Creating and managing Triggers in SQL Server

**What are triggers:**   
  
Triggers are a special type of stored procedure which are executed automatically based on the occurrence of a database event.

These events can be categorized as

1. Data Manipulation Language (DML) and
2. Data Definition Language (DDL) events.

A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updation of data. It is a database object which is bound to a table and is executed automatically. You can’t explicitly invoke triggers. The only way to do this is by performing the required action no the table that they are assigned to.v

The benefits derived from triggers is based in their events driven nature. Once created, the trigger automatically fires without user intervention based on an event in the database.

Types Of Triggers

There are three action query types that you use in SQL which are INSERT, UPDATE and DELETE. So, there are three types of triggers and hybrids that come from mixing and matching the events and timings that fire them. Basically, triggers are classified into two main types:

1. After Triggers (For Triggers)
2. Instead Of Triggers

(i) After Triggers

These triggers run after an insert, update or delete on a table. They are **not supported for views.**   
AFTER TRIGGERS can be classified further into three types as:

1. AFTER INSERT Trigger.
2. AFTER UPDATE Trigger.
3. AFTER DELETE Trigger.

**A) Using DML Triggers:**   
  
DML triggers are invoked when any DML commands like INSERT, DELETE, and UPDATE happen on the data of a table and or view.  
  
**Points to remember:**

1. DML triggers are powerful objects for maintaining database integrity and consistency.
2. DML triggers evaluate data before it has been committed to the database.
3. During this evaluation following actions are performed.
   * Compare before and after versions of data
   * Roll back invalid modification
   * Read from other tables ,those in other database
   * Modify other tables, including those in other database.
   * Execute local and remote stored procedures.

1. We cannot use following commands in DML trigger
   * ALTER DATABASE
   * CREATE DATABASE
   * DISK DATABASE
   * LOAD DATABASE
   * RESTORE DATABASE

1. Using the sys.triggers catalog view is a good way to list all the triggers in a database. To use it, we simply open a new query editor window in SSMS and select all the rows from the view as shown below;  
   select \* from sys.triggers

So let us create DML trigger.  
  
You can create and manage triggers in SQL Server Management Studio or directly via Transact-SQL (T-SQL) statements.  
  
**1) Using AFTER triggers:**

* An AFTER trigger is the original mechanism that SQL Server created to provide an automated response to data modifications
* AFTER triggers fire after the data modification statement completes but before the statement's work is committed to the databases.
* The trigger has the capability to roll back its actions as well as the actions of the modification statement that invoked it.

CREATE TRIGGER tr\_au\_upd ON authors  
AFTER UPDATE,INSERT,DELETE  
AS   
PRINT 'TRIGGER OUTPUT' +  CONVERT(VARCHAR(5),@@ROWCOUNT)  
+ 'ROW UPDATED'  
GO

UPDATE Statement  
  
UPDATE authors  
SET au\_fname = au\_fname  
WHERE state ='UT'  
  
Result:  
----------------------------------------------------  
TRIGGER OUTPUT2ROW UPDATED  
  
(2 row(s) affected)  
  
**Point to remember:**  
1) If we have a constraint and trigger defined on the same column, any violations to the constraint abort the statement and the trigger execution does not occur. For example, if we have a foreign key constraint on a table that ensures referential integrity and a trigger that that does some validation on that same foreign key column then the trigger validation will only execute if the foreign key validation is successful.  
  
**Can we create more than one trigger on one table?**

* We can create more than one trigger on a table for each data modification action. In other words, we can have multiple triggers responding to an INSERT, an UPDATE, or a DELETE command.
* The sp\_settriggerorder procedure is the tool we use to set the trigger order. This procedure takes the trigger name, order value (FIRST, LAST, or NONE), and action (INSERT, UPDATE, or DELETE) as parameters.  
  sp\_settriggerorder tr\_au\_upd, FIRST, 'UPDATE'
* AFTER triggers can only be placed on tables, not on views.
* A single AFTER trigger cannot be placed on more than one table.
* The text, ntext, and image columns cannot be referenced in the AFTER trigger logic.

**How to see inserted and deleted rows through Trigger:**

* We can find rows modified in the inserted and deleted temporary tables.
* For AFTER trigger, these temporary memories â€“resident tables contains the rows modified by the statement.
* With the INSTEAD OF trigger, the inserted and deleted tables are actually temporary tables created on-the-fly.

Lets us try and see how this works;  
  
a) Create a table titles\_copy  
  
SELECT \*  
INTO titles\_copy  
FROM titles  
GO  
  
b) Create a trigger on this table  
  
CREATE TRIGGER tc\_tr ON titles\_copy  
FOR INSERT , DELETE ,UPDATE  
AS  
PRINT 'Inserted'  
SELECT title\_id, type, price FROM inserted -- THIS IS TEMPORARY TABLE  
PRINT 'Deleted'  
SELECT title\_id, type, price FROM deleted -- THIS IS TEMPORARY TABLE  
--ROLLBACK TRANSACTION  
  
c) Let us UPDATE rows. After which trigger will get fired.  
  
  
We have written two statements in trigger, so these rows get printed. The inserted and deleted tables are available within the trigger after INSERT, UPDATE, and DELETE.  
  
PRINT 'Inserted'  
SELECT title\_id, type, price FROM inserted -- THIS IS TEMPORARY TABLE  
PRINT 'Deleted'  
SELECT title\_id, type, price FROM deleted -- THIS IS TEMPORARY TABLE  
  
Result is based on below rule.  
  
**Statement   Contents of inserted      Contents of deleted**  
-----------------------------------------------------------------  
INSERT         Rows added                     Empty  
UPDATE        New rows                        Old rows  
DELETE         Empty                             Rows deleted  
  
**2) INSTEAD OF Trigger:**

(ii) Instead Of Triggers

These can be used as an interceptor for anything that anyone tried to do on our table or view. If you define an *Instead Of trigger* on a table for the Delete operation, they try to delete rows, and they will not actually get deleted (unless you issue another delete instruction from within the trigger)

INSTEAD OF TRIGGERS can be classified further into three types as:

1. INSTEAD OF INSERT Trigger.
2. INSTEAD OF UPDATE Trigger.
3. INSTEAD OF DELETE Trigger.
4. Provides an alternative to the AFTER trigger that was heavily utilized in prior versions of SQL Server.
5. It performs its actions instead of the action that fired it.
6. This is much different from the AFTER trigger, which performs its actions after the statement that caused it to fire has completed. This means you can have an INSTEAD OF update trigger on a table that successfully completes but does not include the actual update to the table.
7. INSTEAD OF Triggers fire instead of the operation that fires the trigger, so if you define an INSTEAD OF trigger on a table for the Delete operation, they try to delete rows, they will not actually get deleted (unless you issue another delete instruction from within the trigger) as in below example:

Let us create INSTEAD OF trigger.  
  
if exists (select \* from sysobjects   
where id = object\_id('dbo.cust\_upd\_orders')  
and sysstat & 0xf = 8)  
drop trigger dbo.cust\_upd\_orders  
go

CREATE TRIGGER trI\_au\_upd ON authors  
INSTEAD OF UPDATE  
AS   
PRINT 'TRIGGER OUTPUT: '  
+CONVERT(VARCHAR(5), @@ROWCOUNT) + ' rows were updated.'  
GO  
  
Let us write an UPDATE statement now;  
  
UPDATE authors  
SET au\_fname = 'Rachael'  
WHERE state = 'UT'  
  
-----------------------------------------------------  
TRIGGER OUTPUT: 2 rows were updated.  
  
(2 row(s) affected)  
  
Let us see what has been updatded  
  
SELECT au\_fname, au\_lname FROM authors  
WHERE state = 'UT'  
  
au\_fname au\_lname  
----------------------  
Anne Ringer  
Albert Ringer  
  
  
Lets see another example;  
  
Create a Table  
  
CREATE TABLE nayan (Name  varchar(32))  
GO  
  
Create trigger with INSTEAD.  
  
CREATE TRIGGER tr\_nayan ON nayan   
INSTEAD OF DELETE  
AS  
    PRINT 'Sorry - you cannot delete this data'  
GO  
  
INSERT into nayan table  
  
INSERT nayan  
    SELECT 'Cannot' union  
    SELECT 'Delete' union  
    SELECT 'Me'  
GO  
  
Run the SQL DELETE statement.  
  
DELETE nayan  
GO  
  
-------------------------------  
Sorry - you cannot delete this data  
  
(3 row(s) affected)  
  
Run SELECT statement  
  
SELECT \* FROM nayan  
GO  
  
Result is below;  
  
Name  
-----------------  
Cannot  
Delete  
Me  
  
**Points to remember:**

1. As you can see from the results of the SELECT statement, the first name (au\_fname) column is not updated to 'Rachael'. The UPDATE statement is correct, but the INSTEAD OF trigger logic does not apply the update from the statement as part of its INSTEAD OF action. The  
   only action the trigger carries out is to print its message.
2. The important point to realize is that after you define an INSTEAD OF trigger on a table, you need to include all the logic in the trigger to perform the actual modification as well as any other actions that the trigger might need to carry out.
3. Triggering action-The INSTEAD OF trigger fires instead of the triggering action. As shown earlier, the actions of the INSTEAD OF trigger replace the actions of the original data modification that fired the trigger.
4. Constraint processing-Constraint processing-including CHECK constraints, UNIQUE constraints, and PRIMARY KEY constraints-happens after the INSTEAD OF trigger fires.
5. If you were to print out the contents of the inserted and deleted tables from inside an Instead Of trigger, you would see they behave in exactly the same way as normal. In this case, the deleted table holds the rows you were trying to delete, even though they will not get deleted.

**Benefits of INSTEAD Triggers:**

* We can define an INSTEAD OF trigger on a view (something that will not work with AFTER triggers) and this is the basis of the Distributed Partitioned Views that are used so split data across a cluster of SQL Servers.

**B) Using DDL Triggers:**

1. These triggers focus on changes to the definition of database objects as opposed to changes to the actual data.
2. This type of trigger is useful for controlling development and production database environments.

Let us create DDL trigger now;  
  
Below is the syntax.  
  
CREATE TRIGGER trigger\_name  
ON { ALL SERVER | DATABASE }  
[ WITH <ddl\_trigger\_option> [ ,...n ] ]  
{ FOR | AFTER } { event\_type | event\_group } [ ,...n ]  
AS { sql\_statement [ ; ] [ ...n ] | EXTERNAL NAME < method specifier > [ ; ] }  
  
CREATE TRIGGER tr\_TableAudit  
ON DATABASE  
FOR CREATE\_TABLE,ALTER\_TABLE,DROP\_TABLE  
AS  
      PRINT 'You must disable the TableAudit trigger in order  
              to change any table in this database'  
    ROLLBACK  
GO  
  
Other way of writing the same query in more optimized way is below;  
  
IF EXISTS(SELECT \* FROM sys.triggers   
WHERE name = N'tr\_TableAudit' AND parent\_class=0)  
DROP TRIGGER [tr\_TableAudit] ON DATABASE  
GO  
CREATE TRIGGER tr\_TableAudit ON DATABASE  
FOR DDL\_TABLE\_EVENTS  
AS  
      PRINT 'You must disable the TableAudit trigger in  
               order to change any table in this database'             
    ROLLBACK  
GO

All the triggers can be enabled/disabled on the table using the statement

ALTER TABLE Employee\_Test {ENABLE|DISBALE} TRIGGER ALL

Specific Triggers can be enabled or disabled as:

ALTER TABLE Employee\_Test DISABLE TRIGGER trgAfterDelete

This disables the After Delete Trigger named trgAfterDelete on the specified table.