WHERE ProductCategoryID = p.ProductCategoryID
)
ORDER BY ProductCategoryID, ListPrice DESC;
```

JOIN with Aggregation and HAVING

```
-- Find employees who have processed more than 100 orders and the total quantity is greater than 5000
SELECT e.EmployeeID, e.FirstName, e.LastName, COUNT(o.SalesOrderID) AS OrderCount, SUM(od.OrderQty) AS TotalQuantity
FROM SalesLT.Employee e
JOIN SalesLT.SalesOrderHeader o ON e.EmployeeID = o.SalesPersonID
JOIN SalesLT.SalesOrderDetail od ON o.SalesOrderID = od.SalesOrderID
GROUP BY e.EmployeeID, e.FirstName, e.LastName
HAVING COUNT(o.SalesOrderID) > 100 AND SUM(od.OrderQty) > 5000
ORDER BY TotalQuantity DESC;
```

These examples showcase the use of subqueries and joins with various keywords to perform complex analyses and filtering in a relational database.

User

now write some queries to create triggers, procedures and views

ChatGPT

Certainly! Here are examples of SQL queries to create triggers, procedures, and views:

1. Create Trigger:

sql

Copy code

```
-- Create a trigger to update a log table when a new product is inserted
CREATE TRIGGER trg_ProductInsert
ON SalesLT.Product
AFTER INSERT
```

```
AS
BEGIN
    INSERT INTO LogTable (LogMessage, LogDate)
    VALUES ('New product inserted.', GETDATE())
END;
```

2. Create Stored Procedure:

sql

Copy code

```
-- Create a stored procedure to retrieve customer details based on a specific city
CREATE PROCEDURE usp_GetCustomersByCity
    @City NVARCHAR(50)
AS
BEGIN
    SELECT CustomerID, FirstName, LastName, EmailAddress
    FROM SalesLT.Customer
    WHERE City = @City;
END;
```

3. Create View:

```
-- Create a view to display product details along with category information
CREATE VIEW vw_ProductDetailsWithCategory AS
SELECT p.ProductID, p.Name AS ProductName, p.ListPrice, pc.Name AS CategoryName
FROM SalesLT.Product p
JOIN SalesLT.ProductCategory pc ON p.ProductCategoryID = pc.ProductCategoryID;
```

4. Create Trigger with INSTEAD OF Insert Trigger:

sql

Copy code

```
-- Create an INSTEAD OF INSERT trigger to modify the behavior of inserting into a view
CREATE TRIGGER trg_InsteadOfInsert
ON vw_ProductDetailsWithCategory
INSTEAD OF INSERT
AS
BEGIN
    -- Perform custom logic before the actual insert
    -- For example, validate data, transform it, or insert into underlying tables

    -- Insert into the underlying tables
    INSERT INTO SalesLT.Product (ProductID, Name, ListPrice)
    SELECT ProductID, ProductName, ListPrice
    FROM inserted;
END;
```

5. Create Stored Procedure with OUTPUT Parameter:

sql

Copy code

```
-- Create a stored procedure with an OUTPUT parameter to get the total number of customers
CREATE PROCEDURE usp_GetTotalCustomers
    @TotalCustomers INT OUTPUT
AS
BEGIN
    SELECT @TotalCustomers = COUNT(CustomerID)
    FROM SalesLT.Customer;
```

```
END;
```

These examples provide you with the syntax to create triggers, stored procedures, and views in a SQL database. Modify the queries based on your specific database structure and requirements.

User

create triggers for insert,delete,update

ChatGPT

Certainly! Here are examples of triggers for INSERT, DELETE, and UPDATE operations:

1. Trigger for INSERT:

```
-- Create a trigger to log inserted records into a history table
CREATE TRIGGER trg_InsertHistory
ON YourTable -- Replace with your actual table name
AFTER INSERT
AS
BEGIN
   INSERT INTO HistoryTable (ActionType, RecordID, Timestamp)
   SELECT 'INSERT', IDColumn, GETDATE()
   FROM inserted;
END;
```

2. Trigger for DELETE:

```
-- Create a trigger to log deleted records into a history table
CREATE TRIGGER trg_DeleteHistory
ON YourTable -- Replace with your actual table name
AFTER DELETE
AS
BEGIN
   INSERT INTO HistoryTable (ActionType, RecordID, Timestamp)
   SELECT 'DELETE', IDColumn, GETDATE()
   FROM deleted;
```

```
END;
```

3. Trigger for UPDATE:

```
-- Create a trigger to log updated records into a history table
CREATE TRIGGER trg_UpdateHistory
ON YourTable -- Replace with your actual table name
AFTER UPDATE
AS
BEGIN
    INSERT INTO HistoryTable (ActionType, RecordID, Timestamp)
    SELECT 'UPDATE', IDColumn, GETDATE()
    FROM inserted; -- Use 'inserted' to get the updated values
END;
```

Make sure to replace YourTable, HistoryTable, and IDColumn with your actual table name, history table name, and primary key column name, respectively. Also, adjust the logic within the trigger based on your specific requirements.

Note: These triggers are examples, and you should adapt them to your database schema and business rules. Additionally, keep in mind that triggers can impact performance, so use them judiciously.

User

write some queries for procedures with 3 to 4 variables for aggregate functions

ChatGPT

Certainly! Here are examples of stored procedures with 3 to 4 variables that use aggregate functions:

1. Procedure with 3 Variables and Aggregate Function:

```
-- Create a procedure to calculate average sales for a specific product category
CREATE PROCEDURE GetAverageSalesByCategory
    @CategoryID INT,
    @StartDate DATE,
    @EndDate DATE
```

```
AS
BEGIN
  SELECT ProductCategoryID,
      AVG(SalesAmount) AS AverageSales
  FROM Sales
  WHERE ProductCategoryID = @CategoryID
   AND OrderDate BETWEEN @StartDate AND @EndDate
  GROUP BY ProductCategoryID;
END;
```

2. Procedure with 4 Variables and Aggregate Function:

```
-- Create a procedure to calculate total sales for a specific product and customer
CREATE PROCEDURE GetTotalSalesByProductAndCustomer
  @ProductID INT,
  @CustomerID INT,
  @StartDate DATE,
  @EndDate DATE
AS
BEGIN
  SELECT ProductID,
      CustomerID,
      SUM(SalesAmount) AS TotalSales
  FROM Sales
  WHERE ProductID = @ProductID
   AND CustomerID = @CustomerID
   AND OrderDate BETWEEN @StartDate AND @EndDate
  GROUP BY ProductID, CustomerID;
END;
```

In these examples, the procedures take parameters such as @CategoryID, @StartDate, @EndDate, @ProductID, @CustomerID, and use aggregate functions (AVG and SUM) in the SELECT statement along with grouping (GROUP BY). Adjust the table and column names based on your actual database schema.

Remember to customize these examples to fit your specific use case and database structure.

2 / 2

User

now write queries for transactions and save points

ChatGPT

Certainly! Below are examples of queries demonstrating the use of transactions and savepoints:

1. Transaction Example:

```
-- Start a transaction
BEGIN TRANSACTION;

-- Update the quantity of a product
UPDATE Products
SET Quantity = Quantity - 10
WHERE ProductID = 123;

-- Insert a new order
INSERT INTO Orders (OrderID, ProductID, Quantity, OrderDate)
VALUES (456, 123, 10, GETDATE());

-- If an error occurs, roll back the entire transaction
IF @@ERROR <> 0
```

```
    ROLLBACK;
ELSE
    -- If no error, commit the transaction
    COMMIT;
```

This example demonstrates a simple transaction that updates the quantity of a product and inserts a new order. If any error occurs during the transaction, it is rolled back; otherwise, it is committed.

2. Savepoint Example:

```
-- Start a transaction
BEGIN TRANSACTION;

-- Savepoint 1
SAVE TRANSACTION Savepoint1;

-- Update the quantity of a product
UPDATE Products
SET Quantity = Quantity - 10
WHERE ProductID = 123;

-- Savepoint 2
SAVE TRANSACTION Savepoint2;

-- Insert a new order
INSERT INTO Orders (OrderID, ProductID, Quantity, OrderDate)
VALUES (456, 123, 10, GETDATE());

-- If an error occurs, roll back to Savepoint 2
IF @@ERROR <> 0
    ROLLBACK TO Savepoint2;
```

```
ELSE

    -- If no error, commit the transaction

    COMMIT;
```

In this example, savepoints (Savepoint1 and Savepoint2) are used within a transaction. If an error occurs, the transaction can be rolled back to a specific savepoint, allowing you to undo part of the transaction while preserving the changes made before the savepoint.

Customize these examples based on your specific database schema and requirements.

User

write some more for delete insert update

ChatGPT

Certainly! Below are examples of triggers for INSERT, UPDATE, and DELETE operations:

1. Trigger for INSERT Operation:

sql

Copy code

```
CREATE TRIGGER trgAfterInsert

ON SalesLT.Product

AFTER INSERT

AS

BEGIN

    -- Your logic after an insert (e.g., logging, updating other tables)

    INSERT INTO AuditLog (TableName, Action, DateTime)

    VALUES ('Product', 'INSERT', GETDATE());

END;
```

This trigger (trgAfterInsert) is fired after an INSERT operation on the SalesLT.Product table. It logs the insert action into an AuditLog table.

2. Trigger for UPDATE Operation:

sql

Copy code

```
CREATE TRIGGER trgAfterUpdate
ON SalesLT.Product
AFTER UPDATE
AS
BEGIN
   -- Your logic after an update (e.g., logging, sending notifications)
   INSERT INTO AuditLog (TableName, Action, DateTime)
   VALUES ('Product', 'UPDATE', GETDATE());
END;
```

This trigger (trgAfterUpdate) is fired after an UPDATE operation on the SalesLT.Product table. It logs the update action into an AuditLog table.

3. Trigger for DELETE Operation:

sql

Copy code

```
CREATE TRIGGER trgBeforeDelete
ON SalesLT.Product
BEFORE DELETE
AS
BEGIN
   -- Your logic before a delete (e.g., checking dependencies, archiving data)
   INSERT INTO AuditLog (TableName, Action, DateTime)
   VALUES ('Product', 'DELETE', GETDATE());
END;
```

This trigger (trgBeforeDelete) is fired before a DELETE operation on the SalesLT.Product table. It logs the delete action into an AuditLog table. Note that BEFORE DELETE triggers allow you to perform actions before the actual delete occurs.

These triggers are basic examples, and you should customize them based on your specific requirements and database schema. Also, consider implementing proper error handling and additional logic as needed.

1. Subquery with IN Operator:

sql

Copy code

```
-- Find customers who have placed orders in a specific region
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer
WHERE CustomerID IN (
    SELECT CustomerID
    FROM SalesLT.SalesOrderHeader
    WHERE ShipRegion = 'North'
);
```

2. Subquery with NOT IN Operator:

sql

Copy code

```
-- Find customers who have not placed any orders
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer
WHERE CustomerID NOT IN (
    SELECT CustomerID
    FROM SalesLT.SalesOrderHeader
);
```

3. Subquery with EXISTS Operator:

sql

Copy code

```sql
-- Find customers who have placed orders
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer C
WHERE EXISTS (
    SELECT 1
    FROM SalesLT.SalesOrderHeader SO
    WHERE SO.CustomerID = C.CustomerID
);
```

4. Subquery with NOT EXISTS Operator:

sql

Copy code

```sql
-- Find customers who have not placed any orders
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer C
WHERE NOT EXISTS (
    SELECT 1
    FROM SalesLT.SalesOrderHeader SO
    WHERE SO.CustomerID = C.CustomerID
);
```

5. Subquery with Aggregation and HAVING Clause:

sql

Copy code

```sql
-- Find customers who have placed more than 3 orders
SELECT CustomerID, FirstName, LastName
FROM SalesLT.Customer C
WHERE CustomerID IN (
    SELECT CustomerID
    FROM SalesLT.SalesOrderHeader
    GROUP BY CustomerID
    HAVING COUNT(*) > 3
);
```

6. Correlated Subquery with OUTER APPLY:

sql

Copy code

```
-- Find customers along with their latest order details
SELECT C.CustomerID, C.FirstName, C.LastName, LatestOrder.OrderID, LatestOrder.OrderDate
FROM SalesLT.Customer C
OUTER APPLY (
    SELECT TOP 1 OrderID, OrderDate
    FROM SalesLT.SalesOrderHeader SO
    WHERE SO.CustomerID = C.CustomerID
    ORDER BY OrderDate DESC
) AS LatestOrder;
```

1. INNER JOIN:

```
-- Retrieve customers along with their order details
SELECT C.CustomerID, C.FirstName, C.LastName, O.OrderID, O.OrderDate
FROM SalesLT.Customer C
INNER JOIN SalesLT.SalesOrderHeader O ON C.CustomerID = O.CustomerID;
```

2. LEFT JOIN (or LEFT OUTER JOIN):

```
-- Retrieve all products along with their reviews (if available)
SELECT P.ProductID, P.Name, R.Review
FROM SalesLT.Product P
LEFT JOIN SalesLT.ProductReview R ON P.ProductID = R.ProductID;
```

3. RIGHT JOIN (or RIGHT OUTER JOIN):

```
-- Retrieve all employees along with their assigned territories
SELECT E.EmployeeID, E.FirstName, E.LastName, T.TerritoryName
FROM HumanResources.Employee E
RIGHT JOIN SalesLT.EmployeeTerritory ET ON E.EmployeeID = ET.EmployeeID
JOIN SalesLT.SalesTerritory T ON ET.TerritoryID = T.TerritoryID;
```

4. FULL JOIN (or FULL OUTER JOIN):

```
-- Retrieve all customers and their orders, including unmatched records
SELECT C.CustomerID, C.FirstName, C.LastName, O.OrderID, O.OrderDate
FROM SalesLT.Customer C
FULL JOIN SalesLT.SalesOrderHeader O ON C.CustomerID = O.CustomerID;
```

5. CROSS JOIN:

sql

Copy code

```
-- Retrieve all combinations of products and categories
SELECT P.ProductID, P.Name AS ProductName, C.Name AS CategoryName
FROM SalesLT.Product P
CROSS JOIN SalesLT.ProductCategory C;
```

6. SELF JOIN:

```
-- Find employees who share the same manager
SELECT E1.EmployeeID, E1.FirstName, E1.LastName, E2.EmployeeID AS ManagerID, E2.FirstName AS ManagerFirstName, E2.LastName AS ManagerLastName
FROM HumanResources.Employee E1
JOIN HumanResources.Employee E2 ON E1.ManagerID = E2.EmployeeID;
```

7. JOIN with WHERE Clause:

```
-- Retrieve orders and their corresponding ship methods for a specific customer
SELECT O.OrderID, O.OrderDate, O.ShipMethod
FROM SalesLT.SalesOrderHeader O
```

```
JOIN SalesLT.Customer C ON O.CustomerID = C.CustomerID
WHERE C.FirstName = 'John' AND C.LastName = 'Doe';
```

These queries showcase various types of joins and how they can be used to retrieve specific information from related tables in a database. Adjust the column names and table aliases based on your database schema.

User

use multiple joins in one query

ChatGPT

Certainly! Here's an example of a query that involves multiple joins:

sql

Copy code

```
-- Retrieve customer information along with their orders, order details, and product details
SELECT
    C.CustomerID,
    C.FirstName,
    C.LastName,
    O.OrderID,
    O.OrderDate,
    OD.OrderQty,
    P.Name AS ProductName,
    P.ListPrice
FROM
    SalesLT.Customer C
JOIN
    SalesLT.SalesOrderHeader O ON C.CustomerID = O.CustomerID
JOIN
    SalesLT.SalesOrderDetail OD ON O.OrderID = OD.OrderID
```

```
JOIN
    SalesLT.Product P ON OD.ProductID = P.ProductID;
```

Keys:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_CustomerOrder
FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID);

ALTER TABLE Customers
ADD CONSTRAINT PK_Customers
PRIMARY KEY (CustomerID);

ALTER TABLE Customers
ADD Email VARCHAR(100);
```

UPDATE Query:

sql

Copy code

```
-- Update the LastName for a specific customer
UPDATE Customers
SET LastName = 'NewLastName'
WHERE CustomerID = 1;
```

DELETE Query:

sql

Copy code

```
-- Delete a specific order from the Orders table
```

```sql
DELETE FROM Orders
WHERE OrderID = 100;
```

UPDATE Query with WHERE Clause:

sql

Copy code

```sql
-- Update the LastName for a specific customer with WHERE clause
UPDATE Customers
SET LastName = 'NewLastName'
WHERE CustomerID = 1;
```

DELETE Query with WHERE Clause:

Delete

```sql
-- Delete orders for a specific customer with WHERE clause
DELETE FROM Orders
WHERE CustomerID = 1;
```

SELECT Query with WHERE Clause:

Select

```sql
-- Select customers from a specific city with WHERE clause
SELECT * FROM Customers
WHERE City = 'New York';
```