## **Introduction**

Triggers in SQL Server

## **Background**

This article gives a brief introduction about Triggers in SQL Server 2000/2005.

## **What is a Trigger**

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.

## **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.

Let’s create After triggers. First of all, let’s create a table and insert some sample data. Then, on this table, I will be attaching several triggers.

Hide Copy Code

CREATE TABLE Employee\_Test  
(  
Emp\_ID INT Identity,  
Emp\_name Varchar(100),  
Emp\_Sal Decimal (10,2)  
)  
  
INSERT INTO Employee\_Test VALUES ('Anees',1000);  
INSERT INTO Employee\_Test VALUES ('Rick',1200);  
INSERT INTO Employee\_Test VALUES ('John',1100);  
INSERT INTO Employee\_Test VALUES ('Stephen',1300);  
INSERT INTO Employee\_Test VALUES ('Maria',1400);

I will be creating an AFTER INSERT TRIGGER which will insert the rows inserted into the table into another audit table. The main purpose of this audit table is to record the changes in the main table. This can be thought of as a generic audit trigger.

Now, create the audit table as:-

Hide Copy Code

CREATE TABLE Employee\_Test\_Audit  
(  
Emp\_ID int,  
Emp\_name varchar(100),  
Emp\_Sal decimal (10,2),  
Audit\_Action varchar(100),  
Audit\_Timestamp datetime  
)

## **(a) After Insert Trigger**

This trigger is fired after an INSERT on the table. Let’s create the trigger as:

Hide Copy Code

CREATE TRIGGER trgAfterInsert ON [dbo].[Employee\_Test]   
FOR INSERT  
AS  
 declare @empid int;  
 declare @empname varchar(100);  
 declare @empsal decimal(10,2);  
 declare @audit\_action varchar(100);  
  
 select @empid=i.Emp\_ID from inserted i;   
 select @empname=i.Emp\_Name from inserted i;   
 select @empsal=i.Emp\_Sal from inserted i;   
 set @audit\_action='Inserted Record -- After Insert Trigger.';  
  
 insert into Employee\_Test\_Audit  
 (Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)   
 values(@empid,@empname,@empsal,@audit\_action,getdate());  
  
 PRINT 'AFTER INSERT trigger fired.'  
GO

The CREATE TRIGGER statement is used to create the trigger. THE ON clause specifies the table name on which the trigger is to be attached. The FOR INSERT specifies that this is an AFTER INSERT trigger. In place of FOR INSERT, AFTER INSERT can be used. Both of them mean the same.

In the trigger body, table named **inserted** has been used. This table is a logical table and contains the row that has been inserted. I have selected the fields from the logical inserted table from the row that has been inserted into different variables, and finally inserted those values into the Audit table.

To see the newly created trigger in action, lets insert a row into the main table as:

Hide Copy Code

insert into Employee\_Test values('Chris',1500);

Now, a record has been inserted into the Employee\_Test table. The AFTER INSERT trigger attached to this table has inserted the record into the Employee\_Test\_Audit as:

Hide Copy Code

6 Chris 1500.00 Inserted Record -- After Insert Trigger. 2008-04-26 12:00:55.700

## **(b) AFTER UPDATE Trigger**

This trigger is fired after an update on the table. Let’s create the trigger as:

Hide Copy Code

CREATE TRIGGER trgAfterUpdate ON [dbo].[Employee\_Test]   
FOR UPDATE  
AS  
 declare @empid int;  
 declare @empname varchar(100);  
 declare @empsal decimal(10,2);  
 declare @audit\_action varchar(100);  
  
 select @empid=i.Emp\_ID from inserted i;   
 select @empname=i.Emp\_Name from inserted i;   
 select @empsal=i.Emp\_Sal from inserted i;   
   
 if update(Emp\_Name)  
 set @audit\_action='Updated Record -- After Update Trigger.';  
 if update(Emp\_Sal)  
 set @audit\_action='Updated Record -- After Update Trigger.';  
  
 insert into Employee\_Test\_Audit(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)   
 values(@empid,@empname,@empsal,@audit\_action,getdate());  
  
 PRINT 'AFTER UPDATE Trigger fired.'  
GO

The AFTER UPDATE Trigger is created in which the updated record is inserted into the audit table. There is **no logical table updated like the logical table inserted.** We can obtain the updated value of a field from the update(column\_name) function. In our trigger, we have used, if update(Emp\_Name) to check if the column Emp\_Name has been updated. We have similarly checked the column Emp\_Sal for an update.

Let’s update a record column and see what happens.

Hide Copy Code

update Employee\_Test set Emp\_Sal=1550 where Emp\_ID=6

This inserts the row into the audit table as:

Hide Copy Code

6 Chris 1550.00 Updated Record -- After Update Trigger. 2008-04-26 12:38:11.843

## **(c) AFTER DELETE Trigger**

This trigger is fired after a delete on the table. Let’s create the trigger as:

Hide Copy Code

CREATE TRIGGER trgAfterDelete ON [dbo].[Employee\_Test]   
AFTER DELETE  
AS  
 declare @empid int;  
 declare @empname varchar(100);  
 declare @empsal decimal(10,2);  
 declare @audit\_action varchar(100);  
  
 select @empid=d.Emp\_ID from deleted d;   
 select @empname=d.Emp\_Name from deleted d;   
 select @empsal=d.Emp\_Sal from deleted d;   
 set @audit\_action='Deleted -- After Delete Trigger.';  
  
 insert into Employee\_Test\_Audit  
(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)   
 values(@empid,@empname,@empsal,@audit\_action,getdate());  
  
 PRINT 'AFTER DELETE TRIGGER fired.'  
GO

In this trigger, the deleted record’s data is picked from the **logical deleted table** and inserted into the audit table. Let’s fire a delete on the main table. A record has been inserted into the audit table as:

Hide Copy Code

6 Chris 1550.00 Deleted -- After Delete Trigger. 2008-04-26 12:52:13.867

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

Hide Copy Code

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

Specific Triggers can be enabled or disabled as:

Hide Copy Code

ALTER TABLE Employee\_Test DISABLE TRIGGER trgAfterDelete

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

## **(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.

Let’s create an Instead Of Delete Trigger as:

Hide Copy Code

CREATE TRIGGER trgInsteadOfDelete ON [dbo].[Employee\_Test]   
INSTEAD OF DELETE  
AS  
 declare @emp\_id int;  
 declare @emp\_name varchar(100);  
 declare @emp\_sal int;  
   
 select @emp\_id=d.Emp\_ID from deleted d;  
 select @emp\_name=d.Emp\_Name from deleted d;  
 select @emp\_sal=d.Emp\_Sal from deleted d;  
  
 BEGIN  
 if(@emp\_sal>1200)  
 begin  
 RAISERROR('Cannot delete where salary > 1200',16,1);  
 ROLLBACK;  
 end  
 else  
 begin  
 delete from Employee\_Test where Emp\_ID=@emp\_id;  
 COMMIT;  
 insert into Employee\_Test\_Audit(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)  
 values(@emp\_id,@emp\_name,@emp\_sal,'Deleted -- Instead Of Delete Trigger.',getdate());  
 PRINT 'Record Deleted -- Instead Of Delete Trigger.'  
 end  
 END  
GO

This trigger will prevent the deletion of records from the table where Emp\_Sal > 1200. If such a record is deleted, the Instead Of Trigger will rollback the transaction, otherwise the transaction will be committed. Now, let’s try to delete a record with the Emp\_Sal >1200 as:

Hide Copy Code

delete from Employee\_Test where Emp\_ID=4

This will print an error message as defined in the RAISE ERROR statement as:

Hide Copy Code

Server: Msg 50000, Level 16, State 1, Procedure trgInsteadOfDelete, Line 15  
Cannot delete where salary > 1200

And this record will not be deleted.

In a similar way, you can code Instead of Insert and Instead Of Update triggers on your tables.

## **Conclusion**

In this article, I took a brief introduction of triggers, explained the various kinds of triggers – After Triggers and Instead Of Triggers along with their variants and explained how each of them works. I hope you will get a clear understanding about the Triggers in SQL Server and their usage.

## **License**