ARTICLE

READER LEVEL:

Creating and managing Triggers in SQL Server 2005/2008

By Vishal Nayan on Sep 02, 2011

In this article you will see how to Create and manage Triggers in SQL Server 2005/2008.

83.5 k 0 0

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.

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.

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.
4. We cannot use following commands in DML trigger
    - ALTER DATABASE
    - CREATE DATABASE
    - DISK DATABASE
    - LOAD DATABASE
    - RESTORE DATABASE
5. 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.

For all examples shared below I have used Pubs database. You can download its msi file from here and then attach .mdf file in your SQL Sever 2008.

http://www.microsoft.com/downloads/en/details.aspx?FamilyId=06616212-0356-46A0-8DA2-EEBC53A68034&displaylang=en

```
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:

1. Provides an alternative to the AFTER trigger that was heavily utilized in prior versions of SQL Server.
2. It performs its actions instead of the action that fired it.
3. 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.
4. 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.
- We can use INSTEAD OF triggers to simplify the process of updating multiple tables for application developers.
- Mixing Trigger Types.

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
```

Let us try to run a DDL command. See the result in below figure

Now let us look at an example that applies to server-level events. Above example was scoped at database level.

Let us create a trigger which prevents changes to the server logins. When this trigger is installed, it displays a message and rolls back any login changes that are attempted.

```
CREATE TRIGGER tr_LoginAudit
ON ALL SERVER
 FOR CREATE_LOGIN,ALTER_LOGIN,DROP_LOGIN
AS
      PRINT'You must disable the tr_LoginAudit trigger before making login changes'
      ROLLBACK
 GO
```

C) Using CLR Trigger:

1. CLR triggers are trigger based on CLR.
2. CLR integration is new in SQL Server 2008. It allows for the database objects (such as trigger) to be coded in .NET.
3. Object that have heavy computation or require reference to object outside SQL are coded in the CLR.

4. We can code both DDL and DML triggers by using a supported CLR language like C#.

Let us follow below simple steps to create a CLR trigger;

Step 1: Create the CLR class. We code the CLR class module with reference to the namespace required to compile CLR database objects.

Add below reference;

```
using Microsoft.SqlServer.Server;
using System.Data.SqlTypes;
```

So below is the complete code for class;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Sql;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;
using System.Data.SqlTypes;
using System.Text.RegularExpressions;

namespace CLRTrigger
 {
    public class CLRTrigger
   {
        public static void showinserted()
        {
            SqlTriggerContext triggContext = SqlContext.TriggerContext;
            SqlConnection conn = new SqlConnection(" context connection =true ");
            conn.Open();
            SqlCommand sqlComm = conn.CreateCommand();
            SqlPipe sqlP = SqlContext.Pipe;
            SqlDataReader dr;
            sqlComm.CommandText = "SELECT pub_id, pub_name from inserted";
            dr = sqlComm.ExecuteReader();
            while (dr.Read())
                sqlP.Send((string)dr[0] + "," + (string)dr[1]);
        }

    }
 }
```

Step 2: Compile this class and in the BIN folder of project we will get CLRTrigger.dll generated. After compiling for CLRTrigger.dll, we need to load the assembly into SQL Server

Step 3: Now we will use T-SQL command to execute to create the assembly for CLRTrigger.dll. For that we will use CREATE ASSEMBLY in SQL Server.

```
CREATE ASSEMBLY   triggertest
FROM 'C:\CLRTrigger\CLRTrigger.dll'
 WITH PERMISSION_SET = SAFE
```

Step 4: The final step is to create the trigger that references the assembly. Now we will write below T-SQL commands to add a trigger on the publishers table in the Pubs database.

```
CREATE TRIGGER tri_Publishes_clr
ON publishers
FOR INSERT
```

```
AS
    EXTERNAL NAME triggertest.CLRTrigger.showinserted
```

If you get some compatibility error message run the below command to set compatibility.

```
ALTER DATABASE pubs
 SET COMPATIBILITY_LEVEL =  100
```

Step 5: Enable CLR Stored procedure on SQL Server. For this run the below code;

```
EXEC sp_configure 'show advanced options' , '1';
 reconfigure;

EXEC sp_configure 'clr enabled' , '1' ;
 reconfigure;

EXEC sp_configure 'show advanced options' , '0';
 reconfigure;
```

Step 6: Now we will run INSERT statement to the publishers table that fires the newly created CLR trigger.

```
INSERT publishers
(pub_id, pub_name)
 values ('9922','Vishal Nayan')
```

The trigger simply echoes the contents of the inserted table. The output from the trigger is based on the insertion above.

```
-----------------------------------------------------
9922,Vishal Nayan

(1 row(s) affected)
```

The line of code which is printing the query result is actually below code written in a managed environment.

```
while (dr.Read())
 sqlP.Send((string)dr[0] + "," + (string)dr[1]);
```

Conclusion:

The tri_Publishes_clr trigger demonstrates the basic steps for creating a CLR trigger. The true power of CLR triggers lies in performing more complex calculations, string manipulations and things of this nature that the can be done much more efficiently with CLR programming languages than they can in T-SQL.

ARTICLE EXTENSIONS

Contents added by avnish singh on Mar 05, 2013

Vishal Nayan

Vishal is a seasoned professional with hand on experience on Microsoft technologies. He is also a part-time trainer on framework , WPF ,WCF , Silverlight and is active in Mauritius.

Personal Blog: http://www.vishalnayan.wordpress.com

Rank 160 | 894.7k Readers | Bronze Member | MVP

RELATED ARTICLES

- Creating and Managing User Defined Functions in SQL Server 2008
- Input Parameters Scenario in SQL Stored Procedure (SQL Server 2005/2008)
- Creating and Managing Stored Procedure in SQL Server 2008
- Triggers in Sql Server
- How to Work With INSTEAD OF Trigger in SQL Server 2012
- Create Your First CLR Trigger in SQL Server 2008 Using C#
- Dynamic Query to Search by Column Value in All Referencing Tables in SQL SERVER
- How to Use the SQL Server 2005 Tuning Advisor
- Output keyword in Sql Server 2008
- Create, Delete, and Update Triggers in a Database

COMMENTS

of

COMMENT

MVPs MOST VIEWED LEGENDS NOW PRIZES REVIEWS SURVEY CERTIFICATIONS DOWNLOADS

Hosted By CBeyond Cloud Services

PHOTOS CODE SNIPPETS STUDENTS MEMBERS MEDIA KIT ABOUT US LINKS IDEAS

CONTACT US PRIVACY POLICY TERMS & CONDITIONS SITEMAP REPORT ABUSE