Triggers in SQL Server

May 15, 2019 by Ranga Babu

In this article, we will review triggers in SQL Server, different types of trigger events, trigger order and NOT FOR REPLICATION in triggers.
A trigger is a database object that runs automatically when an event occurs. There are three different types of events.

- DML Events
- DDL Events
- LOGON Event – Logon trigger is fired when a LOGON event occurs i.e. when a user session is being established

DML Triggers in SQL Server

DML triggers in SQL Server are fired when a DML event occurs. i.e. when data is inserted/ updated/deleted in the table by a user.

Creating triggers for a DML event

Let us create some sample tables and triggers in SQL Server.

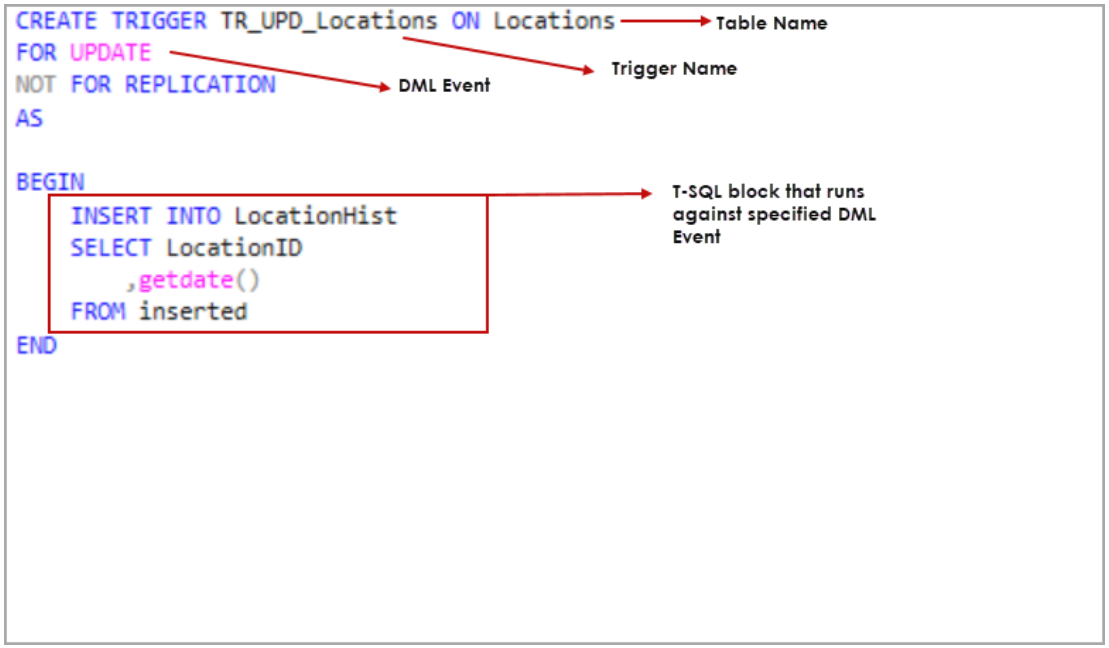
```
CREATE TABLE Locations (LocationID int, LocName varchar(100))
```

```
CREATE TABLE LocationHist (LocationID int, ModifiedDate DATETIME)
```

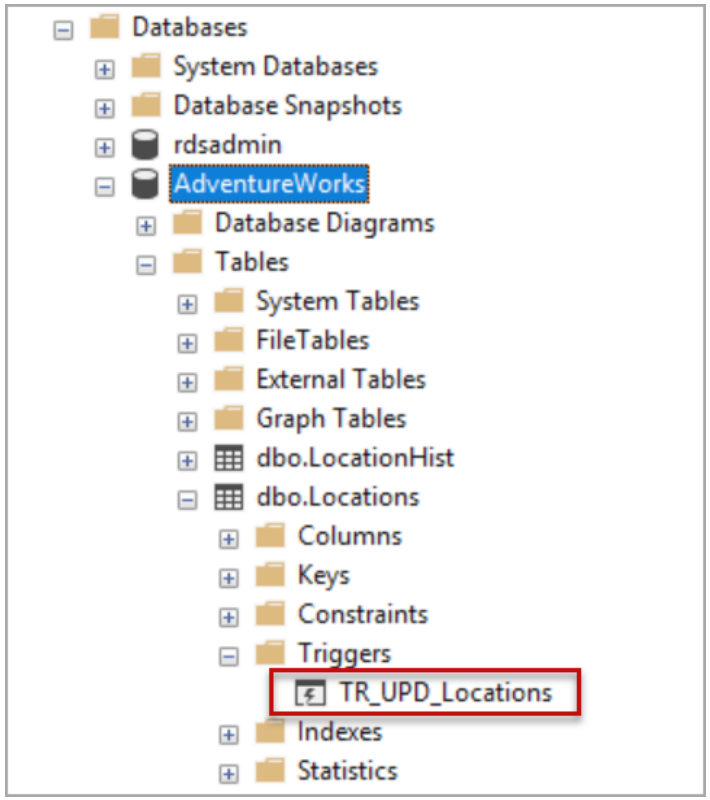
We can create a DML trigger for a specific event or multiple events. The triggers in SQL Server(DML) fire on events irrespective to the number of rows affected.

Below is the sample syntax for creating a DML trigger for update event.

```
CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
NOT FOR REPLICATION
AS

BEGIN
  INSERT INTO LocationHist
  SELECT LocationID
    ,getdate()
  FROM inserted
END
```



These triggers are created at the table level. Upon successful creation of trigger, we can see the triggers by navigating to **Triggers** folder at table level. Please refer to the below image.



Instead of triggers in SQL Server

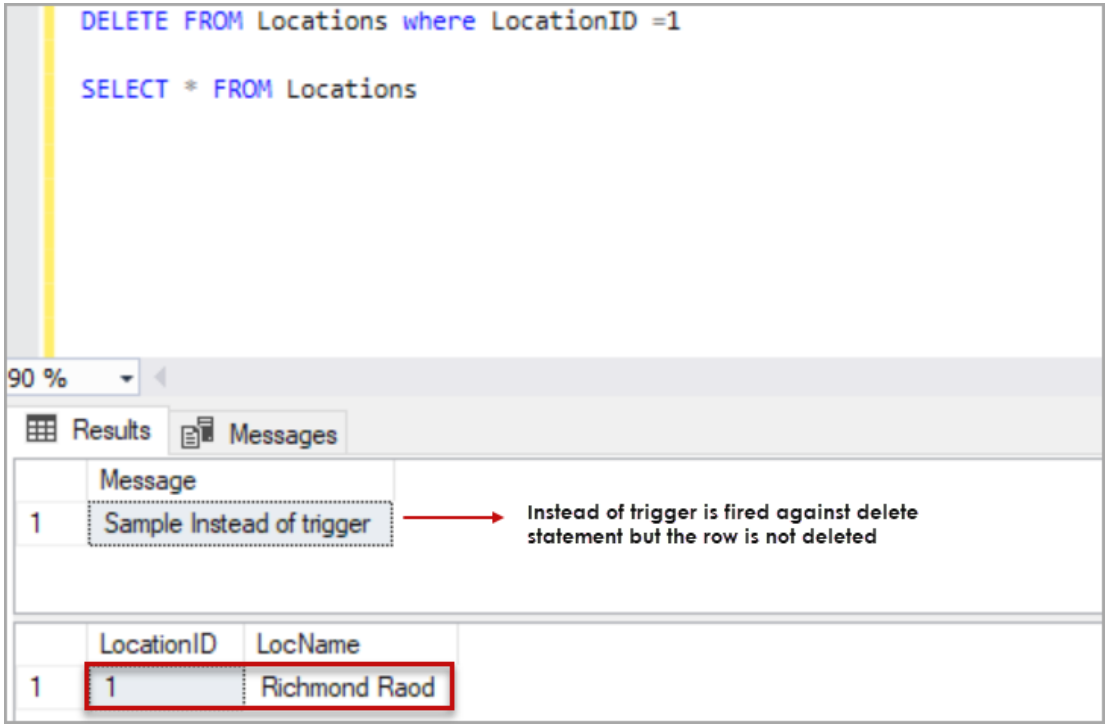
These triggers are fired before the DML event and the actual data is not modified in the table.

For example, if we specify an instead of trigger for delete on a table, when delete statement is issued against the table, the instead of trigger is fired and the T-SQL block inside the triggers in SQL Server is executed but the actual delete does not happen.

T-SQL Syntax for creating an instead of trigger

```
CREATE TRIGGER TR_DEL_Locations ON Locations
INSTEAD OF DELETE
AS
BEGIN
  Select 'Sample Instead of trigger' as [Message]
```

```
END
```



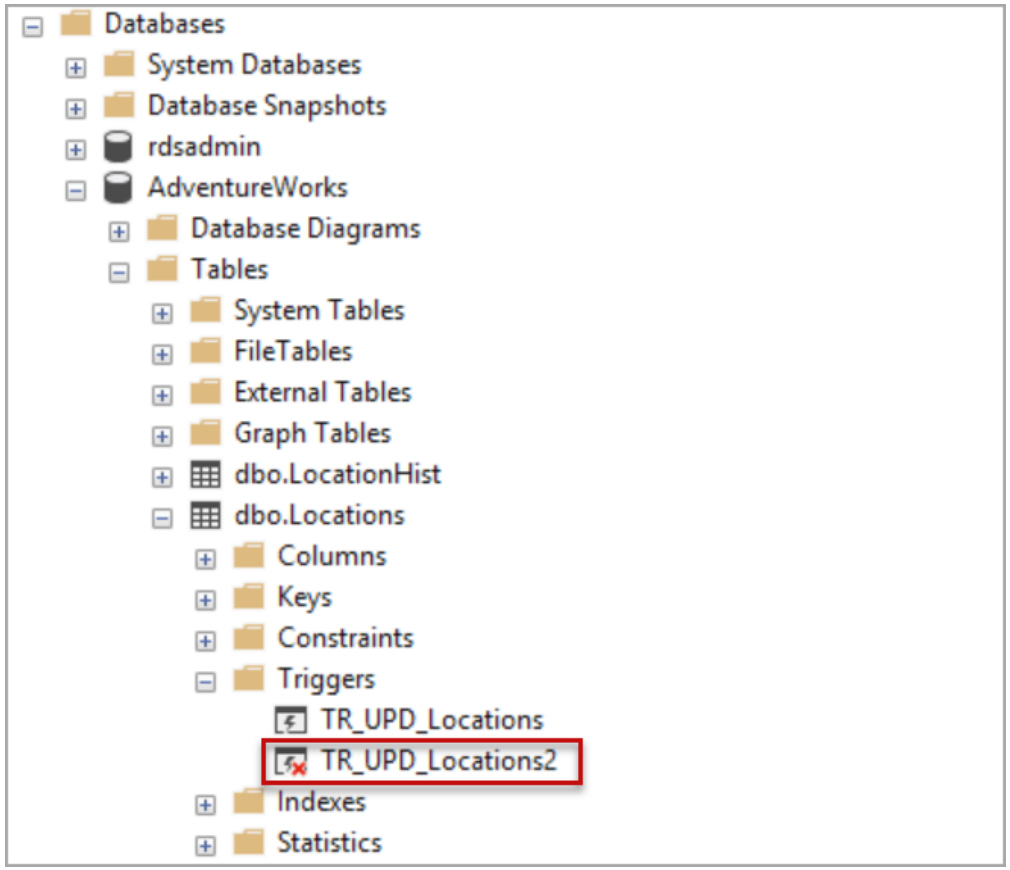
- If there are multiple triggers along with instead of trigger on the table, the instead of trigger is fired first in the order
- INSTEAD of triggers can be created on views
- we can define only one instead of trigger per INSERT, UPDATE, or DELETE statement on a table or view

Enabling and disabling DML triggers on a table

Navigate to triggers folder at the table level, select the trigger, Right click on trigger and Click on **Enable/Disable** to Enable or disable the trigger using **SSMS**.

Disabling specific SQL Server trigger on a table using T-SQL.

```
DISABLE TRIGGER TR_UPD_Locations2 on Locations
```



Enabling specific trigger on the table using T-SQL.

```
ENABLE TRIGGER TR_UPD_Locations2 on Locations
```

To enable all triggers on a table, use below syntax.

```
ENABLE TRIGGER ALL ON Locations
```

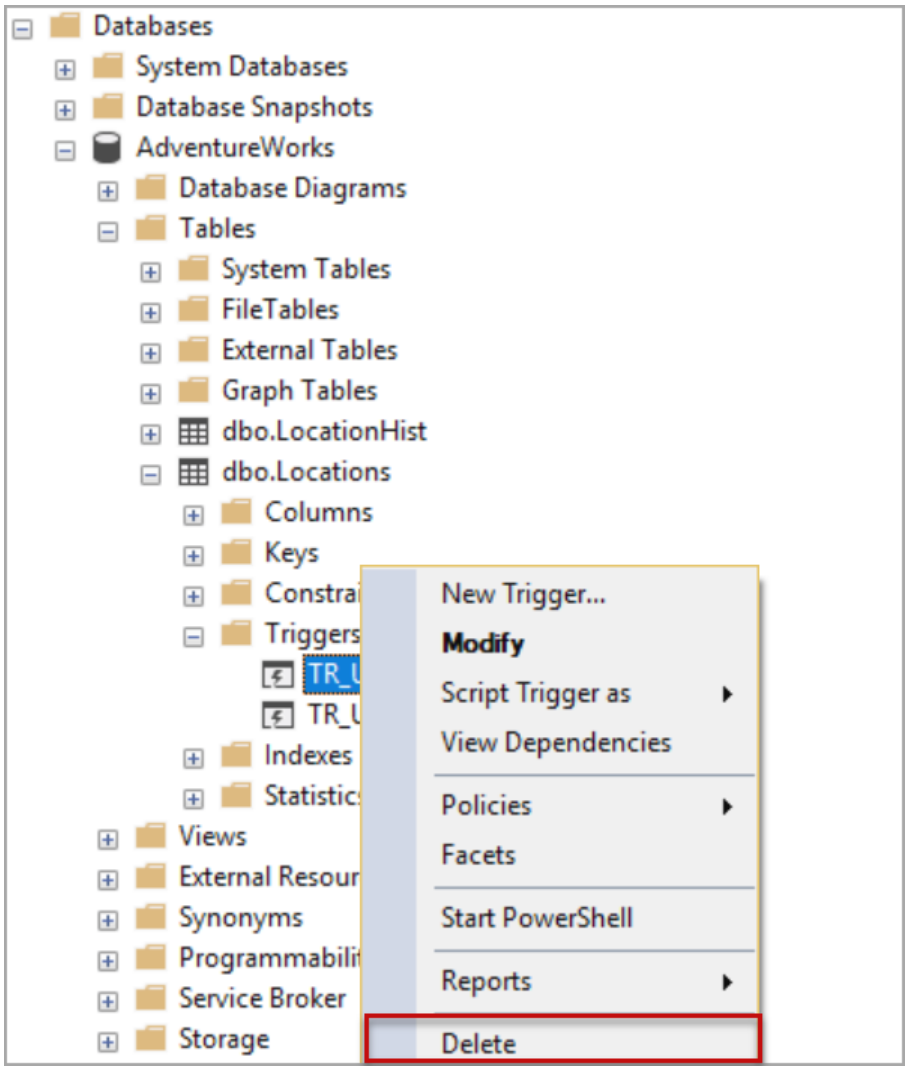
To disable all triggers on a table, use below syntax. This statement is not supported if the table is part of merge replication.

```
DISABLE TRIGGER ALL ON Locations
```

Dropping a trigger on a table

Dropping a trigger on a table:

To drop a DML trigger on the table using SQL Server management studio, navigate to the **Triggers** folder under the table. Select the table you want to drop, Right click on the trigger and click on **Delete**. Click **Ok**.



T-SQL to drop a trigger on the table.

```
DROP TRIGGER TRL_UPD_Locations2
```

Dropping a table will drop all the SQL Server triggers on the table along with the table.

DDL Triggers

DDL triggers in SQL Server are fired on DDL events. i.e. against create, alter and drop statements, etc. These triggers are created at the database level or server level based on the type of DDL event.

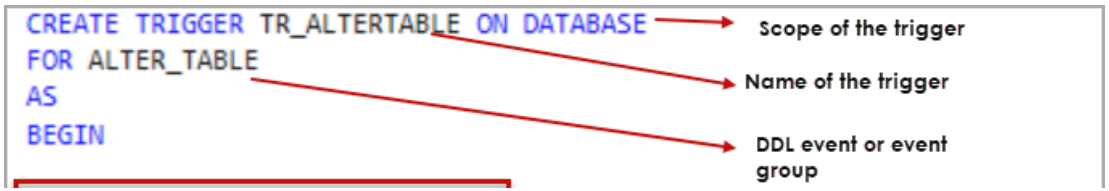
These triggers are useful in the below cases.

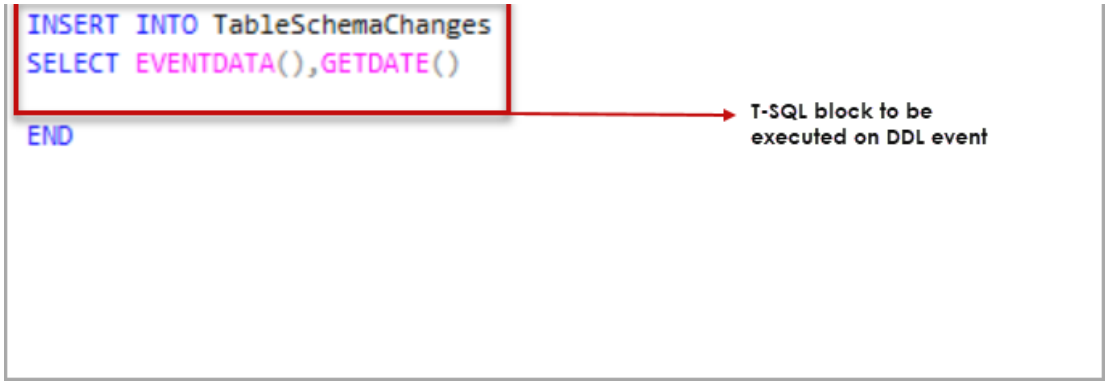
- Prevent changes to the database schema
- Audit database schema changes
- To respond to a change in the database schema

Creating a DDL trigger

Below is the sample syntax for creating a DDL trigger for ALTER TABLE event on a database which records all the alter statements against the table. You can write your custom code to track or audit the schema changes using EVENTDATA().

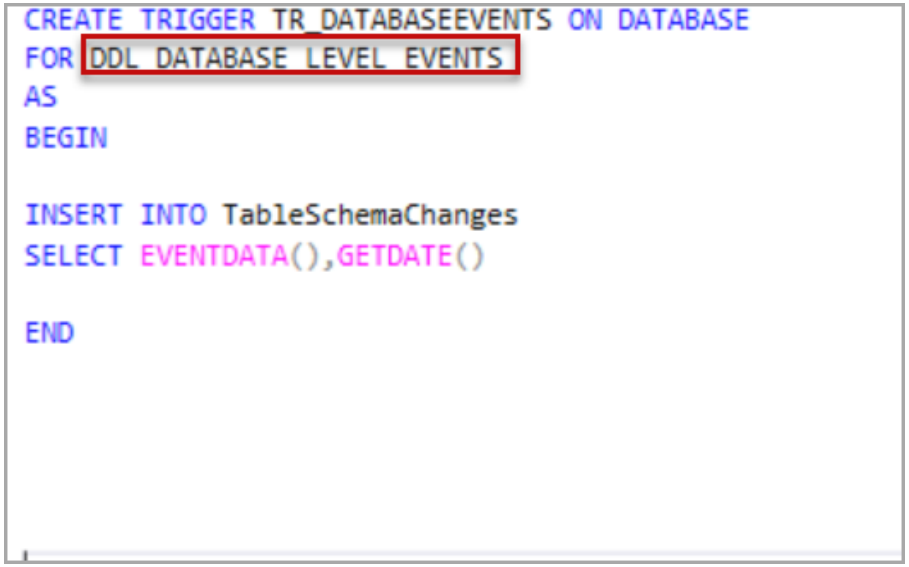
```
CREATE TABLE TableSchemaChanges (ChangeEvent xml, DateModified datetime)

CREATE TRIGGER TR_ALTERTABLE ON DATABASE
FOR ALTER_TABLE
AS
BEGIN

INSERT INTO TableSchemaChanges
SELECT EVENTDATA(),GETDATE()

END
```



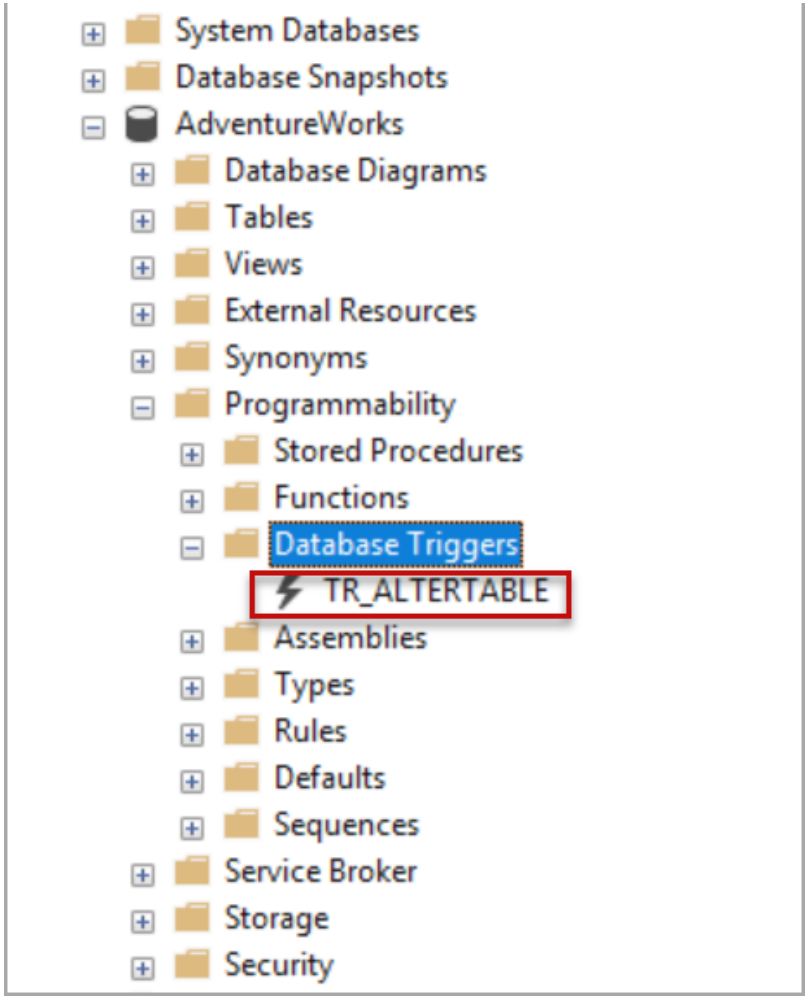


You can specify an event group which consists of different DDL events. If we specify an event group while creating a DDL trigger, the trigger is fired when a DDL event in the group occurs.

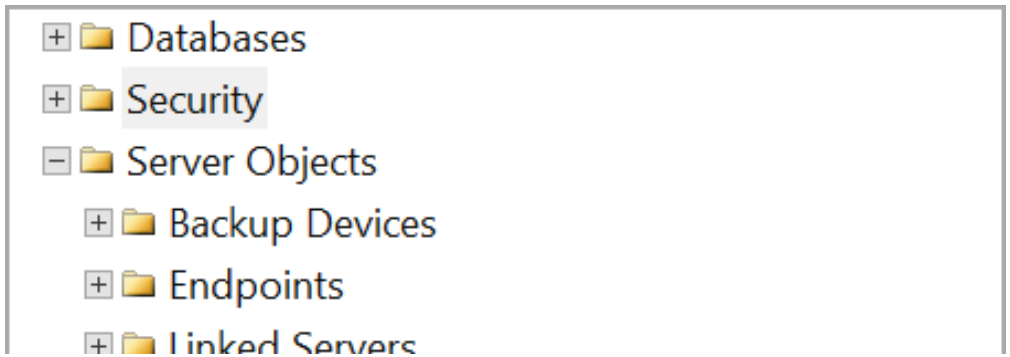
For example, if we want to create a trigger for all DDL events at the database level, we can just specify the DDL_DATABASE_LEVEL_EVENTS event group as shown in the below image.



To view database level triggers, Login to the server using SQL Server management studio and navigate to the database. Expand the database and navigate to **Programmability** -> **Database Triggers.**



To view triggers at the server level, Login to Server using SSMS and navigate to **Server Objects** and then **Triggers** folder.



Enabling and disabling DDL triggers

Use below T-SQL syntax to disable or enable the DDL trigger at the database level.

```
ENABLE TRIGGER TR_DATABASEEVENTS ON DATABASE
GO

DISABLE TRIGGER TR_DATABASEEVENTS ON DATABASE
GO
```

Use below T-SQL syntax to drop a DDL trigger which is created at the database level.

```
DROP TRIGGER TR_DATABASEEVENTS ON DATABASE
```

LOGON Triggers in SQL Server

These triggers in SQL Server fire in response to a LOGON event. LOGON triggers fire after successful authentication and before establishing the user session.

LOGON triggers are created at the server level and are useful below cases.

1. To audit login activity
2. To control the login activity

Creating LOGON triggers

You can use EVENTDATA() and write your custom code to track or control the connections. Here I am creating simple triggers in SQL

Server for LOGON event. Below is the sample syntax for creating a LOGON trigger.

```
CREATE TABLE LoginActivity (LOGONEvent XML ,Logintime datetime)

CREATE TRIGGER [track_logins] ON ALL SERVER
FOR LOGON AS

BEGIN
  INSERT INTO LoginActivity
  SELECT EVENTDATA()
    ,GETDATE()
END
```

We must be cautious while creating these triggers as login may fail if the trigger execution fails or if you do not have access to objects referenced in the LOGON trigger. In such cases, the only member of the sysadmin role can connect to the server using a dedicated administrator connection. So, it is always better to enable dedicated administrator connection when using these triggers.

Enabling and disabling LOGON triggers

Use below T-SQL syntax to disable or enable the LOGON trigger.

```
ENABLE TRIGGER track_logins ON ALL SERVER
GO

DISABLE TRIGGER track_logins ON ALL SERVER
GO
```

Use below T-SQL syntax to drop a LOGON trigger.

```
DROP TRIGGER track_logins ON ALL SERVER
```

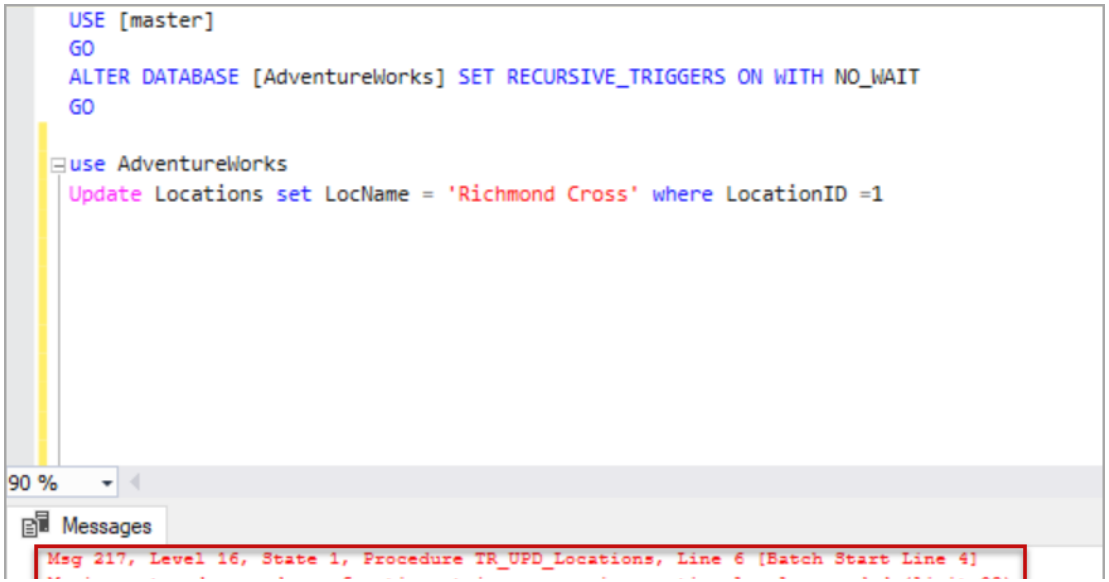
Direct recursion

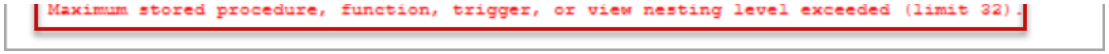
Direct recursion is a case where the SQL Server trigger on the table is fired and performs an action which again triggers the same trigger.

For example, please refer to below sample trigger for an update which is direct recursive.

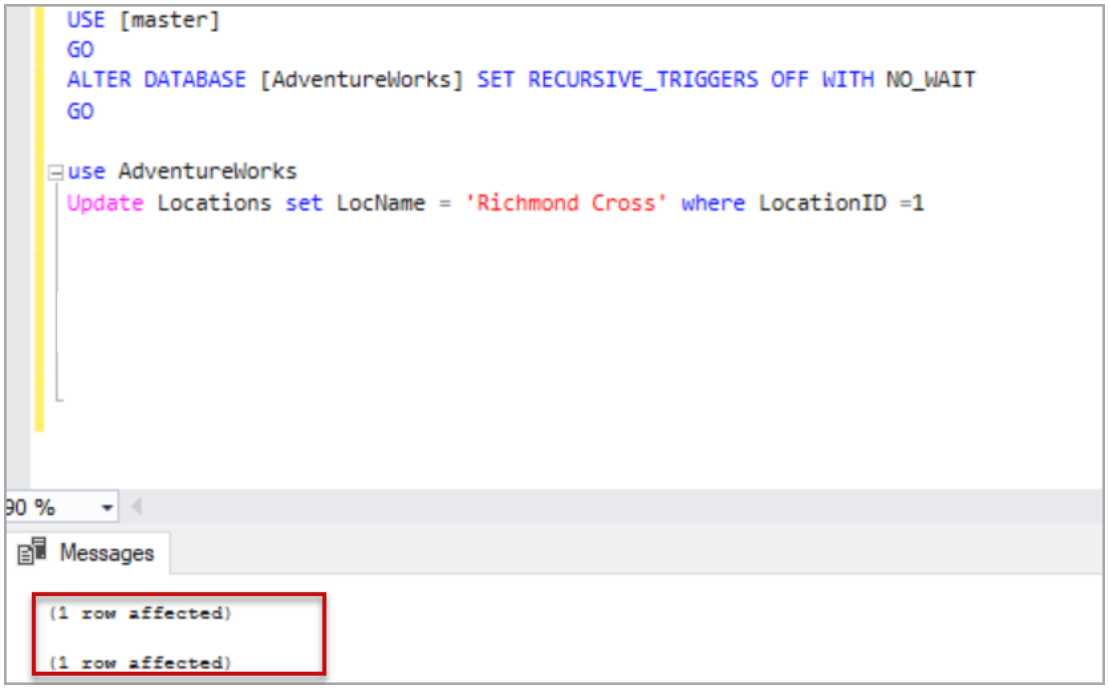
```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Locations](
  [LocationID] [int] NULL,
  [LocName] [varchar](100) NULL,
  DateUpdated datetime
) ON [PRIMARY]
GO

INSERT INTO Locations VALUES(1,'Richmond Road', NULL)

CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
AS

BEGIN
  Update Locations set DateUpdated =GETDATE()
END
```

Direct recursion can be controlled by a database setting **RECURSIVE_TRIGGERS**. If the setting is on, then the above trigger throws an error.

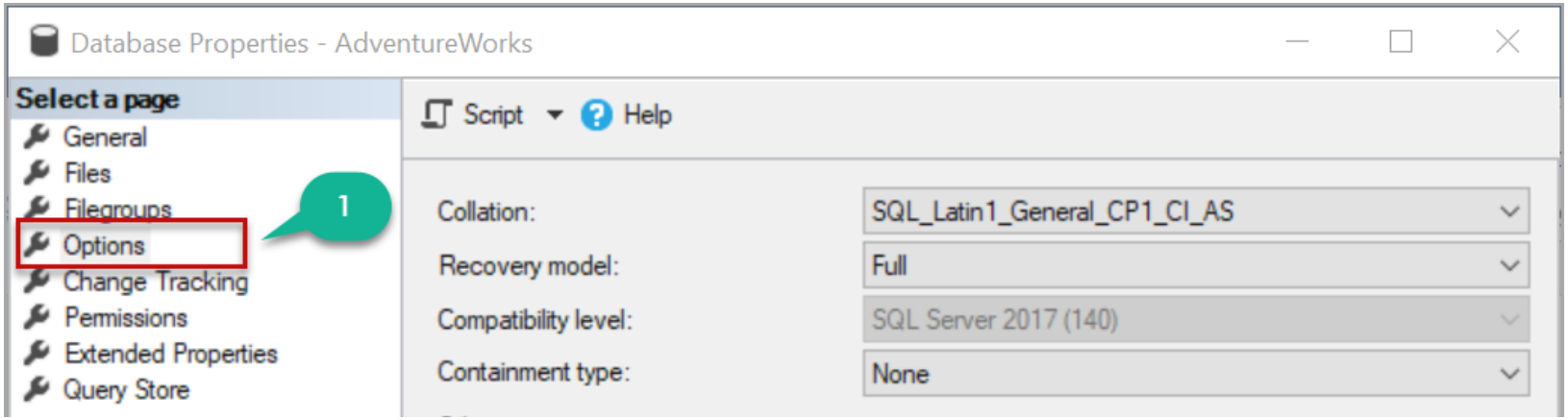


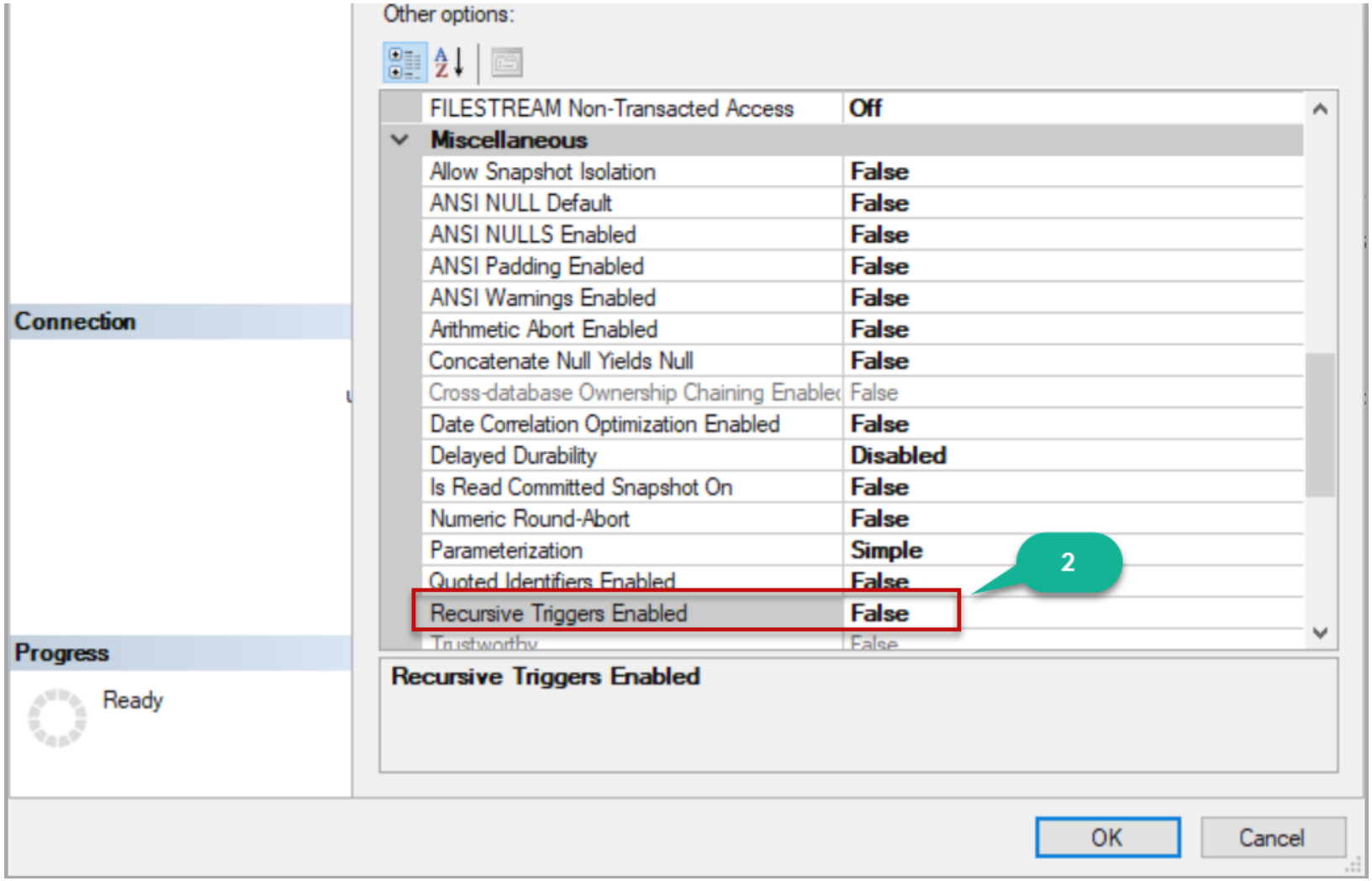


If the database setting RECURSIVE_TRIGGERS is off, then the trigger is fired only once and does not loop.



To change RECURSIVE_TRIGGERS setting using SSMS, navigate to the database, right click on the database and select **Properties.** Click on **Options** and change the setting to the option you want.





To set the RECURSIVE_TRIGGERS OFF using T-SQL, use below statement and replace the database name with your database name.

```
ALTER DATABASE [AdventureWorks] SET RECURSIVE_TRIGGERS OFF WITH NO_WAIT
GO
```

To set the RECURSIVE_TRIGGERS ON using T-SQL, use below statement and replace the database name with your database name.

```
ALTER DATABASE [AdventureWorks] SET RECURSIVE_TRIGGERS ON WITH NO_WAIT
GO
```

Indirect Recursion

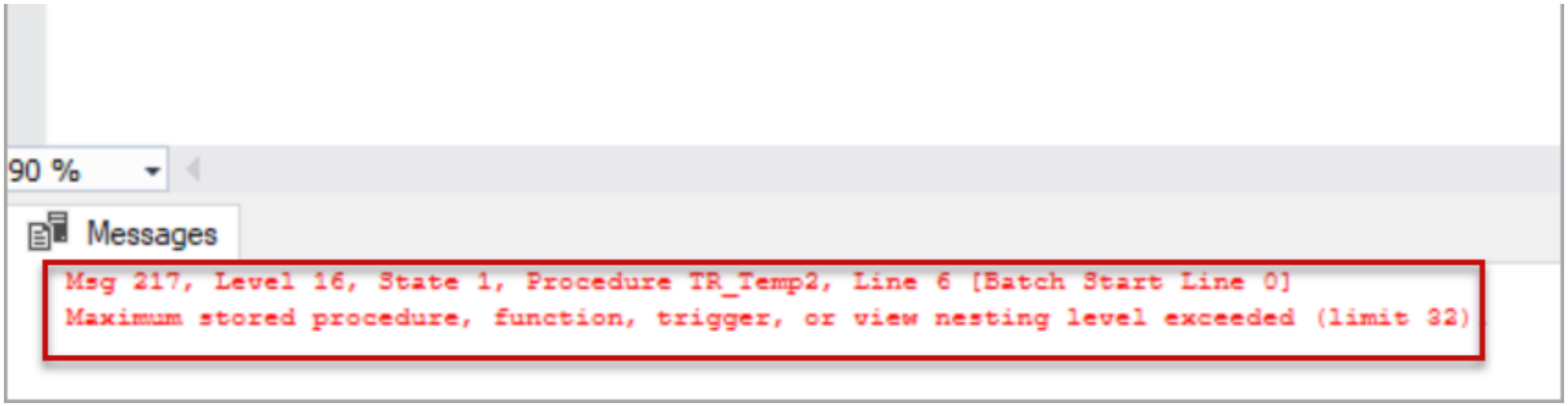
This is a case where a trigger is fired and invokes another trigger of the same type.

Below is the sample trigger for indirect recursion.

```
CREATE TABLE Temp1 (id int)
GO

INSERT INTO Temp1 values (1),(2)
GO

CREATE TABLE Temp2 (id int)
GO

INSERT INTO Temp2 values (1),(2)
GO

CREATE TRIGGER TR_Temp1 on Temp1
FOR UPDATE
AS
BEGIN
UPDATE TEMP2 set ID ='5' where id in (select id from inserted)
END
GO

CREATE TRIGGER TR_Temp2 on Temp2
FOR UPDATE
AS
BEGIN
UPDATE Temp1 set ID ='5' where id in (select id from inserted)
END
```

Now when we update a value in the table Temp1, the trigger TR_Temp1 is fired which updates Temp2 table. TR_Temp2 is fired and updates Temp1 table which causes TR_Temp1 to fire again.

```
UPDATE TEMP1 SET ID =6 WHERE ID =1
```



This behavior can be controlled by setting **nested triggers** off.

```
EXEC sp_configure 'nested triggers', 0 ;
GO
```

SQL Server trigger order

SQL Server allows multiple triggers on the table for the same event and there is no defined order of execution of these triggers.

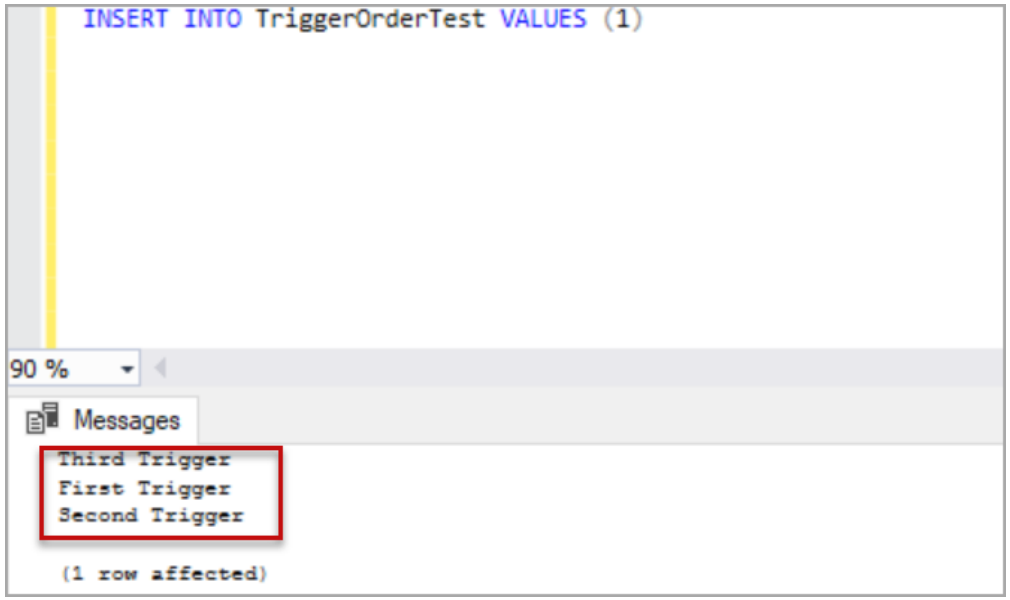
We can set the order of a trigger to either first or last using procedure sp_settriggerorder. There can be only one first or last trigger for each statement on a table.

Below is the sample syntax for setting the order of trigger to first for the INSERT statement.

```
CREATE TABLE TriggerOrderTest (id int)
GO

CREATE TRIGGER TR_1 ON TriggerOrderTest
FOR INSERT
as
BEGIN
PRINT 'First Trigger'
END
GO

CREATE TRIGGER TR_2 ON TriggerOrderTest
```

```
FOR INSERT
as
BEGIN
PRINT 'Second Trigger'
END
GO

CREATE TRIGGER TR_3 ON TriggerOrderTest
FOR INSERT
as
BEGIN
PRINT 'Third Trigger'
END
GO

sp_settriggerorder  @triggername ='TR_3'
    ,  @order = 'FIRST'
    ,  @stmttype =  'INSERT'
```

Now, when the data is inserted into the table "TriggerOrderTest" INSERT event occurs and the trigger TR_3 fires first.



In case DDL triggers we must specify the namespace parameter which is the scope of the SQL Server trigger in the stored procedure sp_settriggerorder.

Below is the sample syntax for setting the DDL trigger order

Below is the sample syntax for setting the DDL trigger order.

```
sp_settriggerorder  @triggername ='DDL_3'
    ,  @order = 'FIRST'
    ,  @stmttype =  'ALTER_TABLE'
   ,  @namespace =   'DATABASE'
```

NOT FOR REPLICATION

NOT FOR REPLICATION indicates that the trigger should not fire when the replication agent syncs the data changes to the subscriber.

For example, if you are replicating both Locations and LocationHist. Now when you update a record on Location the trigger is fired, inserts record in the history table. When these changes sync to another end (subscribers) there is no need of trigger to be fired again. So, if we mark the trigger for "NOT FOR REPLICATION" the trigger does not fire when replication agent sync's the changes and fires only for the data changes done by the user.

Below is the sample syntax to create a triggers in SQL Server with not for replication.

```
CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
NOT FOR REPLICATION
AS

BEGIN
  INSERT INTO LocationHist
  SELECT LocationID
    ,getdate()
  FROM inserted
END
```

If you want the triggers in SQL Server to be fired when the replication agent sync data changes to another end, just create the trigger without specifying "NOT FOR REPLICATION".

See more

To track data / DML changes in SQL Server databases, check out ApexSQL Trigger, a trigger based auditing tool that allows you to create

self-auditing apps with triggers in SQL Server

Ranga Babu

SQL Server DBA, Developer with good experience in SQL Server administration, development, performance tuning, monitoring, high availability and disaster recovery technologies

---

Related Posts:

1. **SQL Server MERGE Statement overview and examples**
2. **Limit SQL Server Login Authentication scope using a Logon Trigger**
3. **Nested Triggers in SQL Server**
4. **Disabling triggers in SQL Server for a specific session**
5. **Merge SQL Server replication parameterized row filter issues**

Auditing

ALSO ON SQL SHACK

Visual Studio Code (VS Code) for SQL …

7 months ago • 1 comment

This article gives an overview of Visual Studio Code for SQL Server …

Don’t fear SQL Server performance tuning

5 months ago • 1 comment

This article mentions 5 basic tips for SQL Server performance tuning.

Exporting SSRS reports to multiple …

5 months ago • 1 comment

This article will discuss how to export SSRS reports to multiple sheets in …

SQL database hotfix testing with tSQLt

6 months ago • 1 comment

This article will show how to use the tSQLt framwork to test database hotfix

How to use and conditi

2 months ago •

In this article, v how to use the and Conditions

Tajudeen Noorudeen • 3 months ago

Hi, Very good article, but it is not mentioned about the 'after delete' trigger. Anyways.. Thank you.