**TRIGGER**

**CREATE TRIGGER (Transact-SQL)**

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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| These triggers fire when any valid event is fired, regardless of whether or not any table rows are affected. |

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.

|  |
| --- |
| **ms189799.security(en-us,SQL.100).gifSecurity Note:** |
| Malicious code inside triggers can run under escalated privileges. For more information on how to mitigate this threat, see [Managing Trigger Security](http://msdn.microsoft.com/en-us/library/ms191134.aspx). |

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif 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

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 ]

<method\_specifier> ::=

assembly\_name.class\_name.method\_name

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 ]

<method\_specifier> ::=

assembly\_name.class\_name.method\_name

http://i.msdn.microsoft.com/Global/Images/clear.gif 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](http://msdn.microsoft.com/en-us/library/ms175874.aspx), 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 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

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.

For more information, see[EXECUTE AS Clause (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188354.aspx).

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.

*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](http://msdn.microsoft.com/en-us/library/bb522542.aspx).

*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](http://msdn.microsoft.com/en-us/library/bb510452.aspx).

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.

WITH APPEND

Specifies that an additional trigger of an existing type should be added. Use of this optional clause is required only when the compatibility level is 65 or lower. If the compatibility level is 70 or higher, the WITH APPEND clause is not required to add an additional trigger of an existing type. This is the default behavior of CREATE TRIGGER with the compatibility level setting of 70 or higher. For more information, see [sp\_dbcmptlevel (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms178653.aspx).

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).

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifImportant:** |
| WITH APPEND will be removed in the next version of Microsoft SQL Server. Do not use WITH APPEND in new development work, and plan to modify applications that currently use it. |

NOT FOR REPLICATION

Indicates that the trigger should not be executed when a replication agent modifies the table that is involved in the trigger. For more information, see [Controlling Constraints, Identities, and Triggers with NOT FOR REPLICATION](http://msdn.microsoft.com/en-us/library/ms152529.aspx).

*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](http://msdn.microsoft.com/en-us/library/ms174290.aspx).

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:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl39other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl39other');)

SELECT \*

FROM deleted

For more information, see [Using the inserted and deleted Tables](http://msdn.microsoft.com/en-us/library/ms191300.aspx).

DDL and logon triggers capture information about the triggering event by using the [EVENTDATA (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms173781.aspx) function. For more information, see [Using the EVENTDATA Function](http://msdn.microsoft.com/en-us/library/ms187909.aspx).

In a DELETE, INSERT, or UPDATE trigger, SQL Server does not allow **text**, **ntext**, or **image** column references in the **inserted** and **deleted** tables if the compatibility level is set to 70. The **text**, **ntext**, and **image** values in the **inserted** and **deleted** tables cannot be accessed. To retrieve the new value in either an INSERT or UPDATE trigger, join the **inserted** table with the original update table. When the compatibility level is 65 or lower, null values are returned for **inserted** or **deleted** **text**, **ntext**, or **image** columns that allow null values; zero-length strings are returned if the columns are not nullable.

If the compatibility level is 80 or higher, SQL Server allows for the update of **text**, **ntext**, or **image** columns through the INSTEAD OF trigger on tables or views.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifImportant:** |
| **ntext**, **text**, and **image** data types will be removed in a future version of Microsoft SQL Server. Avoid using these data types in new development work, and plan to modify applications that currently use them. Use [nvarchar(max)](http://msdn.microsoft.com/en-us/library/ms186939.aspx), [varchar(max)](http://msdn.microsoft.com/en-us/library/ms176089.aspx), and [varbinary(max)](http://msdn.microsoft.com/en-us/library/ms188362.aspx) instead. Both AFTER and INSTEAD OF triggers support **varchar(MAX)**, **nvarchar(MAX)**, and **varbinary(MAX)** data in the **inserted** and **deleted** tables. |

< method\_specifier >

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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| 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](http://msdn.microsoft.com/en-us/library/ms175193.aspx) is enabled by using [sp\_configure](http://msdn.microsoft.com/en-us/library/ms188787.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif 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.

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 [Modifying Data Through a View](http://msdn.microsoft.com/en-us/library/ms180800.aspx).

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()](http://msdn.microsoft.com/en-us/library/ms187326.aspx) or [COLUMNS\_UPDATED](http://msdn.microsoft.com/en-us/library/ms186329.aspx) 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 |
| LOAD DATABASE | LOAD LOG | RECONFIGURE |
| RESTORE DATABASE | RESTORE LOG |  |

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. |  |  |

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| 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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifImportant:** |
| Test your DDL triggers to determine their responses to system stored procedure execution. For example, the CREATE TYPE statement and the **sp\_addtype** stored procedure will both fire a DDL trigger that is created on a CREATE\_TYPE event. However, the **sp\_rename** stored procedure does not fire any DDL triggers. |

For more information about DDL triggers, see [DDL Triggers](http://msdn.microsoft.com/en-us/library/ms190989.aspx).

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, the 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 [Getting Information About DDL Triggers](http://msdn.microsoft.com/en-us/library/ms184304.aspx).

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| 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 fire in response to the LOGON event. This event is raised when a user sessions is being established. For more information, see [Logon Triggers](http://msdn.microsoft.com/en-us/library/bb326598.aspx).

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](http://msdn.microsoft.com/en-us/library/ms186337.aspx) 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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| With the compatibility level of 70, the default behavior for CREATE TRIGGER is to add additional triggers to existing triggers if the trigger names differ. If trigger names are the same, SQL Server returns an error message. However, if the compatibility level is equal to or less than 65, any new triggers created by using the CREATE TRIGGER statement replace any existing triggers of the same type, even if the trigger names are different. For more information, see [sp\_dbcmptlevel (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms178653.aspx). |

**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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| 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. 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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| 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. |

**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. However, if the Transact-SQL stored procedure, trigger, or batch refers to a table that is defined in the stored procedure or trigger, a warning is issued at creation time only if the compatibility level setting is 65. A warning is issued at compile time if a batch is used. An error message is returned at run time when the table referenced does not exist. For more information, see [Deferred Name Resolution and Compilation](http://msdn.microsoft.com/en-us/library/ms190686.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif 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.

http://i.msdn.microsoft.com/Global/Images/clear.gif 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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl99other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl99other');)

USE AdventureWorks;

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

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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl100other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl100other');)

USE AdventureWorks;

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 = 'AdventureWorks 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. This trigger checks to make sure the credit rating for the vendor is good 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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| To view examples of DML AFTER triggers that update multiple rows, see [Multirow Considerations for DML Triggers](http://msdn.microsoft.com/en-us/library/ms190752.aspx). To view examples of DML INSTEAD OF INSERT triggers, see [INSTEAD OF INSERT Triggers](http://msdn.microsoft.com/en-us/library/ms175089.aspx). |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl104other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl104other');)

IF OBJECT\_ID ('Purchasing.LowCredit','TR') IS NOT NULL

DROP TRIGGER Purchasing.LowCredit;

GO

CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader

AFTER INSERT

AS

DECLARE @creditrating tinyint,

@vendorid int

SELECT @creditrating = v.CreditRating, @vendorid = p.VendorID

FROM Purchasing.PurchaseOrderHeader AS p

INNER JOIN inserted AS i ON p.PurchaseOrderID =

i.PurchaseOrderID

JOIN Purchasing.Vendor AS v on v.VendorID = i.VendorID

IF @creditrating = 5

BEGIN

RAISERROR ('This vendor''s credit rating is too low to accept new

purchase orders.', 16, 1)

ROLLBACK TRANSACTION

END

GO

D. Using deferred name resolution

The following example creates two DML triggers to illustrate deferred name resolution.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl105other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl105other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ('HumanResources.trig1','TR') IS NOT NULL

DROP TRIGGER HumanResources.trig1;

GO

-- Creating a trigger on a nonexistent table.

CREATE TRIGGER HumanResources.trig1

on HumanResources.Employee

AFTER INSERT, UPDATE, DELETE

AS

SELECT e.EmployeeID, e.BirthDate, x.info

FROM HumanResources.Employee AS e INNER JOIN does\_not\_exist AS x

ON e.EmployeeID = x.xID

GO

-- Here is the statement to actually see the text of the trigger.

SELECT t.object\_id, m.definition

FROM sys.triggers AS t INNER JOIN sys.sql\_modules AS m

ON t.object\_id = m.object\_id

WHERE t.type = 'TR' and t.name = 'trig1'

AND t.parent\_class = 1

GO

-- Creating a trigger on an existing table, but with a nonexistent

-- column.

USE AdventureWorks;

GO

IF OBJECT\_ID ('HumanResources.trig2','TR') IS NOT NULL

DROP TRIGGER HumanResources.trig2

GO

CREATE TRIGGER HumanResources.trig2

ON HumanResources.Employee

AFTER INSERT, UPDATE

AS

DECLARE @fax varchar(12)

SELECT @fax = 'AltPhone'

FROM HumanResources.Employee

GO

-- Here is the statement to actually see the text of the trigger.

SELECT t.object\_id, m.definition

FROM sys.triggers AS t INNER JOIN sys.sql\_modules AS m

ON t.object\_id = m.object\_id

WHERE t.type = 'TR' and t.name = 'trig2'

AND t.parent\_class = 1

GO

E. Using a database-scoped DDL trigger

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

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl106other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl106other');)

USE AdventureWorks;

GO

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\_SYNONYM

AS

RAISERROR ('You must disable Trigger "safety" to drop synonyms!',10, 1)

ROLLBACK

GO

DROP TRIGGER safety

ON DATABASE;

GO

F. 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.

|  |
| --- |
| **ms189799.note(en-us,SQL.100).gifNote:** |
| For more examples that use EVENTDATA in DDL triggers, see [Using the EVENTDATA Function](http://msdn.microsoft.com/en-us/library/ms187909.aspx). |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl109other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl109other');)

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

G. 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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl110other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl110other');)

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;

H. 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.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl111other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl111other');)

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

**Understanding DDL Triggers vs. DML Triggers**

DDL triggers and DML triggers are used for different purposes.

DML triggers operate on INSERT, UPDATE, and DELETE statements, and help to enforce business rules and extend data integrity when data is modified in tables or views.

DDL triggers operate on CREATE, ALTER, DROP, and other DDL statements and stored procedures that perform DDL-like operations. They are used to perform administrative tasks and enforce business rules that affect databases. They apply to all commands of a single type across a database, or across a server.

DML triggers and DDL triggers are created, modified, and dropped by using similar Transact-SQL syntax, and share other similar behavior.

Like DML triggers, DDL triggers can run managed code packaged in an assembly that was created in the Microsoft .NET Framework and uploaded in SQL Server. For more information, see [Programming CLR Triggers](http://msdn.microsoft.com/en-us/library/ms179562.aspx).

Like DML triggers, more than one DDL trigger can be created on the same Transact-SQL statement. Also, a DDL trigger and the statement that fires it are run within the same transaction. This transaction can be rolled back from within the trigger. Serious errors can cause a whole transaction to be automatically rolled back. DDL triggers that are run from a batch and explicitly include the ROLLBACK TRANSACTION statement will cancel the whole batch. For more information, see [Using DML Triggers That Include COMMIT or ROLLBACK TRANSACTION](http://msdn.microsoft.com/en-us/library/ms190974.aspx).

|  |
| --- |
| **ms189599.note(en-us,SQL.100).gifNote:** |
| An ALTER DATABASE event that occurs inside the body of a DDL trigger cannot be rolled back. |

Like DML triggers, DDL triggers can be nested. For more information, see [Using Nested Triggers](http://msdn.microsoft.com/en-us/library/ms190739.aspx).

When you are designing DDL triggers, consider how they differ from DML triggers in the following ways:

* DDL triggers run only after a Transact-SQL statement is completed. DDL triggers cannot be used as INSTEAD OF triggers.
* DDL triggers do not create the **inserted** and **deleted** tables. The information about an event that fires a DDL trigger, and the subsequent changes caused by the trigger, is captured by using the [EVENTDATA](http://msdn.microsoft.com/en-us/library/ms173781.aspx) function. For more information, see [Using the EVENTDATA Function](http://msdn.microsoft.com/en-us/library/ms187909.aspx).

**Understanding DDL Triggers**

DDL triggers, like regular triggers, fire stored procedures in response to an event. However, unlike DML triggers, they do not fire in response to UPDATE, INSERT, or DELETE statements on a table or view. Instead, they 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, and DROP. Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

|  |
| --- |
| **ms175941.note(en-us,SQL.100).gifImportant:** |
| Test your DDL triggers to determine their responses to system stored procedures that are run. For example, the CREATE TYPE statement and the **sp\_addtype** stored procedure will both fire a DDL trigger that is created on a CREATE\_TYPE event. . |

DDL triggers can be used for administrative tasks such as auditing and regulating database operations.

Use DDL triggers when you want to do the following:

* You want to prevent certain changes to your database schema.
* You want something to occur in the database in response to a change in your database schema.
* You want to record changes or events in the database schema.

DDL triggers fire only after the DDL statements that trigger them are run. DDL triggers cannot be used as INSTEAD OF triggers.

The following example shows how a DDL trigger can be used to prevent any table in a database from being modified or dropped.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl02other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl02other');)

CREATE TRIGGER safety

ON DATABASE

FOR DROP\_TABLE, ALTER\_TABLE

AS

PRINT 'You must disable Trigger "safety" to drop or alter tables!'

ROLLBACK ;

DDL triggers can fire in response to a Transact-SQL event that is processed in the current database or on the current server. The scope of the trigger depends on the event. For more information about the scope of a DDL trigger, see [Designing DDL Triggers](http://msdn.microsoft.com/en-us/library/ms186406.aspx).

To obtain a DDL trigger example that is available in the **AdventureWorks** sample database, in Object Explorer in the SQL Server Management Studio, open the Database Triggers folder located in the Programmability folder of the **AdventureWorks** database. Right-click **ddlDatabaseTriggerLog** and select **Script Database Trigger as**. By default, the DDL trigger **ddlDatabaseTriggerLog** is disabled.

**Understanding DML Triggers**

Microsoft SQL Server provides two primary mechanisms for enforcing business rules and data integrity: constraints and triggers. A trigger is a special type of stored procedure that automatically takes effect when a language event executes. SQL Server includes three general types of triggers: DML triggers, DDL triggers, and logon triggers.

DDL triggers are invoked when a data definition language (DDL) event takes place in the server or database. They are explained in more detail in [DDL Triggers](http://msdn.microsoft.com/en-us/library/ms190989.aspx). 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. For more information, see [Logon Triggers](http://msdn.microsoft.com/en-us/library/bb326598.aspx).

DML triggers are invoked when a data manipulation language (DML) event takes place in the database. DML events include INSERT, UPDATE, or DELETE statements that modify data in a specified table or view. A DML trigger can query other tables and can 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 are useful in these ways:

* They can cascade changes through related tables in the database; however, these changes can be executed more efficiently using cascading referential integrity constraints.
* They can guard against malicious or incorrect INSERT, UPDATE, and DELETE operations and enforce other restrictions that are more complex than those defined with CHECK constraints.   
  Unlike CHECK constraints, DML triggers can reference columns in other tables. For example, a trigger can use a SELECT from another table to compare to the inserted or updated data and to perform additional actions, such as modify the data or display a user-defined error message.
* They can evaluate the state of a table before and after a data modification and take actions based on that difference.
* Multiple DML triggers of the same type (INSERT, UPDATE, or DELETE) on a table allow multiple, different actions to take place in response to the same modification statement.

**Types of DML Triggers**

You can program the following types of DML Triggers:

AFTER Triggers

AFTER triggers are executed after the action of the INSERT, UPDATE, or DELETE statement is performed. Specifying AFTER is the same as specifying FOR, which is the only option available in earlier versions of Microsoft SQL Server. AFTER triggers can be specified only on tables.

INSTEAD OF Triggers

INSTEAD OF triggers are executed in place of the usual triggering action. INSTEAD OF triggers can also be defined on views with one or more base tables, where they can extend the types of updates a view can support.

For more information about AFTER and INSTEAD OF triggers, see [DML Trigger Planning Guidelines](http://msdn.microsoft.com/en-us/library/ms190267.aspx).

CLR Triggers

A CLR Trigger can be either an AFTER or INSTEAD OF trigger. A CLR trigger can also be a DDL trigger. Instead of executing a Transact-SQL stored procedure, a CLR trigger executes one or more methods written in managed code that are members of an assembly created in the .NET Framework and uploaded in SQL Server. For more information, see [Programming CLR Triggers](http://msdn.microsoft.com/en-us/library/ms179562.aspx).

**ALTER TRIGGER (Transact-SQL)**

Modifies the definition of a DML, DDL, or logon trigger that was previously created by the CREATE TRIGGER statement. Triggers are created by using CREATE TRIGGER. They 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. For more information about the parameters that are used in the ALTER TRIGGER statement, see [CREATE TRIGGER (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189799.aspx).

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

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

Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or UPDATE statement (DDL Trigger)

ALTER TRIGGER trigger\_name

ON { DATABASE | ALL SERVER }

[ WITH <ddl\_trigger\_option> [ ,...n ] ]

{ FOR | AFTER } { event\_type [ ,...n ] | event\_group }

AS { sql\_statement [ ; ] | EXTERNAL NAME <method specifier>

[ ; ] }

}

<ddl\_trigger\_option> ::=

[ ENCRYPTION ]

[ <EXECUTE AS Clause> ]

<method\_specifier> ::=

        assembly\_name.class\_name.method\_name

Trigger on a LOGON event (Logon Trigger)

ALTER 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 ]

<method\_specifier> ::=

assembly\_name.class\_name.method\_name

http://i.msdn.microsoft.com/Global/Images/clear.gif 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* is optional only if the DML trigger and its corresponding table or view belong to the default schema. *schema\_name* cannot be specified for DDL or logon triggers.

*trigger\_name*

Is the existing trigger to modify.

*table* | *view*

Is the table or view on which the DML trigger is executed. Specifying the fully-qualified name of the table or view is optional.

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 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

Encrypts the **syscomments** entries that contain the text of the ALTER 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.

|  |
| --- |
| **ms176072.note(en-us,SQL.100).gifNote:** |
| If a trigger is created by using WITH ENCRYPTION, it must be specified again in the ALTER TRIGGER statement for this option to remain enabled. |

EXECUTE AS

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

For more information, see [EXECUTE AS Clause (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188354.aspx).

AFTER

Specifies that the trigger is fired only after the triggering SQL statement is executed successfully. All referential cascade actions and constraint checks also must have been successful before this trigger fires.

AFTER is the default, if only the FOR keyword is specified.

DML AFTER triggers may be defined only on tables.

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 views created by using WITH CHECK OPTION. SQL Server raises an error when an INSTEAD OF trigger is added to a view for which WITH CHECK OPTION was specified. The user must remove that option using ALTER VIEW before defining the INSTEAD OF trigger.

{ [ DELETE ] [ **,** ] [ INSERT ] [ **,** ] [ UPDATE ] } | { [INSERT ] [ **,** ] [ UPDATE ] }

Specifies the data modification statements, when tried against this table or view, activate the DML trigger. At least one option must be specified. Any combination of these in any order is allowed in the trigger definition. If more than one option is specified, separate the options with commas.

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. For more information, see [ALTER TABLE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190273.aspx).

*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](http://msdn.microsoft.com/en-us/library/bb522542.aspx).

*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](http://msdn.microsoft.com/en-us/library/bb510452.aspx). After ALTER 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

Indicates that the trigger should not be executed when a replication agent modifies the table involved in the trigger. For more information, see [Controlling Constraints, Identities, and Triggers with NOT FOR REPLICATION](http://msdn.microsoft.com/en-us/library/ms152529.aspx).

*sql\_statement*

Is the trigger conditions and actions.

<method\_specifier>

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. The class cannot be a nested class.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

For more information about ALTER TRIGGER, see Remarks in [CREATE TRIGGER (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189799.aspx).

DML Triggers

ALTER TRIGGER supports manually updatable views through INSTEAD OF triggers on tables and views. SQL Server applies ALTER TRIGGER the same way for all kinds of triggers (AFTER, INSTEAD-OF).

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 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. The AFTER trigger operation checks for the effects of the triggering statement and also all referential cascade UPDATE and DELETE actions that are caused by the triggering statement.

When a DELETE action to a child or referencing table is the result of a CASCADE on a DELETE from the parent table, and an INSTEAD OF trigger on DELETE is defined on that child table, the trigger is ignored and the DELETE action is executed.

DDL Triggers

Unlike DML triggers, DDL triggers are not scoped to schemas. Therefore, the OBJECT\_ID, OBJECT\_NAME, OBJECTPROPERTY, and OBJECTPROPERTY(EX) cannot be used when querying metadata about DDL triggers. Use the catalog views instead. For more information, see [Getting Information About DDL Triggers](http://msdn.microsoft.com/en-us/library/ms184304.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

To alter a DML trigger requires ALTER permission on the table or view on which the trigger is defined.

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

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

The following example creates a DML trigger that prints a user-defined message to the client when a user tries to add or change data in the SalesPersonQuotaHistory table. The trigger is then modified by using ALTER TRIGGER to apply the trigger only on INSERT activities. This trigger is helpful because it reminds the user that updates or inserts rows into this table to also notify the Compensation department.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl35other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl35other');)

USE AdventureWorks;

GO

IF OBJECT\_ID(N'Sales.bonus\_reminder', N'TR') IS NOT NULL

DROP TRIGGER Sales.bonus\_reminder;

GO

CREATE TRIGGER Sales.bonus\_reminder

ON Sales.SalesPersonQuotaHistory

WITH ENCRYPTION

AFTER INSERT, UPDATE

AS RAISERROR ('Notify Compensation', 16, 10);

GO

-- Now, change the trigger.

USE AdventureWorks;

GO

ALTER TRIGGER Sales.bonus\_reminder

ON Sales.SalesPersonQuotaHistory

AFTER INSERT

AS RAISERROR ('Notify Compensation', 16, 10);

GO

**ENABLE TRIGGER (Transact-SQL)**

Enables a DML, DDL, or logon trigger.

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

ENABLE TRIGGER { [ schema\_name . ] trigger\_name [ ,...n ] | ALL }

ON { object\_name | DATABASE | ALL SERVER } [ ; ]

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

*schema\_name*

Is the name of the schema to which the trigger belongs. *schema\_name* cannot be specified for DDL or logon triggers.

*trigger\_name*

Is the name of the trigger to be enabled.

ALL

Indicates that all triggers defined at the scope of the ON clause are enabled.

*object\_name*

Is the name of the table or view on which the DML trigger *trigger\_name* was created to execute.

DATABASE

For a DDL trigger, indicates that *trigger\_name* was created or modified to execute with database scope.

ALL SERVER

For a DDL trigger, indicates that *trigger\_name* was created or modified to execute with server scope. ALL SERVER also applies to logon triggers.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

Enabling a trigger does not re-create it. A disabled trigger still exists as an object in the current database, but does not fire. Enabling a trigger causes it to fire when any Transact-SQL statements on which it was originally programmed are executed. Triggers are disabled by using [DISABLE TRIGGER](http://msdn.microsoft.com/en-us/library/ms189748.aspx). DML triggers defined on tables can be also be disabled or enabled by using [ALTER TABLE](http://msdn.microsoft.com/en-us/library/ms190273.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

To enable a DML trigger, at a minimum, a user must have ALTER permission on the table or view on which the trigger was created.

To enable a DDL trigger with server scope (ON ALL SERVER) or a logon trigger, a user must have CONTROL SERVER permission on the server. To enable a DDL trigger with database scope (ON DATABASE), at a minimum, a user must have ALTER ANY DATABASE DDL TRIGGER permission in the current database.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

A. Enabling a DML trigger on a table

The following example disables trigger uAddress that was created on table Address, and then enables it.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl22other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl22other');)

USE AdventureWorks;

GO

DISABLE TRIGGER Person.uAddress ON Person.Address;

GO

ENABLE Trigger Person.uAddress ON Person.Address;

GO

B. Enabling a DDL trigger

The following example creates a DDL trigger safety with database scope, and then disables it.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl23other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl23other');)

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

ENABLE TRIGGER safety ON DATABASE;

GO

C. Enabling all triggers that were defined with the same scope

The following example enables all DDL triggers that were created at the server scope.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)

USE AdventureWorks;

GO

ENABLE Trigger ALL ON ALL SERVER;

GO

**DISABLE TRIGGER (Transact-SQL)**

Disables a trigger.

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

DISABLE TRIGGER { [ schema\_name . ] trigger\_name [ ,...n ] | ALL }

ON { object\_name | DATABASE | ALL SERVER } [ ; ]

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

*schema\_name*

Is the name of the schema to which the trigger belongs. *schema\_name* cannot be specified for DDL or logon triggers.

*trigger\_name*

Is the name of the trigger to be disabled.

ALL

Indicates that all triggers defined at the scope of the ON clause are disabled.

|  |
| --- |
| **ms189748.Caution(en-us,SQL.100).gifCaution:** |
| SQL Server creates triggers in databases that are published for merge replication. Specifying ALL in published databases disables these triggers, which disrupts replication. Verify that the current database is not published for merge replication before specifying ALL. |

*object\_name*

Is the name of the table or view on which the DML trigger *trigger\_name* was created to execute.

DATABASE

For a DDL trigger, indicates that *trigger\_name* was created or modified to execute with database scope.

ALL SERVER

For a DDL trigger, indicates that *trigger\_name* was created or modified to execute with server scope. ALL SERVER also applies to logon triggers.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

Triggers are enabled by default when they are created. Disabling a trigger does not drop it. The trigger still exists as an object in the current database. However, the trigger does not fire when any Transact-SQL statements on which it was programmed are executed. Triggers can be re-enabled by using [ENABLE TRIGGER](http://msdn.microsoft.com/en-us/library/ms182706.aspx). DML triggers defined on tables can be also be disabled or enabled by using [ALTER TABLE](http://msdn.microsoft.com/en-us/library/ms190273.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

To disable a DML trigger, at a minimum, a user must have ALTER permission on the table or view on which the trigger was created.

To disable a DDL trigger with server scope (ON ALL SERVER) or a logon trigger, a user must have CONTROL SERVER permission on the server. To disable a DDL trigger with database scope (ON DATABASE), at a minimum, a user must have ALTER ANY DATABASE DDL TRIGGER permission in the current database.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

A. Disabling a DML trigger on a table

The following example disables trigger uAddress that was created on table Address.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)

USE AdventureWorks;

GO

DISABLE TRIGGER Person.uAddress ON Person.Address;

GO

B. Disabling a DDL trigger

The following example creates a DDL trigger safety with database scope, and then disables it.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl25other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl25other');)

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

C. Disabling all triggers that were defined with the same scope

The following example disables all DDL triggers that were created at the server scope.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl26other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl26other');)

USE AdventureWorks;

GO

DISABLE Trigger ALL ON ALL SERVER;

GO

**TRIGGER\_NESTLEVEL (Transact-SQL)**

Returns the number of triggers executed for the statement that fired the trigger. TRIGGER\_NESTLEVEL is used in DML and DDL triggers to determine the current level of nesting.

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

TRIGGER\_NESTLEVEL ( [ object\_id ] , [ 'trigger\_type' ] , [ 'trigger\_event\_category' ] )

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

*object\_id*

Is the object ID of a trigger. If *object\_id* is specified, the number of times the specified trigger has been executed for the statement is returned. If *object\_id* is not specified, the number of times all triggers have been executed for the statement is returned.

**'** *trigger\_type* **'**

Specifies whether to apply TRIGGER\_NESTLEVEL to AFTER triggers or INSTEAD OF triggers. Specify **AFTER** for AFTER triggers. Specify **IOT** for INSTEAD OF triggers. If *trigger\_type* is specified, *trigger\_event\_category* must also be specified.

**'** *trigger\_event\_category* **'**

Specifies whether to apply TRIGGER\_NESTLEVEL to DML or DDL triggers. Specify **DML** for DML triggers. Specify **DDL** for DDL triggers. If *trigger\_event\_category* is specified, *trigger\_type* must also be specified. Note that only **AFTER** can be specified with **DDL**, because DDL triggers can only be AFTER triggers.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

When no parameters are specified, TRIGGER\_NESTLEVEL returns the total number of triggers on the call stack. This includes itself. Omission of parameters can occur when a trigger executes commands causing another trigger to be fired or creates a succession of firing triggers.

To return the total number of triggers on the call stack for a particular trigger type and event category, specify *object\_id* = 0.

TRIGGER\_NESTLEVEL returns 0 if it is executed outside a trigger and any parameters are not NULL.

When any parameters are explicitly specified as NULL, a value of NULL is returned regardless of whether TRIGGER\_NESTLEVEL was used within or external to a trigger.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

A. Testing the nesting level of a specific DML trigger

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl16other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl16other');)

IF ( (SELECT TRIGGER\_NESTLEVEL( OBJECT\_ID('xyz') , 'AFTER' , 'DML' ) ) > 5 )

RAISERROR('Trigger xyz nested more than 5 levels.',16,-1)

B. Testing the nesting level of a specific DDL trigger

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl17other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl17other');)

IF ( ( SELECT TRIGGER\_NESTLEVEL ( ( SELECT object\_id FROM sys.triggers

WHERE name = 'abc' ), 'AFTER' , 'DDL' ) ) > 5 )

RAISERROR ('Trigger abc nested more than 5 levels.',16,-1)

C. Testing the nesting level of all triggers executed

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl18other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl18other');)

IF ( (SELECT trigger\_nestlevel() ) > 5 )

RAISERROR

('This statement nested over 5 levels of triggers.',16,-1)

**DROP TRIGGER (Transact-SQL)**

Removes one or more DML or DDL triggers from the current database.

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

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

DROP TRIGGER [schema\_name.]trigger\_name [ ,...n ] [ ; ]

Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE or UPDATE statement (DDL Trigger)

DROP TRIGGER trigger\_name [ ,...n ]

ON { DATABASE | ALL SERVER }

[ ; ]

Trigger on a LOGON event (Logon Trigger)

DROP TRIGGER trigger\_name [ ,...n ]

ON ALL SERVER

http://i.msdn.microsoft.com/Global/Images/clear.gif 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 to remove. To see a list of currently created triggers, use [sys.server\_assembly\_modules](http://msdn.microsoft.com/en-us/library/ms188746.aspx) or [sys.server\_triggers](http://msdn.microsoft.com/en-us/library/ms176054.aspx).

DATABASE

Indicates the scope of the DDL trigger applies to the current database. DATABASE must be specified if it was also specified when the trigger was created or modified.

ALL SERVER

Indicates the scope of the DDL trigger applies to the current server. ALL SERVER must be specified if it was also specified when the trigger was created or modified. ALL SERVER also applies to logon triggers.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

You can remove a DML trigger by dropping it or by dropping the trigger table. When a table is dropped, all associated triggers are also dropped.

When a trigger is dropped, information about the trigger is removed from the **sys.objects**, **sys.triggers** and **sys.sql\_modules** catalog views.

Multiple DDL triggers can be dropped per DROP TRIGGER statement only if all triggers were created using identical ON clauses.

To rename a trigger, use DROP TRIGGER and CREATE TRIGGER. To change the definition of a trigger, use ALTER TRIGGER.

For more information about determining dependencies for a specific trigger, see [sys.sql\_expression\_dependencies](http://msdn.microsoft.com/en-us/library/bb677315.aspx), [sys.dm\_sql\_referenced\_entities (Transact-SQL)](http://msdn.microsoft.com/en-us/library/bb677185.aspx), and [sys.dm\_sql\_referencing\_entities (Transact-SQL)](http://msdn.microsoft.com/en-us/library/bb630351.aspx).

For more information about viewing the text of the trigger, see [sp\_helptext (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms176112.aspx) and [sys.sql\_modules (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms175081.aspx).

For more information about viewing a list of existing triggers, see [sys.triggers (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188746.aspx) and [sys.server\_triggers (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms176054.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

To drop a DML trigger requires ALTER permission on the table or view on which the trigger is defined.

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

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

A. Dropping a DML trigger

The following example drops the employee\_insupd trigger.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl36other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl36other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ('employee\_insupd', 'TR') IS NOT NULL

DROP TRIGGER employee\_insupd;

GO

B. Dropping a DDL trigger

The following example drops DDL trigger safety.

|  |
| --- |
| **ms173497.note(en-us,SQL.100).gifImportant:** |
| Because DDL triggers are not schema-scoped and, therefore do not appear in the **sys.objects** catalog view, the OBJECT\_ID function cannot be used to query whether they exist in the database. Objects that are not schema-scoped must be queried by using the appropriate catalog view. For DDL triggers, use **sys.triggers**. |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl38other');)Copy Code](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl38other');)

USE AdventureWorks;

GO

IF EXISTS (SELECT \* FROM sys.triggers

WHERE parent\_class = 0 AND name = 'safety')

DROP TRIGGER safety

ON DATABASE;

GO