CREATE TRIGGER (Transact-SQL)

Updated: April 6, 2016

THIS TOPIC APPLIES TO: ✓ SQL Server (starting with 2008) ✓ Azure SQL Database ✗ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

Creates a DML, DDL, or logon trigger. A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. These triggers fire when any valid event is fired, regardless of whether or not any table rows are affected. For more information, see DML Triggers.

DDL triggers execute in response to a variety of data definition language (DDL) events. These events primarily correspond to Transact-SQL CREATE, ALTER, and DROP statements, and certain system stored procedures that perform DDL-like operations. Logon triggers fire in response to the LOGON event that is raised when a user sessions is being established. Triggers can be created directly from Transact-SQL statements or from methods of assemblies that are created in the Microsoft .NET Framework common language runtime (CLR) and uploaded to an instance of SQL Server. SQL Server allows for creating multiple triggers for any specific statement.

Important

Malicious code inside triggers can run under escalated privileges. For more information on how to mitigate this threat, see Manage Trigger Security.

Note

The integration of .NET Framework CLR into SQL Server is discussed in this topic. CLR integration does not apply to Azure SQL Database.

Transact-SQL Syntax Conventions

Syntax

```
-- SQL Server Syntax
Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

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

```
-- SQL Server Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to a
-- table (DML Trigger on memory-optimized tables)

CREATE TRIGGER [ schema_name . ]trigger_name
ON { table }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement  [ ; ] [ ,...n ] }

<dml_trigger_option> ::=
    [ NATIVE_COMPILATION ]
    [ SCHEMABINDING ]
    [ EXECUTE AS Clause ]
```

```
-- Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE,
--  or UPDATE STATISTICS statement (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 ]
```

```
-- Trigger on a LOGON event (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 ]
```

Syntax

```
-- Windows Azure SQL Database Syntax
-- Trigger on an INSERT, UPDATE, or DELETE statement to
-- a table or view (DML Trigger)

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

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

```
-- Windows Azure SQL Database Syntax
Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or UPDATE STATISTICS statement
(DDL Trigger)

CREATE TRIGGER trigger_name
ON { DATABASE }
 [ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ]  [ ; ] }

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

Arguments

schema_name
Is the name of the schema to which a DML trigger belongs. DML triggers are scoped to the schema of the table or view on which they are created. *schema_name* cannot be specified for DDL or logon triggers.

trigger_name
Is the name of the trigger. A *trigger_name* must comply with the rules for identifiers, except that *trigger_name* cannot start with # or ##.

table | *view*
Is the table or view on which the DML trigger is executed and is sometimes referred to as the trigger table or trigger view. Specifying the fully qualified name of the table or view is optional. A view can be referenced only by an INSTEAD OF trigger. DML triggers cannot be defined on local or global temporary tables.

DATABASE
Applies the scope of a DDL trigger to the current database. If specified, the trigger fires whenever *event_type* or *event_group* occurs in the current database.

ALL SERVER

Applies to: SQL Server 2008 through SQL Server 2016.

Applies the scope of a DDL or logon trigger to the current server. If specified, the trigger fires whenever *event_type* or *event_group* occurs anywhere in the current server.

WITH ENCRYPTION

Applies to: SQL Server 2008 through SQL Server 2016.

Obfuscates the text of the CREATE TRIGGER statement. Using WITH ENCRYPTION prevents the trigger from being published as part of SQL Server replication. WITH ENCRYPTION cannot be specified for CLR triggers.

EXECUTE AS
Specifies the security context under which the trigger is executed. Enables you to control which user account the instance of SQL Server uses to validate permissions on any database objects that are referenced by the trigger.

This option is required for triggers on memory-optimized tables.

For more information, seeEXECUTE AS Clause (Transact-SQL).

NATIVE_COMPILATION
Indicates that the trigger is natively compiled.

This option is required for triggers on memory-optimized tables.

SCHEMABINDING
Ensures that tables that are referenced by a trigger cannot be dropped or altered.

This option is required for triggers on memory-optimized tables and is not supported for triggers on traditional tables.

FOR | AFTER
AFTER specifies that the DML trigger is fired only when all operations specified in the triggering SQL statement have executed successfully. All referential cascade actions and constraint checks also must succeed before this trigger fires.

AFTER is the default when FOR is the only keyword specified.

AFTER triggers cannot be defined on views.

INSTEAD OF
Specifies that the DML trigger is executed *instead of* the triggering SQL statement, therefore, overriding the actions of the triggering statements. INSTEAD OF cannot be specified for DDL or logon triggers.

At most, one INSTEAD OF trigger per INSERT, UPDATE, or DELETE statement can be defined on a table or view. However, you can define views on views where each view has its own INSTEAD OF trigger.

INSTEAD OF triggers are not allowed on updatable views that use WITH CHECK OPTION. SQL Server raises an error when an INSTEAD OF trigger is added to an updatable view WITH CHECK OPTION specified. The user must remove that option by using ALTER VIEW before defining the INSTEAD OF trigger.

{ [DELETE] [,] [INSERT] [,] [UPDATE] }
Specifies the data modification statements that activate the DML trigger when it is tried against this table or view. At least one option must be specified. Any combination of these options in any order is allowed in the trigger definition.

For INSTEAD OF triggers, the DELETE option is not allowed on tables that have a referential relationship specifying a cascade action ON DELETE. Similarly, the UPDATE option is not allowed on tables that have a referential relationship specifying a cascade action ON UPDATE.

WITH APPEND

Applies to: SQL Server 2008 through SQL Server 2008 R2.

Specifies that an additional trigger of an existing type should be added. WITH APPEND cannot be used with INSTEAD OF triggers or if AFTER trigger is explicitly stated. WITH APPEND can be used only when FOR is specified, without INSTEAD OF or AFTER, for backward compatibility reasons. WITH APPEND cannot be specified if EXTERNAL NAME is specified (that is, if the trigger is a CLR trigger).

event_type
Is the name of a Transact-SQL language event that, after execution, causes a DDL trigger to fire. Valid events for DDL triggers are listed in DDL Events.

event_group
Is the name of a predefined grouping of Transact-SQL language events. The DDL trigger fires after execution of any Transact-SQL language event that belongs to *event_group*. Valid event groups for DDL triggers are listed in DDL Event Groups.

After the CREATE TRIGGER has finished running, *event_group* also acts as a macro by adding the event types it covers to the sys.trigger_events catalog view.

NOT FOR REPLICATION

Applies to: SQL Server 2008 through SQL Server 2016.

Indicates that the trigger should not be executed when a replication agent modifies the table that is involved in the trigger.

sql_statement
Is the trigger conditions and actions. Trigger conditions specify additional criteria that determine whether the tried DML, DDL, or logon events cause the trigger actions to be performed.

The trigger actions specified in the Transact-SQL statements go into effect when the operation is tried.

Triggers can include any number and kind of Transact-SQL statements, with exceptions. For more information, see Remarks. A trigger is designed to check or change data based on a data modification or definition statement; it should not return data to the user. The Transact-SQL statements in a trigger frequently include control-of-flow language.

DML triggers use the deleted and inserted logical (conceptual) tables. They are structurally similar to the table on which the trigger is defined, that is, the table on which the user action is tried. The deleted and inserted tables hold the old values or new values of the rows that may be changed by the user action. For example, to retrieve all values in the `deleted` table, use:

```
SELECT * FROM deleted;
```

For more information, see Use the inserted and deleted Tables.

DDL and logon triggers capture information about the triggering event by using the EVENTDATA (Transact-SQL) function. For more information, see Use the EVENTDATA Function.

SQL Server allows for the update of **text**, **ntext**, or **image** columns through the INSTEAD OF trigger on tables or views.

Important

ntext, **text**, and **image** data types will be removed in a future version of MicrosoftSQL Server. Avoid using these data types in new development work, and plan to modify applications that currently use them. Use nvarchar(max), varchar(max), and varbinary(max) instead. Both AFTER and INSTEAD OF triggers support **varchar(MAX)**, **nvarchar(MAX)**, and **varbinary(MAX)** data in the inserted and deleted tables.

For triggers on memory-optimized tables, the only *sql_statement* allowed at the top level is an ATOMIC block. The T-SQL allowed inside the ATOMIC block is limited by the T-SQL allowed inside native procs.

< method_specifier >

Applies to: SQL Server 2008 through SQL Server 2016.

For a CLR trigger, specifies the method of an assembly to bind with the trigger. The method must take no arguments and return void. *class_name* must be a valid SQL Server identifier and must exist as a class in the assembly with assembly visibility. If the class has a namespace-qualified name that uses '.' to separate namespace parts, the class name must be delimited by using [ ] or " " delimiters. The class cannot be a nested class.

Note

By default, the ability of SQL Server to run CLR code is off. You can create, modify, and drop database objects that reference managed code modules, but these references will not execute in an instance of SQL Server unless the clr enabled Option is enabled by using sp_configure.

Remarks

DML Triggers

DML triggers are frequently used for enforcing business rules and data integrity. SQL Server provides declarative referential integrity (DRI) through the ALTER TABLE and CREATE TABLE statements. However, DRI does not provide cross-database referential integrity. Referential integrity refers to the rules about the relationships between the primary and foreign keys of tables. To enforce referential integrity, use the PRIMARY KEY and FOREIGN KEY constraints in ALTER TABLE and CREATE TABLE. If constraints exist on the trigger table, they are checked after the INSTEAD OF trigger execution and before the AFTER trigger execution. If the constraints are violated, the INSTEAD OF trigger actions are rolled back and the AFTER trigger is not fired.

The first and last AFTER triggers to be executed on a table can be specified by using sp_settriggerorder. Only one first and one last AFTER trigger for each INSERT, UPDATE, and DELETE operation can be specified on a table. If there are other AFTER triggers on the same table, they are randomly executed.

If an ALTER TRIGGER statement changes a first or last trigger, the first or last attribute set on the modified trigger is dropped, and the order value must be reset by using sp_settriggerorder.

An AFTER trigger is executed only after the triggering SQL statement has executed successfully. This successful execution includes all referential cascade actions and constraint checks associated with the object updated or deleted. An AFTER trigger will not recursively fire an INSTEAD OF trigger on the same table.

If an INSTEAD OF trigger defined on a table executes a statement against the table that would ordinarily fire the INSTEAD OF trigger again, the trigger is not called recursively. Instead, the statement is processed as if the table had no INSTEAD OF trigger and starts the chain of constraint operations and AFTER trigger executions. For example, if a trigger is defined as an INSTEAD OF INSERT trigger for a table, and the trigger executes an INSERT statement on the same table, the INSERT statement executed by the INSTEAD OF trigger does not call the trigger again. The INSERT executed by the trigger starts the process of performing constraint actions and firing any AFTER INSERT triggers defined for the table.

If an INSTEAD OF trigger defined on a view executes a statement against the view that would ordinarily fire the INSTEAD OF trigger again, it is not called recursively. Instead, the statement is resolved as modifications against the base tables underlying the view. In this case, the view definition must meet all the restrictions for an updatable view. For a definition of updatable views, see Modify Data Through a View.

For example, if a trigger is defined as an INSTEAD OF UPDATE trigger for a view, and the trigger executes an UPDATE statement referencing the same view, the UPDATE statement executed by the INSTEAD OF trigger does not call the trigger again. The UPDATE executed by the trigger is processed against the view as if the view did not have an INSTEAD OF trigger. The columns changed by the UPDATE must be resolved to a single base table. Each modification to an underlying base table starts the chain of applying constraints and firing AFTER triggers defined for the table.

Testing for UPDATE or INSERT Actions to Specific Columns

You can design a Transact-SQL trigger to perform certain actions based on UPDATE or INSERT modifications to specific columns. Use UPDATE() or COLUMNS_UPDATED in the body of the trigger for this purpose. UPDATE() tests for UPDATE or INSERT tries on one column. COLUMNS_UPDATED tests for UPDATE or INSERT actions that are performed on multiple columns and returns a bit pattern that indicates which columns were inserted or updated.

Trigger Limitations

CREATE TRIGGER must be the first statement in the batch and can apply to only one table.

A trigger is created only in the current database; however, a trigger can reference objects outside the current database.

If the trigger schema name is specified to qualify the trigger, qualify the table name in the same way.

The same trigger action can be defined for more than one user action (for example, INSERT and UPDATE) in the same CREATE TRIGGER statement.

INSTEAD OF DELETE/UPDATE triggers cannot be defined on a table that has a foreign key with a cascade on DELETE/UPDATE action defined.

Any SET statement can be specified inside a trigger. The SET option selected remains in effect during the execution of the trigger and then reverts to its former setting.

When a trigger fires, results are returned to the calling application, just like with stored procedures. To prevent having results returned to an application because of a trigger firing, do not include either SELECT statements that return results or statements that perform variable assignment in a trigger. A trigger that includes either SELECT statements that return results to the user or statements that perform variable assignment requires special handling; these returned results would have to be written into every application in which modifications to the trigger table are allowed. If variable assignment must occur in a trigger, use a SET NOCOUNT statement at the start of the trigger to prevent the return of any result sets.

Although a TRUNCATE TABLE statement is in effect a DELETE statement, it does not activate a trigger because the operation does not log individual row deletions. However, only those users with permissions to execute a TRUNCATE TABLE statement need be concerned about inadvertently circumventing a DELETE trigger this way.

The WRITETEXT statement, whether logged or unlogged, does not activate a trigger.

The following Transact-SQL statements are not allowed in a DML trigger:

ALTER DATABASE	CREATE DATABASE	DROP DATABASE
RESTORE DATABASE	RESTORE LOG	RECONFIGURE

Additionally, the following Transact-SQL statements are not allowed inside the body of a DML trigger when it is used against the table or view that is the target of the triggering action.

CREATE INDEX (including CREATE SPATIAL INDEX and CREATE XML INDEX)	ALTER INDEX	DROP INDEX
DBCC DBREINDEX	ALTER PARTITION FUNCTION	DROP TABLE
ALTER TABLE when used to do the following: Add, modify, or drop columns. Switch partitions. Add or drop PRIMARY KEY or UNIQUE constraints.		

Note

Because SQL Server does not support user-defined triggers on system tables, we recommend that you do not create user-defined triggers on system tables.

DDL Triggers

DDL triggers, like standard triggers, execute stored procedures in response to an event. But unlike standard triggers, they do not execute in response to UPDATE, INSERT, or DELETE statements on a table or view. Instead, they primarily execute in response to data definition language (DDL) statements. These include CREATE, ALTER, DROP, GRANT, DENY, REVOKE, and UPDATE STATISTICS statements. Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

Important

Test your DDL triggers to determine their responses to system stored procedure execution. For example, the CREATE TYPE statement and the sp_addtype and sp_rename stored procedures will fire a DDL trigger that is created on a CREATE_TYPE event.

For more information about DDL triggers, see DDL Triggers.

DDL triggers do not fire in response to events that affect local or global temporary tables and stored procedures.

Unlike DML triggers, DDL triggers are not scoped to schemas. Therefore, functions such as OBJECT_ID, OBJECT_NAME, OBJECTPROPERTY, and OBJECTPROPERTYEX cannot be used for querying metadata about DDL triggers. Use the catalog views instead. For more information, see Get Information About DDL Triggers.

Note

Server-scoped DDL triggers appear in the SQL Server Management Studio Object Explorer in the **Triggers** folder. This folder is located under the **Server Objects** folder. Database-scoped DDL Triggers appear in the **Database Triggers** folder. This folder is located under the **Programmability** folder of the corresponding database.

Logon Triggers

Logon triggers execute 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. Therefore, all messages originating inside the trigger that would typically reach the user, such as error messages and messages from the PRINT statement, are diverted to the SQL Server error log. For more information, see Logon Triggers.

Logon triggers do not fire if authentication fails.

Distributed transactions are not supported in a logon trigger. Error 3969 is returned when a logon trigger containing a distributed transaction is fired.

Disabling a Logon Trigger

A logon trigger can effectively prevent successful connections to the Database Engine for all users, including members of the **sysadmin** fixed server role. When a logon trigger is preventing connections, members of the **sysadmin** fixed server role can connect by using the dedicated administrator connection, or by starting the Database Engine in minimal configuration mode (-f). For more information, see Database Engine Service Startup Options.

General Trigger Considerations

Returning Results

The ability to return results from triggers will be removed in a future version of SQL Server. Triggers that return result sets may cause unexpected behavior in applications that are not designed to work with them. Avoid returning result sets from triggers in new development work, and plan to modify applications that currently do this. To prevent triggers from returning result sets, set the disallow results from triggers option to 1.

Logon triggers always disallow results sets to be returned and this behavior is not configurable. If a logon trigger does generate a result set, the trigger fails to execute and the login attempt that fired the trigger is denied.

Multiple Triggers

SQL Server allows for multiple triggers to be created for each DML, DDL, or LOGON event. For example, if CREATE TRIGGER FOR UPDATE is executed for a table that already has an UPDATE trigger, an additional update trigger is created. In earlier versions of SQL Server, only one trigger for each INSERT, UPDATE, or DELETE data modification event is allowed for each table.

Recursive Triggers

SQL Server also allows for recursive invocation of triggers when the RECURSIVE_TRIGGERS setting is enabled using ALTER DATABASE.

Recursive triggers enable the following types of recursion to occur:

- Indirect recursion

  With indirect recursion, an application updates table T1. This fires trigger TR1, updating table T2. In this scenario, trigger T2 then fires and updates table T1.

- Direct recursion

  With direct recursion, the application updates table T1. This fires trigger TR1, updating table T1. Because table T1 was updated, trigger TR1 fires again, and so on.

The following example uses both indirect and direct trigger recursion Assume that two update triggers, TR1 and TR2, are defined on table T1. Trigger TR1 updates table T1 recursively. An UPDATE statement executes each TR1 and TR2 one time. Additionally, the execution of TR1 triggers the execution of TR1 (recursively) and TR2. The inserted and deleted tables for a specific trigger contain rows that correspond only to the UPDATE statement that invoked the trigger.

Note

The previous behavior occurs only if the RECURSIVE_TRIGGERS setting is enabled by using ALTER DATABASE. There is no defined order in which multiple triggers defined for a specific event are executed. Each trigger should be self-contained.

Disabling the RECURSIVE_TRIGGERS setting only prevents direct recursions. To disable indirect recursion also, set the nested triggers server option to 0 by using sp_configure.

If any one of the triggers performs a ROLLBACK TRANSACTION, regardless of the nesting level, no more triggers are executed.

Nested Triggers

Triggers can be nested to a maximum of 32 levels. If a trigger changes a table on which there is another trigger, the second trigger is activated and can then call a third trigger, and so on. If any trigger in the chain sets off an infinite loop, the nesting level is exceeded and the trigger is canceled. When a Transact-SQL trigger executes managed code by referencing a CLR routine, type, or aggregate, this reference counts as one level against the 32-level nesting limit. Methods invoked from within managed code do not count against this limit

To disable nested triggers, set the nested triggers option of sp_configure to 0 (off). The default configuration allows for nested triggers. If nested triggers is off, recursive triggers is also disabled, regardless of the RECURSIVE_TRIGGERS setting set by using ALTER DATABASE.

The first AFTER trigger nested inside an INSTEAD OF trigger fires even if the **nested triggers** server configuration option is set to 0. However, under this setting, later AFTER triggers do not fire. We recommend that you review your applications for nested triggers to determine whether the applications comply with your business rules with regard to this behavior when the **nested triggers** server configuration option is set to 0, and then make appropriate modifications.

Deferred Name Resolution

SQL Server allows for Transact-SQL stored procedures, triggers, and batches to refer to tables that do not exist at compile time. This ability is called deferred name resolution.

Permissions

To create a DML trigger requires ALTER permission on the table or view on which the trigger is being created.

To create a DDL trigger with server scope (ON ALL SERVER) or a logon trigger requires CONTROL SERVER permission on the server. To create a DDL trigger with database scope (ON DATABASE) requires ALTER ANY DATABASE DDL TRIGGER permission in the current database.

Examples

A. Using a DML trigger with a reminder message

The following DML trigger prints a message to the client when anyone tries to add or change data in the `Customer` table in the AdventureWorks2012 database.

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

B. Using a DML trigger with a reminder e-mail message

The following example sends an e-mail message to a specified person (`MaryM`) when the `Customer` table changes.

```
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
```

C. Using a DML AFTER trigger to enforce a business rule between the PurchaseOrderHeader and Vendor tables

Because CHECK constraints can reference only the columns on which the column-level or table-level constraint is defined, any cross-table constraints (in this case, business rules) must be defined as triggers.

The following example creates a DML trigger in the AdventureWorks2012 database. This trigger checks to make sure the credit rating for the vendor is good (not 5) when an attempt is made to insert a new purchase order into the `PurchaseOrderHeader` table. To obtain the credit rating of the vendor, the `Vendor` table must be referenced. If the credit rating is too low, a message is displayed and the insertion does not execute.

```
-- This trigger prevents a row from being inserted in the
Purchasing.PurchaseOrderHeader table
-- when the credit rating of the specified vendor is set to 5 (below average).

CREATE TRIGGER Purchasing.LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT
AS
IF EXISTS (SELECT *
           FROM Purchasing.PurchaseOrderHeader AS p
           JOIN inserted AS i
           ON p.PurchaseOrderID = i.PurchaseOrderID
           JOIN Purchasing.Vendor AS v
           ON v.BusinessEntityID = p.VendorID
           WHERE v.CreditRating = 5
          )
BEGIN
RAISERROR ('A vendor''s credit rating is too low to accept new
purchase orders.', 16, 1);
ROLLBACK TRANSACTION;
RETURN
END;
GO

-- This statement attempts to insert a row into the PurchaseOrderHeader table
-- for a vendor that has a below average credit rating.
-- The AFTER INSERT trigger is fired and the INSERT transaction is rolled back.

INSERT INTO Purchasing.PurchaseOrderHeader (RevisionNumber, Status, EmployeeID,
VendorID, ShipMethodID, OrderDate, ShipDate, SubTotal, TaxAmt, Freight)
VALUES (
2
,3
,261
,1652
,4
,GETDATE()
,GETDATE()
,44594.55
,3567.564
,1114.8638 );
GO
```

D. Using a database-scoped DDL trigger

The following example uses a DDL trigger to prevent any synonym in a database from being dropped.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_SYNONYM
AS
   RAISERROR ('You must disable Trigger "safety" to drop synonyms!',10, 1)
   ROLLBACK
GO
DROP TRIGGER safety
ON DATABASE;
GO
```

E. Using a server-scoped DDL trigger

The following example uses a DDL trigger to print a message if any CREATE DATABASE event occurs on the current server instance, and uses the `EVENTDATA` function to retrieve the text of the corresponding Transact-SQL statement. For more examples that use EVENTDATA in DDL triggers, see Use the EVENTDATA Function.

Applies to: SQL Server 2008 through SQL Server 2016.

```
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
```

F. Using a logon trigger

The following logon trigger example denies an attempt to log in to SQL Server as a member of the *login_test* login if there are already three user sessions running under that login.

Applies to: SQL Server 2008 through SQL Server 2016.

```
USE master;
GO
```

```
CREATE LOGIN login_test WITH PASSWORD = '3KHJ6dhx(0xVYsdf' MUST_CHANGE,
    CHECK_EXPIRATION = ON;
GO
GRANT VIEW SERVER STATE TO login_test;
GO
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'login_test'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'login_test' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'login_test') > 3
    ROLLBACK;
END;
```

G. Viewing the events that cause a trigger to fire

The following example queries the `sys.triggers` and `sys.trigger_events` catalog views to determine which Transact-SQL language events cause trigger `safety` to fire. `safety` is created in the previous example.

```
SELECT TE.*
FROM sys.trigger_events AS TE
JOIN sys.triggers AS T ON T.object_id = TE.object_id
WHERE T.parent_class = 0 AND T.name = 'safety';
GO
```

See Also

ALTER TABLE (Transact-SQL)
ALTER TRIGGER (Transact-SQL)
COLUMNS_UPDATED (Transact-SQL)
CREATE TABLE (Transact-SQL)
DROP TRIGGER (Transact-SQL)
ENABLE TRIGGER (Transact-SQL)
DISABLE TRIGGER (Transact-SQL)
TRIGGER_NESTLEVEL (Transact-SQL)
EVENTDATA (Transact-SQL)
sys.dm_sql_referenced_entities (Transact-SQL)
sys.dm_sql_referencing_entities (Transact-SQL)
sys.sql_expression_dependencies (Transact-SQL)
sp_help (Transact-SQL)
sp_helptrigger (Transact-SQL)
sp_helptext (Transact-SQL)
sp_rename (Transact-SQL)
sp_settriggerorder (Transact-SQL)
UPDATE() (Transact-SQL)
Get Information About DML Triggers
Get Information About DDL Triggers
sys.triggers (Transact-SQL)

sys.trigger_events (Transact-SQL)
sys.sql_modules (Transact-SQL)
sys.assembly_modules (Transact-SQL)
sys.server_triggers (Transact-SQL)
sys.server_trigger_events (Transact-SQL)
sys.server_sql_modules (Transact-SQL)
sys.server_assembly_modules (Transact-SQL)

Community Additions