**Understanding DDL Triggers in SQL Server 2005**

## Introduction:

DDL triggers are a new feature in SQL Server 2005. DDL triggers are similar to DML triggers.DML triggers fires before or after issuing the UPDATE, DELETE or INSERT commands. Similarly the DDL triggers fire after the issue of CREATE, DROP or ALTER commands.

DDL Triggers can be used to:

1. Audit the DDL (Data Definition Language) events
2. Prevent changes happening to the database.

DDL triggers cannot be used as INSTEAD OF triggers. They fire after the issue of DDL commands.

TSQL 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 > [ ; ] }

<ddl_trigger_option> ::=
 [ENCRYPTION]
 [EXECUTE AS Clause]

<method_specifier>:=
 assembly_name.class_name.method_name
```

## Scope:

The DDL triggers have to be creating defining the scope of that trigger.DDL trigger fires in response to an event which has occurred in the current database or in the current server. So the scope of the DDL trigger is Database or Server.

The Database level events that can be audited are:

1. DDL Table events: Create table, Alter table, Drop table
2. DDL view events : Create view, Alter view, Drop view
3. DDL trigger events :Create trigger, Drop trigger, Alter trigger
4. DDL synonym events: Create synonym, drop synonym
5. DDL Index events: Create index, Alter index, Drop Index
6. DDL Database level security events:
    - Create User, Drop user, Alter user

- Create role, Drop role, Alter role
- Create application role, Drop application role, Alter Application role
- Create Schema, Drop Schema, Alter Schema
- Grant database access, Revoke database access, Deny Database access

7. DDL Service broker events:
   - Create Message type, Alter Message type, Drop Message type
   - Create contract, Drop contract, Alter contract
   - Create Service, Alter service, Drop Service
   - Create route, Drop route, Alter route

The server level events that can be audited are

1. Create Database, Drop Database
2. Create Login, Drop Login, Alter Login

Event type or Event group: The trigger can be created for a specific events or a group of events.

## Encryption

The text of the trigger can be encrypted using the "WITH ENCRYPTION" clause in the DDL trigger syntax.

## Execute As

It specifies the security context in which the trigger will be executed.

## Example

1)

```
Create trigger test_ddl_trigger on ALL Server for DDL_LOGIN_Events as
Print 'You can create Logins in this server.You do not have sufficient permissions'
Rollback;
```

2)

```
Create trigger test_ddl_dbtrigger on Database for Create_table as
Print 'Cannot create table due to insufficient space'
Rollback;
```

## Querying about triggers

The information about the database scoped triggers can be obtained by querying the **sys.triggers** table. Database scoped triggers are stored in the database in which they are created. So information about them can be queried by appending the database name with sys.triggers. (Adventureworks.sys.triggers).

The information about the server scoped triggers can be obtained by querying the **sys.server_triggers.**

## Eventdata Function

The DML triggers stores the new values in the **INSERTED** table and the old values in **DELETED** table.DDL trigger do not store in the inserted and deleted tables. The information about an event which fires the DDL trigger is available through the **EVENTDATA** function.

From the eventdata function,we can obtain information about the event type,the database name in which the trigger was fired,the T-SQL command text of the event that caused the trigger to fire.The output of eventdata function is of xml datatype. The eventdata function will return data only when it is referenced from inside the DDL trigger.It does not return data if it is queried by other routines.

For example, I have created trig_event_data that prevents the user from creating tables in the database.

```
Create TRIGGER trig_event_data
ON DATABASE
FOR CREATE_TABLE
AS
 PRINT 'CREATE TABLE Issued.'
 select eventdata()
 RAISERROR ('New tables cannot be created in this database.', 16, 1)
 ROLLBACK
;
```

To test the trigger, create the table "test5"

```
Create table test5(employee_id int,name_emp varchar(50))
```

The eventdata function generated the results as follows:

```
EVENT_INSTANCE>
 <EventType>CREATE_TABLE</EventType>
 <PostTime>2007-02-23T15:17:01.510</PostTime>
 <SPID>54</SPID>
 <ServerName>MNGKTR24012</ServerName>
 <LoginName>sa</LoginName>
 <UserName>dbo</UserName>
 <DatabaseName>AdventureWorks</DatabaseName>
 <SchemaName>dbo</SchemaName>
 <ObjectName>test5</ObjectName>
 <ObjectType>TABLE</ObjectType>
 <TSQLCommand>
 <SetOptions ANSI_NULLS="ON"
      ANSI_NULL_DEFAULT="ON" ANSI_PADDING="ON"
      QUOTED_IDENTIFIER="ON" ENCRYPTED="FALSE" />
 <CommandText>create table test5(employee_id int,name_emp varchar(50))
</CommandText>
 </TSQLCommand>

</EVENT_INSTANCE>
```

If the user wants only the TSQL command which caused the trigger to fire,replace the "Select.eventdata ()" by

```
SELECT EVENTDATA().value

('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')
```

We reach to the value of CommandText node by traversing /Event_Instance/TSQLCommand.The CommandText is the second sub-node within the TSQLCommand node.So 1 in the square brackets (starting with zero). The function can be modified to insert all the create table events into a log table. First create a table which will maintain a log of the create table events

```
USE AdventureWorks;

GO

CREATE TABLE create_table_log
  ( create_time datetime
    , DB_User nvarchar(100)
    , Event_type nvarchar(100)
    , TSQL nvarchar(2000));
GO
```

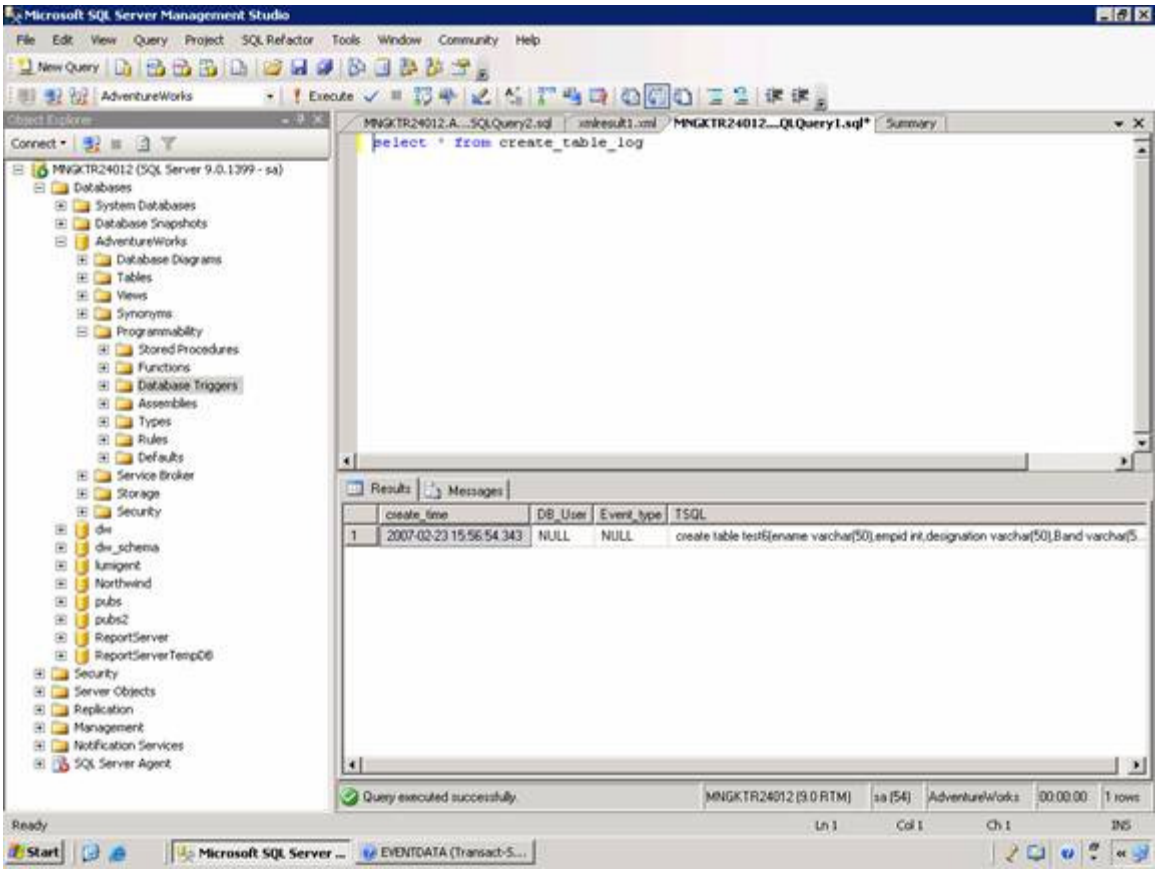
Second,create the trigger and insert the created time,user,event type and the TSQL command of the event type into the "CREATE_TABLE_LOG" table.

```
create trigger trig_create_table on database for create_table
as
Declare @data xml
set @data=Eventdata()

Insert into create_table_log values
(getdate(),
@data.value('(/EVENT_INSTANCE/LOGINNAME)[1]','nvarchar(100)'),
@data.value('(/EVENT_INSTANCE/EVENTTYPE)[1]','nvarchar(100)'),
@data.value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(200)')
)
Go
```

3)Test the trigger by issuing the command

```
createtable test6(ename
varchar(50),empid int,designation varchar(50),Band varchar(50),Salary int)
```




A record is inserted corresponding to the table "test6"

Thus DDL triggers is a powerful way of auditing the database and the server for DDL operations.