Database System

Triggers

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.

DML Triggers

- DML triggers is a special type of stored procedure that automatically takes effect when a DML event takes place that affects the table or view defined in the trigger.
- DML events include INSERT, UPDATE, or DELETE statements.
- DML triggers can be used to enforce business rules and data integrity, query other tables, and include complex Transact-SQL statements.
- The trigger and the statement that fires it are treated as a single transaction, which can be rolled back from within the trigger.
- If a severe error is detected (for example, insufficient disk space), the entire transaction automatically rolls back.

DML Triggers

- DML triggers are similar to constraints in that they can enforce entity integrity or domain integrity.
- In general, entity integrity should always be enforced at the lowest level by indexes that are part of PRIMARY KEY and UNIQUE constraints or are created independently of constraints.
- Domain integrity should be enforced through CHECK constraints, and referential integrity (RI) should be enforced through FOREIGN KEY constraints.
- DML triggers are most useful when the features supported by constraints cannot meet the functional needs of the application.

DML Triggers – uses

- Some common uses of triggers include:
  - **Enforcing referential integrity**: Although it is recommended using *Declarative Referential Integrity* (*DRI*) whenever possible, there are many things that DRI won't do (for example, referential integrity across databases or even servers, many complex types of relationships, and so on).
  - **Creating audit trails**: This means writing out records that keep track of not just the most current data, but also the actual change history for each record.
  - **Creating functionality similar to a** CHECK **constraint**: Unlike CHECK constraints, this can work across tables, databases, or even servers.
  - **Substituting your own statements in the place of a user's action statement**: This is typically used to enable inserts in complex views.

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.

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.

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

CREATE TRIGGER (Transact-SQL)

- Creates a DML, DDL, or logon trigger in SQL Server
- Syntax(DML Trigger)

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ 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
```

CREATE TRIGGER (Transact-SQL)

- Creates a DML, DDL, or logon trigger in SQL Server
- Example(DML Trigger)

```
USE AdventureWorks2012;
  GO

IF OBJECT_ID ('Sales.reminder1', 'TR') IS NOT NULL
     DROP TRIGGER Sales.reminder1;
  GO

CREATE TRIGGER reminder1
  ON Sales.Customer
  AFTER INSERT, UPDATE
  AS RAISERROR ('Notify Customer Relations', 16, 10);
  GO
```

CREATE TRIGGER (Transact-SQL)

- Creates a DML, DDL, or logon trigger in SQL Server
- Example(DML Trigger)

```sql
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
```

CREATE TRIGGER (Transact-SQL)

- 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 TRIGGER (Transact-SQL)

- 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
```

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;
```

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
```