DDL Triggers

Source	http://dotnet.sys-con.com/read/253397.htm

SQL Server 2005 supports DDL Triggers. DDL Triggers are triggers that fire in response to data definition language (DDL) statements such as CREATE TABLE or UPDATE STATISTICS. They're similar to the data manipulation language (DML) triggers that we've been using for years, except that they're tied to a database or server instead of a table or view. With DDL Triggers we can write code that runs in response to changes made to server and database objects. This can be a very powerful tool. (It becomes even more powerful when used with SQL Server's CLR Integration feature, which allows SQL Server objects to be created with our choice of .NET language.) DDL Triggers can be used for many purposes, but most commonly for change tracking and prevention.

This article introduces DDL Triggers and shows how to use them to track and prevent changes to database objects. We'll discuss trigger creation, trigger deletion, and trigger security. We'll also walk through some examples of typical DDL Trigger use. Along the way, we'll see how we can use XQuery to retrieve specific information about the event that caused the trigger to fire. Although a thorough knowledge of XQuery isn't critical, it's definitely beneficial. You might want to refer to SQL Server Books Online for more information.

Managing DDL Triggers

DDL Triggers are managed in much the same way that DML Triggers are managed. You use the CREATE TRIGGER, ALTER TRIGGER, and DROP TRIGGER statements. As you might expect, however, there are subtle differences. The first difference is scope. DML Triggers work against a particular table or view, but DDL Triggers work against a database or server. When managing DDL Triggers, you use the ON DATABASE clause or the ON ALL SERVER clause to specify the scope. Next, DDL Triggers fire in response to particular events or groups of events. You use the FOR clause to specify them. Another important difference is that DDL Triggers can't be declared with the INSTEAD OF clause because these triggers can't fire instead of the indicated event like DML Triggers can. A final, but important, difference is that the ON DATABASE or ON ALL SERVER clause must be specified in all CREATE TRIGGER, ALTER TRIGGER or DROP TRIGGER statements. If you don't specify one of these clauses, SQL Server will assume that you are referring to a DML Trigger instead of a DDL Trigger. Your statement will then fail to work properly.

Let's walk through an example.

LISTING 1 - SIMPLE TRIGGER

```
USE AdventureWorks
GO
-- create a database DDL trigger
CREATE TRIGGER trgDropTable
ON DATABASE
FOR DROP_TABLE
AS
 PRINT 'Table deletion is not allowed.'
 ROLLBACK TRANSACTION
GO
-- create a test table
CREATE TABLE Test(ColA INT)
GO
-- attempt to drop the test table
-- this will fail
DROP TABLE TEST
GO
-- drop the trigger
DROP TRIGGER trgDropTable ON DATABASE
GO
-- attempt to drop the test table
-- this will now succeed
DROP TABLE TEST
GO
```

In Listing 1 you can see the T-SQL code that creates a trigger that will fire in response to any DROP TABLE statement that is executed in the AdventureWorks database. The CREATE TRIGGER statement contains the ON DATABASE clause, which indicates that the scope of the trigger is the current database. The FOR DROP_TABLE clause specifies that the trigger should fire in response to a DROP TABLE statement. (See "DDL Events for Use with DDL Triggers" in the SQL Server 2005 Books Online for a list of available DDL events.) In the trigger body, I've included a PRINT statement and a ROLLBACK TRANSACTION statement. This trigger will print an explanatory message and then roll back the current transaction. This effectively "prevents" any DROP TABLE statement from

executing. Listing 1 also contains code to create and drop a test table. Because of the trigger, the DROP TABLE statement will fail, producing the output shown in Listing 2.

LISTING 2 - DROP TABLE PREVENTED

```
Table deletion is not allowed.
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger.
The batch has been aborted.
```

(See Figure 1 for an example of the output produced by a DROP TABLE command issued inside SQL Server Management Studio.) Next, the code drops the trigger (note the ON DATABASE clause) and then successfully drops the test table.

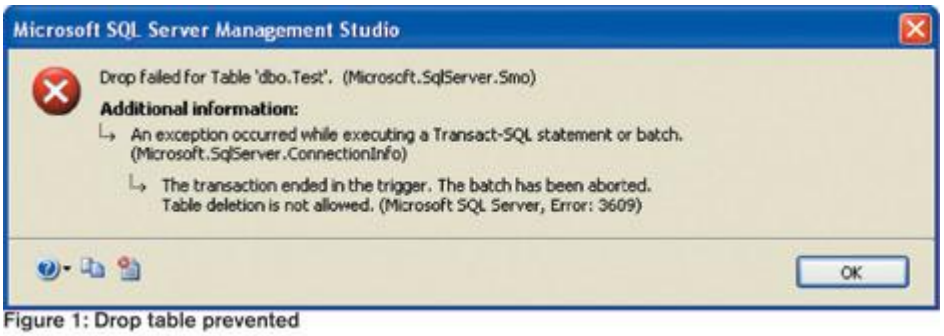

Figure 1: Drop table prevented

Auditing Changes

The previous example "prevented" changes by rolling back the transaction that caused them. While this works well, it's not always what's needed. Sometimes we want to allow changes, but we must have a record of what changes were made and who made them. DDL Triggers make this easy. In Listing 3, I've written code that creates a trigger that records information about any change made to the AdventureWorks database. This code is more complex so let's walk through it a little more slowly. First, I create a table to hold the desired audit information. In this case, I want to know which command was executed by which user and at what time. Next, I create the trigger. Once again, I use the ON DATABASE clause to specify the current database as the trigger scope. Next, however, I use the ON DDL_DATABASE_LEVEL_EVENTS clause to specify that this trigger should fire in response to any change event in the entire database. (This demonstrates the power of Event Groups. See "Event Groups for Use with DDL Triggers" in the SQL Server 2005 Books Online for a list of available DDL event groups.) The trigger body performs three tasks. First, it uses the EVENTDATA function to obtain an XML document that contains information about the event that caused the trigger to fire. The EVENTDATA function

returns this data as an instance of the XML data type, and this data is stored in the @Data variable. Next, the QUERY method is used to get specific pieces of information from the XML document stored in the @Data variable. (The QUERY method is one of the five methods provided by the XML data type. This particular method accepts an XQuery query string, and returns an instance of the XML data type.) This process is repeated twice; once to get the text of the executed command (//TSQLCommand/CommandText), and once to get the time when the command was executed (//PostTime). The standard SYSTEM_USER function provides the name of the user who invoked the command. Finally, the collected information in inserted into the DDLAudit table.

The rest of the code in Listing 3 tests the trigger. A test table is created and dropped and then the contents of the DDLAudit table are shown. See Figure 2 for the results.

	Command	PostUser	PostTime
1	CREATE TABLE Test(ColA INT)
ACHFOOD\jdixon	2006-05-15T13:34:01.407
2	DROP TABLE TEST
ACHFOOD\jdixon	2006-05-15T13:34:01.467

Figure 2: Audit table rows

You can see the commands that were executed, the user who executed the commands, and the time when the commands were executed.

Permissions

You may be wondering how all of this helps. If a developer has permission to create and drop tables, why can't he or she disable the database-scoped trigger before a change and re-enable it afterwards? The answer is that different permissions are needed for these actions. I can grant a developer full rights to a database, but deny him or her the ALTER ANY DATABASE DDL TRIGGER permission. The developer will then be able to make practically any change to the database, but not be able to affect the trigger that logs those changes. Similarly, managing server-scoped triggers requires the CONTROL SERVER permission. Users lacking the proper permissions won't be able to delete or disable the DDL triggers.

Summary

DDL Triggers bring a new level of control to SQL Server 2005. They allow code to run when changes are made to database-level objects such as tables, roles, or stored procedures. They also let code run when changes are made to server-level objects such as logins, endpoints, and databases. The triggers can perform actions ranging from change prevention (ROLLBACK TRANSACTION) to change auditing (storing change information in a table) and notification (sending an e-mail to the DBA when a change is

made). In addition, because DDL Trigger management requires its own set of permissions, developers with high-level rights can be prevented from deleting or overriding the triggers.

~~~ End of Article ~~~