Database System

Triggers – II

Muhammad Tariq Mahmood
tariq@koreatech.ac.kr
School of Computer Science and Engineering
Korea University of Technology and Education

Triggers

- Trigger: A procedure that starts automatically if specified changes occur to the DBMS
- SQL Server implements three types of triggers:
- Data Manipulation Language (DML) triggers, which fire in response to INSERT, UPDATE, and DELETE events against tables;
- Data Definition Language (DDL) triggers, which fire in response to CREATE, ALTER, and DROP statements
- logon triggers, which fire in response to LOGON events.

CREATE TRIGGER (Transact-SQL)

- Example(DML Trigger)

```
USE AdventureWorks2012;
  GO

 IF OBJECT_ID ('Sales.reminder2','TR') IS NOT NULL
      DROP TRIGGER Sales.reminder2;
  GO

CREATE TRIGGER reminder2
  ON Sales.Customer
  AFTER INSERT, UPDATE, DELETE
  AS
     EXEC msdb.dbo.sp_send_dbmail
          @profile_name = 'AdventureWorks2012 Administrator',
          @recipients = 'danw@Adventure-Works.com',
          @body = 'Don''t forget to print a report for the sales force.',
          @subject = 'Reminder';
  GO
```

ALTER TRIGGER

- Modifies the definition of a DML, DDL, or logon trigger that was previously created by the CREATE TRIGGER statement.

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

```
ALTER TRIGGER schema_name.trigger_name
ON ( table | view )
[ WITH <dml_trigger_option> [ ,...n ] ]
(FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier> [ ; ] }

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

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

ENABLE TRIGGER (Transact-SQL)

- Enables a DML, DDL, or logon trigger

- Syntax

```
ENABLE TRIGGER { [ schema_name . ] trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

- Examples

```
DISABLE TRIGGER Person.uAddress ON Person.Address;
  GO
  ENABLE Trigger Person.uAddress ON Person.Address;
  GO
```

DISABLE TRIGGER (Transact-SQL)

- Disables a trigger.
- Syntax

```
DISABLE TRIGGER { [ schema_name . ] trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

- Examples

```
IF EXISTS (SELECT * FROM sys.triggers
     WHERE parent_class = 0 AND name = 'safety')
  DROP TRIGGER safety ON DATABASE;
  GO
  CREATE TRIGGER safety
  ON DATABASE
  FOR DROP_TABLE, ALTER_TABLE
  AS
     PRINT 'You must disable Trigger "safety" to drop or alter tables!'
     ROLLBACK;
  GO
  DISABLE TRIGGER safety ON DATABASE;
  GO
```

```
DISABLE Trigger ALL ON ALL SERVER;
  GO
```

DDL Triggers

- DDL triggers fire in response to a variety of Data Definition Language (DDL) events.

- These events primarily correspond to Transact-SQL statements that start with the keywords CREATE, ALTER, DROP, GRANT, DENY, REVOKE or UPDATE STATISTICS.

- Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

- Use DDL triggers when you want to do the following:
  - Prevent certain changes to your database schema.
  - Have something occur in the database in response to a change in your database schema.
  - Record changes or events in the database schema.

CREATE DDL TRIGGER

- Syntax(DDL Trigger)

```
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 ]
```

CREATE DDL TRIGGER

- Example(DDL Trigger)

```
IF EXISTS (SELECT * FROM sys.server_triggers
      WHERE name = 'ddl_trig_database')
  DROP TRIGGER ddl_trig_database
  ON ALL SERVER;
  GO

CREATE TRIGGER ddl_trig_database
  ON ALL SERVER
  FOR CREATE_DATABASE
  AS
      PRINT 'Database Created.'
      SELECT  EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')
  GO

DROP TRIGGER ddl_trig_database
  ON ALL SERVER;
  GO
```

DDL Triggers

- DDL Events
  - The following partial list the DDL events that can be used to fire a DDL trigger or event notification

CREATE_FUNCTION	ALTER_FUNCTION	DROP_FUNCTION
CREATE_INDEX	ALTER_INDEX (Applies to the ALTER INDEX statement and **sp_indexoption**.)	DROP_INDEX
CREATE_MASTER_KEY	ALTER_MASTER_KEY	DROP_MASTER_KEY
CREATE_MESSAGE_TYPE	ALTER_MESSAGE_TYPE	DROP_MESSAGE_TYPE
CREATE_PARTITION_FUNCTION	ALTER_PARTITION_FUNCTION	DROP_PARTITION_FUNCTION
CREATE_PARTITION_SCHEME	ALTER_PARTITION_SCHEME	DROP_PARTITION_SCHEME

Trigger Functions (Transact-SQL)

- The following scalar functions can be used in the definition of a trigger to test for changes in data values or to return other data.

  COLUMNS_UPDATED() : Returns a varbinary bit pattern that indicates the columns in a table or view that were inserted or updated.

  EVENTDATA():Returns information about server or database events

  TRIGGER_NESTLEVEL(): Returns the number of triggers executed for the statement that fired the trigger

  UPDATE(): Returns a Boolean value that indicates whether an INSERT or UPDATE attempt was made on a specified column of a table or view

EVENTDATA Function

- Information about an event that fires a DDL trigger is captured by using the EVENTDATA function. This function returns an **xml** value. The XML schema includes information about the following:
  - The time of the event.
  - The System Process ID (SPID) of the connection when the trigger executed.
  - The type of event that fired the trigger.
  - Depending on the event type, the schema then includes additional information such as the database in which the event occurred, the object against which the event occurred, and the Transact-SQL statement of the event.

```
CREATE TRIGGER safety
ON DATABASE
FOR CREATE_TABLE
AS
    PRINT 'CREATE TABLE Issued.'
    SELECT EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')
   RAISERROR ('New tables cannot be created in this database.', 16, 1)
   ROLLBACK;
```

DDL Trigger (Example)

```
USE AdventureWorks2012;
GO
CREATE TABLE ddl_log (PostTime datetime, DB_User nvarchar(100), Event nvarchar(100), TSQL nvarchar(2000));
GO

CREATE TRIGGER log
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @data XML
SET @data = EVENTDATA()
INSERT ddl_log
   (PostTime, DB_User, Event, TSQL)
   VALUES
   (GETDATE(),
   CONVERT(nvarchar(100), CURRENT_USER),
   @data.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2000)') ) ;
GO


--Test the trigger
CREATE TABLE TestTable (a int)
DROP TABLE TestTable ;
GO
SELECT * FROM ddl_log ;
GO
```

Event Notifications

- Event notifications send information about events to a Service Broker service.

- Event notifications execute in response to a variety of Transact-SQL data definition language (DDL) statements and SQL Trace events by sending information about these events to a Service Broker service.

- Event notifications can be used to do the following:
  - Log and review changes or activity occurring on the database.
  - Perform an action in response to an event in an asynchronous instead of synchronous manner.
  - Event notifications can offer a programming alternative to DDL triggers and SQL Trace.

```
USE AdventureWorks2012;
GO
CREATE EVENT NOTIFICATION NotifyALTER_T1
ON DATABASE
FOR ALTER_TABLE
TO SERVICE '//Adventure-Works.com/ArchiveService' ,
    '8140a771-3c4b-4479-8ac0-81008ab17984';
```

Triggers vs Event Notifications

Triggers	Event Notifications
DML triggers respond to data manipulation language (DML) events. DDL triggers respond to data definition language (DDL) events.	Event notifications respond to DDL events and a subset of SQL trace events.
Triggers can run Transact-SQL or common language runtime (CLR) managed code.	Event notifications do not run code. Instead, they send **xml** messages to a Service Broker service.
Triggers are processed synchronously, within the scope of the transactions that cause them to fire.	Event notifications may be processed asynchronously and do not run in the scope of the transactions that cause them to fire.
The consumer of a trigger is tightly coupled with the event that causes it to fire.	The consumer of an event notification is decoupled from the event that causes it to fire.
Triggers must be processed on the local server.	Event notifications can be processed on a remote server.
Triggers can be rolled back.	Event notifications cannot be rolled back.
DML trigger names are schema-scoped. DDL trigger names are database-scoped or server-scoped.	Event notification names are scoped by the server or database. Event notifications on a QUEUE_ACTIVATION event are scoped to a specific queue.
DML triggers are owned by the same owner as the tables on which they are applied.	The owner of an event notification on a queue may have a different owner than the object on which it is applied.
Triggers support the EXECUTE AS clause.	Event notifications do not support the EXECUTE AS clause.
DDL trigger event information can be captured using the EVENTDATA function, which returns an **xml** data type.	Event notifications send **xml** event information to a Service Broker service. The information is formatted to the same schema as that of the EVENTDATA function.
Metadata about triggers is found in the **sys.triggers** and **sys.server_triggers** catalog views.	Metadata about event notifications is found in the **sys.event_notifications** and **sys.server_event_notifications** catalog views.

Logon Triggers

- Logon triggers fire stored procedures in response to a LOGON event.
- This event is raised when a user session is established with an instance of SQL Server.
- Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established.
- Logon triggers do not fire if authentication fails.
- logon triggers can be used to audit and control server sessions, such as by tracking login activity, restricting logins to SQL Server, or limiting the number of sessions for a specific login.

CREATE TRIGGER (Transact-SQL)

- Syntax(LOGON Trigger)

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR| AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >  [ ; ] }

<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

CREATE TRIGGER (Transact-SQL)

- Example(LOGON Trigger)

```
USE master;
  GO

CREATE LOGIN login_test3 WITH PASSWORD = '3KHJ6dhx(0xVYsdf' MUST_CHANGE,
      CHECK_EXPIRATION = ON;
  GO

GRANT VIEW SERVER STATE TO login_test3;
  GO


 CREATE TRIGGER connection_limit_trigger3
  ON ALL SERVER WITH EXECUTE AS 'login_test'
  FOR LOGON
  AS
  BEGIN
  IF ORIGINAL_LOGIN()= 'login_test3' AND
      (SELECT COUNT(*) FROM sys.dm_exec_sessions
              WHERE is_user_process = 1 AND
                  original_login_name = 'login_test3') > 3
      ROLLBACK;
  END;
```