# [Triggers](https://docs.microsoft.com/en-us/sql/relational-databases/triggers/logon-triggers)

## Logon Triggers

Logon triggers fire stored procedures in response to a LOGON event. This event is raised when a user session is established with an instance of SQL Server. Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established. 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. Logon triggers do not fire if authentication fails.1

You can use logon triggers to audit and control server sessions, such as by tracking login activity, restricting logins to SQL Server, or limiting the number of sessions for a specific login.

For example, in the following code, the logon trigger denies log in attempts to SQL Server initiated by login login\_test if there are already three user sessions created by that login.

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;

## DDL Triggers

DDL triggers fire in response to a variety of Data Definition Language (DDL) events. These events primarily correspond to Transact-SQL statements that start with the keywords CREATE, ALTER, DROP, GRANT, DENY, REVOKE or UPDATE STATISTICS. Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

Use DDL triggers when you want to do the following:

* Prevent certain changes to your database schema.
* Have something occur in the database in response to a change in your database schema.
* Record changes or events in the database schema.

Important

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 Trigger Scope

DDL triggers can fire in response to a Transact-SQL event processed in the current database, or on the current server. The scope of the trigger depends on the event. For example, a DDL trigger created to fire in response to a CREATE\_TABLE event can do so whenever a CREATE\_TABLE event occurs in the database, or on the server instance. A DDL trigger created to fire in response to a CREATE\_LOGIN event can do so only when a CREATE\_LOGIN event occurs in the server instance.

CREATE TRIGGER safety ON DATABASE FOR DROP\_TABLE, ALTER\_TABLE

AS

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

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

### Use the EVENTDATA Function

Information about an event that fires a DDL trigger is captured by using the EVENTDATA function. This function returns an **xml** value. The XML schema includes information about the following:

* The time of the event.
* The System Process ID (SPID) of the connection when the trigger executed.
* The type of event that fired the trigger.

Depending on the event type, the schema then includes additional information such as the database in which the event occurred, the object against which the event occurred, and the Transact-SQL statement of the event.

## DML Triggers

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

### DML Trigger Benefits

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

The following list compares DML triggers with constraints and identifies when DML triggers have benefits over .

* DML triggers can cascade changes through related tables in the database; however, these changes can be executed more efficiently using cascading referential integrity constraints. FOREIGN KEY constraints can validate a column value only with an exact match to a value in another column, unless the REFERENCES clause defines a cascading referential action.
* 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.
* Constraints can communicate about errors only through standardized system error messages. If your application requires, or can benefit from, customized messages and more complex error handling, you must use a trigger.
* DML triggers can disallow or roll back changes that violate referential integrity, thereby canceling the attempted data modification. Such a trigger might go into effect when you change a foreign key and the new value does not match its primary key. However, FOREIGN KEY constraints are usually used for this purpose.
* If constraints exist on the trigger table, they are checked after the INSTEAD OF trigger execution but prior to the AFTER trigger execution. If the constraints are violated, the INSTEAD OF trigger actions are rolled back and the AFTER trigger is not executed.

### Types of DML Triggers

AFTER trigger  
AFTER triggers are executed after the action of the INSERT, UPDATE, MERGE, or DELETE statement is performed. AFTER triggers are never executed if a constraint violation occurs; therefore, these triggers cannot be used for any processing that might prevent constraint violations. For every INSERT, UPDATE, or DELETE action specified in a MERGE statement, the corresponding trigger is fired for each DML operation.

INSTEAD OF trigger  
INSTEAD OF triggers override the standard actions of the triggering statement. Therefore, they can be used to perform error or value checking on one or more columns and the perform additional actions before insert, updating or deleting the row or rows. For example, when the value being updated in an hourly wage column in a payroll table exceeds a specified value, a trigger can be defined to either produce an error message and roll back the transaction, or insert a new record into an audit trail before inserting the record into the payroll table. The primary advantage of INSTEAD OF triggers is that they enable views that would not be updatable to support updates. For example, a view based on multiple base tables must use an INSTEAD OF trigger to support inserts, updates, and deletes that reference data in more than one table. Another advantage of INSTEAD OF triggers is that they enable you to code logic that can reject parts of a batch while letting other parts of a batch to succeed.

CREATE TRIGGER NewPODetail3 ON Purchasing.PurchaseOrderDetail FOR INSERT AS

CREATE TRIGGER NewPODetail ON Purchasing.PurchaseOrderDetail AFTER INSERT AS

# Create Nested Triggers

Both DML and DDL triggers are nested when a trigger performs an action that initiates another trigger. These actions can initiate other triggers, and so on. DML and DDL triggers can be nested up to 32 levels. You can control whether AFTER triggers can be nested through the **nested triggers** server configuration option. INSTEAD OF triggers (only DML triggers can be INSTEAD OF triggers) can be nested regardless of this setting.

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

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)

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

RAISERROR

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

##### Recursive Triggers

An AFTER trigger does not call itself recursively unless the RECURSIVE\_TRIGGERS database option is set.

There are two types of recursion:

* Direct recursion

This recursion occurs when a trigger fires and performs an action that causes the same trigger to fire again. For example, an application updates table **T3**; this causes trigger **Trig3** to fire. **Trig3** updates table **T3** again; this causes trigger **Trig3** to fire again.

Direct recursion can also occur when the same trigger is called again, but after a trigger of a different type (AFTER or INSTEAD OF) is called. In other words, direct recursion of an INSTEAD OF trigger can occur when the same INSTEAD OF trigger is called for a second time, even if one or more AFTER triggers are called in between. Likewise, direct recursion of an AFTER trigger can occur when the same AFTER trigger is called for a second time, even if one or more INSTEAD OF triggers are called in between. For example, an application updates table **T4**. This update causes INSTEAD OF trigger **Trig4** to fire. **Trig4** updates table **T5**. This update causes AFTER trigger **Trig5** to fire. **Trig5** updates table **T4**, and this update causes INSTEAD OF trigger **Trig4** to fire again. This chain of events is considered direct recursion for **Trig4**.

* Indirect recursion

This recursion occurs when a trigger fires and performs an action that causes another trigger of the same type (AFTER or INSTEAD OF) to fire. This second trigger performs an action that causes the original trigger to fire again. In other words, indirect recursion can occur when an INSTEAD OF trigger is called for a second time, but not until another INSTEAD OF trigger is called in between. Likewise, indirect recursion can occur when an AFTER trigger is called for a second time, but not until another AFTER trigger is called in between. For example, an application updates table **T1**. This update causes AFTER trigger **Trig1** to fire. **Trig1** updates table **T2**, and this update causes AFTER trigger **Trig2** to fire. **Trig2** in turn updates table **T1** that causes AFTER trigger **Trig1** to fire again.

Only direct recursion of AFTER triggers is prevented when the RECURSIVE\_TRIGGERS database option is set to OFF. To disable indirect recursion of AFTER triggers, also set the **nested triggers** server option to **0**.

### Specify First and Last Triggers

You can specify that one of the AFTER triggers associated with a table be either the first AFTER trigger or the last AFTER trigger that is fired for each INSERT, DELETE, and UPDATE triggering actions. The AFTER triggers that are fired between the first and last triggers are executed in undefined order.

sp\_settriggerorder @triggername = 'MyTrigger', @order = 'first/lant/none', @stmttype = 'UPDATE'

## Use the inserted and deleted Tables

DML trigger statements use two special tables: the deleted table and the inserted tables. SQL Server automatically creates and manages these tables. You can use these temporary, memory-resident tables to test the effects of certain data modifications and to set conditions for DML trigger actions. You cannot directly modify the data in the tables or perform data definition language (DDL) operations on the tables, such as CREATE INDEX.

In DML triggers, the inserted and deleted tables are primarily used to perform the following:

* Extend referential integrity between tables.
* Insert or update data in base tables underlying a view.
* Test for errors and take action based on the error.
* Find the difference between the state of a table before and after a data modification and take actions based on that difference.

The deleted table stores copies of the affected rows during DELETE and UPDATE statements. During the execution of a DELETE or UPDATE statement, rows are deleted from the trigger table and transferred to the deleted table. The deleted table and the trigger table ordinarily have no rows in common.

The inserted table stores copies of the affected rows during INSERT and UPDATE statements. During an insert or update transaction, new rows are added to both the inserted table and the trigger table. The rows in the inserted table are copies of the new rows in the trigger table.

An update transaction is similar to a delete operation followed by an insert operation; the old rows are copied to the deleted table first, and then the new rows are copied to the trigger table and to the inserted table.

When you set trigger conditions, use the inserted and deleted tables appropriately for the action that fired the trigger. Although referencing the deleted table when testing an INSERT or the inserted table when testing a DELETE does not cause any errors, these trigger test tables do not contain any rows in these cases.

The inserted and deleted tables passed to INSTEAD OF triggers defined on tables follow the same rules as the inserted and deleted tables passed to AFTER triggers. The format of the inserted and deleted tables is the same as the format of the table on which the INSTEAD OF trigger is defined. Each column in the inserted and deleted tables maps directly to a column in the base table.

The following rules regarding when an INSERT or UPDATE statement referencing a table with an INSTEAD OF trigger must supply values for columns are the same as if the table did not have an INSTEAD OF trigger:

* Values cannot be specified for computed columns or columns with a **timestamp** data type.
* Values cannot be specified for columns with an IDENTITY property, unless IDENTITY\_INSERT is ON for that table. When IDENTITY\_INSERT is ON, INSERT statements must supply a value.
* INSERT statements must supply values for all NOT NULL columns that do not have DEFAULT constraints.
* For any columns except computed, identity, or **timestamp** columns, values are optional for any column that allows nulls, or any NOT NULL column that has a DEFAULT definition.

## Delete or Disable DML Triggers

**DROP** **TRIGGER** <Trigger\_Name>;

DISABLE TRIGGER <Trigger\_Name> ON <Table\_Name>;

DISABLE TRIGGER ALL ON DATABASE

DISABLE TRIGGER ALL ON ALL SERVER

ENABLE TRIGGER <Trigger\_Name> ON <Table\_Name>;