Triggers in SQL Server

Introduction

Triggers are special kind of stored procedure that are executed / triggered as a certain operation on a table like inserting, updating or even deleting of data inside a table

It is also considered as database object which is fired automatically and must be related to a table

Also, it is not possible to manually invoke triggers ... To achieve this approach, they must perform the action needed on a specific table

---

Example of what is done after writing triggers:

Let's say that you write a trigger for updating some data inside a table

After firing the trigger

step 1: The trigger creates a table called 'UPDATED' in the memory

step 2: The update operations are executed

step 3: The statements which were written inside the triggers are executed

step 4: We can use the 'updated' table to query the data to perform any operations on its rows

---

Why Triggers?

What if we have a table called products and each product has a price value and a discount value

We want to know, how many times that this product's price is changed? or even how many times that the discount value is changed by inc. or dec.?

In the scenario, we will need to write a trigger that inserts the changed data into another table whenever the product price or the discount value is changed.

Types of Triggers

There are 3 types of triggers in SQL Server

1. **DML – Data Manipulation Language Triggers**

   These triggers allow some code to be executed when operations like insertion, updating or deleting are performed

2. **DDL – Data Definition Language Triggers**

   These triggers allow user to execute code as a response of changes in the structure of the Database (DROP, CREATE) or a Server Event (user login)

   Based on their scope, they are divided into 2 types

   - **Database Scoped DDL Triggers**
   - **Server Scoped DDL Triggers**

3- Logon Triggers

They are specifically Server Scoped DDL Triggers which is executed when LOGON event happened when the session of the user begins

Let's have a detailed talk for each type

1- DML Triggers

Fired as a response for DML events and not performed manually

They can be associated with only a table or view also with multiple DML events

Here is a syntax for DML Trigger

```
CREATE TRIGGER trigger_name
ON { Table name or view name }
{ FOR | AFTER | INSTEAD OF }
{ [INSERT], [UPDATE] , [DELETE] }
```

DML Triggers are classified into 2 types

1- AFTER Triggers

Fired after the action is completed successfully

Syntax:

```
CREATE TRIGGER schema_name.trigger_name
ON table_name
AFTER {INSERT | UPDATE | DELETE}
AS
    BEGIN
      -- Trigger Statements
                  -- Insert, Update, Or Delete Statements
                              END
```

Also, it is classifying into

a. AFTER INSERT Trigger

Fired after insert statement is completed successfully

Syntax:

```
-- Create the Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50)
);

-- Create the EmployeeAudit table to track the audit
trail
CREATE TABLE EmployeeAudit (
    AuditID INT PRIMARY KEY,
    Action VARCHAR(50),
    EmployeeID INT,
    InsertedDateTime DATETIME
);

-- Create the AFTER INSERT trigger
CREATE TRIGGER tr_AfterInsertEmployee
ON Employees
AFTER INSERT
AS
BEGIN
    -- Insert the audit record into the EmployeeAudit
table
    INSERT INTO EmployeeAudit (Action, EmployeeID,
InsertedDateTime)
    SELECT 'INSERT', EmployeeID, GETDATE()
    FROM inserted;
END;
```

The previous code shows that we have 2 tables 'Employees' and 'EmployeeAudit'

The AFTER INSERT trigger, 'tr_AfterInsertEmployee', will automatically fire when a new record is inserted in the table 'Employees'. The trigger will insert an audit record into the table 'EmployeeAudit'. It will record the 'INSERT' action that occurred, the 'EmployeeID' of the new employee added, and the current date and time.

NOTE:

"inserted' is the pseudo table that contains rows affected by 'INSERT' operations. It is a temporary table that mirrors the structure of the table owned by the trigger.

b. AFTER UPDATE Trigger

The AFTER UPDATE trigger allows to perform additional actions or execute specific logic immediately after an UPDATE operation has successfully modified the data.

Syntax :

```
-- Create the Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50)
);

-- Create the EmployeeAudit table to track the
audit trail
CREATE TABLE EmployeeAudit (
    AuditID INT PRIMARY KEY,
    Action VARCHAR(50),
    EmployeeID INT,
    InsertedDateTime DATETIME
);

-- Create the AFTER INSERT trigger
CREATE TRIGGER tr_AfterInsertEmployee
ON Employees
AFTER INSERT
AS
BEGIN
    -- Insert the audit record into the
EmployeeAudit table
    INSERT INTO EmployeeAudit (Action,
EmployeeID, InsertedDateTime)
    SELECT 'INSERT', EmployeeID, GETDATE()
    FROM inserted;
```

```
END;
```

In the above code, we created two tables, 'Products' and 'ProductPriceAudit'. Now when the 'Price' column of a product is updated in the 'Products' table, the trigger, 'tr_AfterUpdateProductPrice' is fired. It checks if the product price was updated using the 'IF UPDATE' condition. If the price was updated, then a record is inserted into the table, 'ProductPriceAudit'. The record will include 'ProductID', the old price, the new price, and the current date and time when the update occurred.

c. AFTER DELETE Trigger

The AFTER DELETE trigger allows the user to perform additional actions or execute specific logic immediately after a DELETE operation has successfully removed the data.

2- INSTEAD OF Triggers

The INSTEAD OF trigger fires before the triggered operation is executed. It fires even if the constraint check fails. It is further divided into three types as the AFTER Triggers

Syntax:

```
CREATE TRIGGER schema_name.trigger_name

ON table_name
INSTEAD OF {INSERT | UPDATE | DELETE}
      AS
            BEGIN
             -- trigger statements
```

```
    -- Insert, Update, or Delete commands
END
```

a. INSTEAD OF INSERT Trigger

b. INSTEAD OF UPDATE Trigger

c. INSTEAD OF DELETE Trigger

As the After triggers, all INSTEAD OF Triggers have the same way of work and even the same syntax

But there is one difference, They work before the operation of (inserting, updating or deleting) is performed

Example of INSTEAD OF INSERT Trigger

```
-- Create the Employees table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Salary DECIMAL(10, 2)
);

-- Create the INSTEAD OF INSERT trigger
CREATE TRIGGER tr_InsteadOfInsertEmployee
ON Employees
INSTEAD OF INSERT
AS
BEGIN
    -- Perform custom logic before the actual insert
    INSERT INTO Employees (EmployeeID, FirstName, LastName,
Salary)
    SELECT EmployeeID, FirstName, LastName,
           CASE
               WHEN Salary < 0 THEN 0  -- Ensure that
Salary cannot be negative
               ELSE Salary
           END
    FROM inserted;
           END;
```

In the above code, a table 'Employees' is created. When we try to insert a new record, the trigger 'tr_InsteadOfInsertEmployee' will be fired instead of the default insert operation. The trigger will check the 'Salary' column. If there is a negative value, it will convert it to zero. If the 'Salary' value is positive, then the record will be inserted with the original 'Salary' value. It will ensure there are no negative salary values inserted into the table.

2 – DDL Triggers

The DDL triggers are fired by a DDL event, such as CREATE, ALTER, DROP, and UPDATE statements. It can also be fired in response to certain system-defined stored procedures.

They are used when we have to prevent, audit, or respond to a change in the database schema.

To create a DML trigger we use the CREATE TRIGGER statement. Here is the syntax for the same.

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH [ENCRYPTION | EXECUTE AS Clause] ]
{ FOR | AFTER } { DDL event }
AS
      { Your code goes here }
```

3- LOGON Triggers

They are types of triggers that are executed after a successful logon event, which means that they are not fired on unsuccessful logon attempts

They are usually used to control server sessions such as tracking login sessions and limiting the number of sessions for certain logins

Syntax:

```
-- Create the audit log table to store successful logon events
CREATE TABLE LogonAudit (
    LogonID INT PRIMARY KEY IDENTITY(1,1),
    LoginName NVARCHAR(128),
    LogonTime DATETIME,
    ClientIP NVARCHAR(50)
);

-- Create the logon trigger
CREATE TRIGGER tr_LogonAudit
ON ALL SERVER
FOR LOGON
AS
BEGIN
    INSERT INTO LogonAudit (LoginName, LogonTime, ClientIP)
    VALUES (ORIGINAL_LOGIN(), GETDATE(), HOST_NAME());
     END;
```

The above code will create a table 'LogonAudit' to store the audit log for successful logon events. It has columns, 'LoginID', 'LoginName' to store the name of the user who has logged in, 'LogonTime' to record the timestamp of the logon event, and 'ClientIP' to store the IP address of the client.

The trigger 'tr_LogonAudit' will fire for all logon events on the SQL Server. Whenever a user will successfully log in to the SQL Server, the trigger will capture the login name of the user with 'ORIGINAL_LOGIN()', the current

timestamp using 'GETDATE()', and the client's IP address using 'HOSTNAME()'. It will be further inserted in the table 'LogonAudit'