Trigger

Trigger in MSsql basic to Advance

6 / 6

A trigger in MS SQL Server is a special type of stored procedure that is automatically executed in response to certain events, such as a data modification or a login attempt. Triggers can be used to enforce business rules, perform auditing, or execute complex tasks that involve multiple tables.

Here are some basic to advanced concepts of triggers in MS SQL Server:

1. ***Basic Syntax***: A trigger consists of three basic parts: the trigger event (INSERT, UPDATE, or DELETE), the trigger action (what happens when the trigger is fired), and the trigger scope (which tables are affected by the trigger).
2. ***Trigger Types***: There are two types of triggers in MS SQL Server: DML (Data Manipulation Language) triggers and DDL (Data Definition Language) triggers. DML triggers are fired in response to data changes, while DDL triggers are fired in response to changes in the database structure.
3. ***Trigger Timing:*** Triggers can be executed either before or after the triggering event. Before triggers are used to validate data or perform other checks before the data modification is allowed, while after triggers are used to perform additional processing after the data modification has already occurred.
4. ***Trigger Nesting:*** Triggers can be nested, meaning that one trigger can call another trigger, which in turn can call another trigger. However, care must be taken when using nested triggers to avoid infinite loops or excessive processing.
5. ***Trigger Performance:*** Triggers can have a significant impact on database performance, especially if they are complex or perform operations on large data sets. As a best practice, triggers should be kept as simple as possible and tested thoroughly before being deployed in a production environment.
6. ***Trigger Security:*** Triggers can also be used for security purposes, such as auditing user actions or enforcing data access rules. However, care must be taken to ensure that triggers are not used to bypass standard security measures or introduce security vulnerabilities.

In summary, triggers in MS SQL Server are a powerful tool for automating tasks and enforcing business rules. However, they must be used carefully and thoughtfully to avoid performance issues, security risks, and other problems.

1. ***Basic Syntax***

--=========================== Create =========================

CREATE TRIGGER trigger\_name

ON table\_name -- tableName

AFTER INSERT, UPDATE, DELETE -- trigger event(s)

AS

BEGIN

-- trigger action(s)

END

--=========================== Drop =========================

drop trigger triggerName

**Example \_1: Store Activities**

---\_\_\_ Table to Store Activites \_\_\_\_\_\_\_\_

CREATE TABLE AuditLog (

LogID int IDENTITY(1,1) PRIMARY KEY,

TableName varchar(50) NOT NULL,

Operation varchar(10) NOT NULL,

DateTime datetime NOT NULL DEFAULT GETDATE()

);

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Create Trigger \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TRIGGER AuditCustomers

ON Customers

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

INSERT INTO AuditLog (TableName, Operation, DateTime)

VALUES (

'Customers',

CASE

WHEN EXISTS(SELECT \* FROM inserted) AND EXISTS(SELECT \* FROM deleted) THEN 'UPDATE'

WHEN EXISTS(SELECT \* FROM inserted) THEN 'INSERT'

ELSE 'DELETE'

END,

GETDATE()

)

END

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ insert Customer \_\_\_\_\_\_\_\_\_\_\_\_\_\_

insert into customers values ('Rashid','Ali','saqib')

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ delete Customer \_\_\_\_\_\_\_\_\_\_\_\_\_\_

delete customers where id = 4

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Update Customer \_\_\_\_\_\_\_\_\_\_\_\_\_\_

update customers set f\_name = 'bilala' where id = 5

--\_\_\_\*\*\*\*\*\*\*\*\*\*\*\*\* See Last Activities \*\*\*\*\*\*\*\*\*\*\*\*

SELECT \* FROM AuditLog WHERE TableName = 'Customers';

**Example \_2: Validate Date**

1. Trigger with Validation: The following trigger prevents any rows from being inserted into the "Orders" table if the "OrderDate" is in the future:

-- create the Orders table

CREATE TABLE Ordersss (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerName VARCHAR(50)

)

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Create Trigger To validate Order date not Feauter date\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TRIGGER PreventFutureOrders

ON Ordersss

AFTER INSERT

AS

BEGIN

IF EXISTS(SELECT \* FROM inserted WHERE OrderDate > GETDATE())

BEGIN

RAISERROR ('Cannot insert orders with future dates.', 16, 1)

ROLLBACK TRANSACTION

END

END

-- insert 10 records into the Orders table

INSERT INTO Ordersss (OrderID, OrderDate, CustomerName)

VALUES

(1, '2022-01-01', 'John Doe'),

(2, '2022-02-15', 'Jane Smith'),

(3, '2022-03-31', 'Bob Johnson'),

(4, '2022-04-05', 'Mary Williams'),

(5, '2022-05-12', 'Tom Brown'),

(6, '2022-06-30', 'Samantha Lee'),

(7, '2022-07-04', 'George Wilson'),

(8, '2022-08-15', 'Linda Davis'),

(9, '2022-09-22', 'Richard Kim'),

(10, '2022-10-31', 'Susan Chen');

--\_\_\_\*\*\*\*\*\*\*\*\*\*\*\*\* See Trigger Activities \*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO Ordersss (OrderID, OrderDate, CustomerName)

VALUES

(11, '2023-03-25', 'John Doe'),

**Example \_3: Validate Date**

1. Trigger with Complex Logic: The following trigger updates a "SalesTotals" table with the total sales for each customer whenever a new order is inserted into the "Orders" table:

-- create the Customers table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(50),

Address VARCHAR(100),

City VARCHAR(50),

Country VARCHAR(50)

)

-- create the Orders table

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),

OrderAmount DECIMAL(10, 2)

)

-- create the SalesTotals table

CREATE TABLE SalesTotals (

CustomerID INT PRIMARY KEY,

TotalSales DECIMAL(10, 2)

)

-- insert some data into the Customers table

INSERT INTO Customers (CustomerID, CustomerName, Address, City, Country)

VALUES

(1, 'John Doe', '123 Main St', 'New York', 'USA'),

(2, 'Jane Smith', '456 Elm St', 'Los Angeles', 'USA'),

(3, 'Bob Johnson', '789 Oak St', 'Chicago', 'USA'),

(4, 'Mary Williams', '1010 Pine St', 'Houston', 'USA'),

(5, 'Tom Brown', '1111 Maple St', 'Miami', 'USA');

-- insert some data into the Orders table

INSERT INTO Orders (OrderID, OrderDate, CustomerID, OrderAmount)

VALUES

(1, '2022-01-01', 1, 100.00),

(2, '2022-02-15', 2, 200.00),

(3, '2022-03-31', 1, 150.00),

(4, '2022-04-05', 3, 75.00),

(5, '2022-05-12', 2, 300.00);

-- insert some data into the SalesTotals table

INSERT INTO SalesTotals (CustomerID, TotalSales)

VALUES

(1, 0.00),

(2, 0.00),

(3, 0.00),

(4, 0.00),

(5, 0.00);

--\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Trigger when Insert into Order \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

-- this will Insert record in SaleTotale Table

CREATE TRIGGER UpdateSalesTotals

ON Orders

AFTER INSERT

AS

BEGIN

UPDATE SalesTotals

SET TotalSales = TotalSales + (SELECT SUM(OrderAmount) FROM inserted WHERE CustomerID = SalesTotals.CustomerID)

WHERE CustomerID IN (SELECT CustomerID FROM inserted)

END

--\_\_\_\_\_\_\_ Trigger Event by Insert Record \_\_\_\_\_\_\_\_\_\_\_\_\_

INSERT INTO Orders (OrderID, OrderDate, CustomerID, OrderAmount)

VALUES

(15, '2022-01-01', 2, 1100.00)

--\_\_\_\_\_\_ Check Data \_\_\_\_\_\_\_

select \* from SalesTotals