CREATE TRIGGER (Transact-SQL)

Applies to: SQL Server · Azure SQL Database · Azure SQL Managed Instance · SQL database in Microsoft Fabric Preview

Creates a DML, DDL, or logon trigger. A trigger is a special type of stored procedure that automatically runs when an event occurs in the database server. DML triggers run 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 fires, whether table rows are affected or not. For more information, see DML Triggers.

DDL triggers run in response to various 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's session is being established. You can create triggers 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 lets you create 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 article. CLR integration doesn't apply to Azure SQL Database or SQL database in Microsoft Fabric Preview.

Transact-SQL syntax conventions

Syntax

SQL Server syntax

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

```
syntaxsql
CREATE [ OR ALTER ] 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 an `INSERT`, `UPDATE`, or `DELETE` statement to a table (DML trigger on memory-optimized tables):

```
syntaxsql
CREATE [ OR ALTER ] 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` statement (DDL trigger):

```
syntaxsql
CREATE [ OR ALTER ] 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):

syntaxsql

```
CREATE [ OR 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 ]
```

Azure SQL Database or SQL database in Fabric syntax

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

syntaxsql

```
CREATE [ OR ALTER ] 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 ]
```

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

syntaxsql

```
CREATE [ OR ALTER ] 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

OR ALTER

Applies to: SQL Server 2016 (13.x) SP1 and later versions, Azure SQL Database, and SQL database in Microsoft Fabric Preview.

Conditionally alters the trigger only if it already exists.

schema_name

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're created. *schema_name* can't be specified for DDL or logon triggers.

trigger_name

The name of the trigger. A *trigger_name* must follow the rules for identifiers, except that *trigger_name* can't start with `#` or `##`.

table | view

The table or view on which the DML trigger runs. This table or view is sometimes referred to as the trigger table or trigger view. Specifying the fully qualified name of the table or view is optional. You can only reference a view by an `INSTEAD OF` trigger. You can't define DML triggers 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

Obscures the text of the `CREATE TRIGGER` statement. Using `WITH ENCRYPTION` prevents the trigger from being published as part of SQL Server replication. `WITH ENCRYPTION` can't 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, see EXECUTE AS Clause.

NATIVE_COMPILATION

Indicates that the trigger is natively compiled.

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

SCHEMABINDING

Ensures that tables referenced by a trigger can't be dropped or altered.

This option is required for triggers on memory-optimized tables and isn't supported for triggers on traditional tables.

FOR | AFTER

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

You can't define `AFTER` triggers on views.

INSTEAD OF

Specifies that the DML trigger launches *instead of* the triggering SQL statement, thus, overriding the actions of the triggering statements. You can't specify `INSTEAD OF` for DDL or logon triggers.

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

You can't define `INSTEAD OF` triggers on updatable views that use `WITH CHECK OPTION`. Doing so results in an error when an `INSTEAD OF` trigger is added to an updatable view `WITH CHECK OPTION` specified. You 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's tried against this table or view. Specify at least one option. Use any combination of these options in any order in the trigger definition.

For `INSTEAD OF` triggers, you can't use the `DELETE` option on tables that have a referential relationship, specifying a cascade action `ON DELETE`. Similarly, the `UPDATE` option isn't allowed on tables that have a referential relationship, specifying a cascade action `ON UPDATE`.

WITH APPEND

Applies to: SQL Server 2008 (10.0.x) through SQL Server 2008 R2 (10.50.x).

Specifies that an additional trigger of an existing type should be added. `WITH APPEND` can't be used with `INSTEAD OF` triggers or if an `AFTER` trigger is explicitly stated. For backward compatibility, only use `WITH APPEND` when `FOR` is specified, without `INSTEAD OF` or `AFTER`. You can't specify `WITH APPEND` if using `EXTERNAL NAME` (that is, if the trigger is a CLR trigger).

event_type

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

event_group

The name of a predefined grouping of Transact-SQL language events. The DDL trigger fires after launch 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

Indicates that the trigger shouldn't be run when a replication agent modifies the table that's involved in the trigger.

sql_statement

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 run.

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

Triggers can include any number and type 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. The trigger shouldn't 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're 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 might be changed by the user action. For example, to retrieve all values in the `deleted` table, use:

SQL

```
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 function. For more information, see Use the EVENTDATA Function.

SQL Server allows 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 Microsoft SQL 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>

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 can't 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 don't run in an instance of SQL Server unless the clr enabled option is enabled with sp_configure.

Remarks for 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 doesn't 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're checked after the `INSTEAD OF` trigger runs and before the `AFTER` trigger runs. If the constraints are violated, the `INSTEAD OF` trigger actions are rolled back and the `AFTER` trigger isn't fired.

You can specify the first and last `AFTER` triggers to be run on a table by using `sp_settriggerorder`. You can specify only one first and one last `AFTER` trigger for each `INSERT`, `UPDATE`, and `DELETE` operation on a table. If there are other `AFTER` triggers on the same table, they're randomly run.

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

An `AFTER` trigger is run only after the triggering SQL statement runs successfully. This successful execution includes all referential cascade actions and constraint checks associated with the object updated or deleted. An `AFTER` doesn't recursively fire an `INSTEAD OF` trigger on the same table.

If an `INSTEAD OF` trigger defined on a table runs a statement against the table that would ordinarily fire the `INSTEAD OF` trigger again, the trigger isn't called recursively. Instead, the statement processes 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, if the trigger runs an `INSERT` statement on the same table, the `INSERT` statement launched by the `INSTEAD OF` trigger doesn't call the trigger again. The `INSERT` launched by the trigger starts the process of running constraint actions and firing any `AFTER INSERT` triggers defined for the table.

When an `INSTEAD OF` trigger defined on a view runs a statement against the view that would ordinarily fire the `INSTEAD OF` trigger again, it's 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 runs an `UPDATE` statement referencing the same view, the `UPDATE` statement launched by the `INSTEAD OF` trigger doesn't call the trigger again. The `UPDATE` launched by the trigger is processed against the view as if the view didn't 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.

Test for UPDATE or INSERT actions to specific columns

You can design a Transact-SQL trigger to do 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` attempts on one column. `COLUMNS_UPDATED` tests for `UPDATE` or `INSERT` actions that run on multiple columns. This function 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`/`INSTEAD OF UPDATE` triggers can't 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 results being returned to an application because of a trigger firing, don't include either `SELECT` statements that return results or statements that carry out variable assignment in a trigger. A trigger that includes either `SELECT` statements that return results to the user or statements that do variable assignment, requires special handling. You'd have to write the returned results 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 doesn't activate a trigger because the operation doesn't log individual row deletions. However, only those users with permissions to run a `TRUNCATE TABLE` statement need be concerned about inadvertently circumventing a `DELETE` trigger this way.

The `WRITETEXT` statement, whether logged or unlogged, doesn't activate a trigger.

The following Transact-SQL statements aren't allowed in a DML trigger:

- `ALTER DATABASE`
- `CREATE DATABASE`
- `DROP DATABASE`
- `RESTORE DATABASE`
- `RESTORE LOG`
- `RECONFIGURE`

Additionally, the following Transact-SQL statements aren't allowed inside the body of a DML trigger when it's used against the table or view that's the target of the triggering action.

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

> **Note**
>
> Because SQL Server doesn't support user-defined triggers on system tables, we recommend that you don't create user-defined triggers on system tables.

Optimize DML triggers

Triggers work in transactions (implied or otherwise) and while they're open, they lock resources. The lock remains in place until the transaction is confirmed (with `COMMIT`) or rejected (with a `ROLLBACK`). The longer a trigger runs, the higher the probability that another process is then blocked. So, write triggers to lessen their duration whenever possible. One way to achieve shorter duration is to release a trigger when a DML statement changes zero rows.

To release the trigger for a command that doesn't change any rows, employ the system variable ROWCOUNT_BIG.

The following T-SQL code snippet shows how to release the trigger for a command that doesn't change any rows. This code should be present at the beginning of each DML trigger:

```
SQL
IF (ROWCOUNT_BIG() = 0)
RETURN;
```

Remarks for DDL triggers

DDL triggers, like standard triggers, launch stored procedures in response to an event. But, unlike standard triggers, they don't run in response to `UPDATE`, `INSERT`, or `DELETE` statements on a table or view. Instead, they primarily run in response to data definition language (DDL) statements. The statement types include `CREATE`, `ALTER`, `DROP`, `GRANT`, `DENY`, `REVOKE`, and `UPDATE STATISTICS`. Certain system stored procedures that carry out 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 fire a DDL trigger that's created on a `CREATE_TYPE` event.

For more information about DDL triggers, see DDL triggers.

DDL triggers don't fire in response to events that affect local or global temporary tables and stored procedures.

Unlike DML triggers, DDL triggers aren't scoped to schemas. So, you can't use functions such as `OBJECT_ID`, `OBJECT_NAME`, `OBJECTPROPERTY`, and `OBJECTPROPERTYEX` 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 carry out stored procedures in response to a `LOGON` event. This event happens 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 established. So, 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 don't fire if authentication fails.

Distributed transactions aren't supported in a logon trigger. Error 3969 returns when a logon trigger that contains a distributed transaction fire.

Disable 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

Return results

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

Logon triggers always disallow the return of results sets and this behavior isn't configurable. If a logon trigger generates a result set, the trigger fails to launch and the login attempt that fired the

trigger is denied.

Multiple triggers

SQL Server lets you create multiple triggers for each DML, DDL, or `LOGON` event. For example, if `CREATE TRIGGER FOR UPDATE` is run 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 supports 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`. Trigger `T2` then fires and updates table `T1`.
- **Direct recursion**: In 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 runs each `TR1` and `TR2` one time. Additionally, the launch 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's no defined order in which multiple triggers defined for a specific event are run. 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 carries out a `ROLLBACK TRANSACTION`, regardless of the nesting level, no more triggers are run.

Nested triggers

You can nest triggers to a maximum of 32 levels. If a trigger changes a table on which there's another trigger, the second trigger activates 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 launches 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 don't count against this limit.

To disable nested triggers, set the nested triggers option of `sp_configure` to 0 (off). The default configuration supports nested triggers. If nested triggers are off, recursive triggers are also disabled, despite the `RECURSIVE_TRIGGERS` setting that's 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 0. But, under this setting, the later `AFTER` triggers don't fire. Review your applications for nested triggers to determine if the applications follow your business rules when the **nested triggers** server configuration option is set to 0. If not, make the appropriate modifications.

Deferred name resolution

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

Permissions

To create a DML trigger, it 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. Use 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 AdventureWorks2022 database.

SQL

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

B. Use 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.

SQL

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
    EXECUTE msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks2022 Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales force.',
        @subject = 'Reminder';
GO
```

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

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

The following example creates a DML trigger in the `AdventureWorks2022` database. This trigger checks to make sure the credit rating for the vendor is good (not 5) when there's an attempt to insert a new purchase order into the `PurchaseOrderHeader` table. To get the credit rating of the

vendor, the `Vendor` table must be referenced. If the credit rating is too low, a message appears and the insertion doesn't happen.

SQL

```
USE AdventureWorks2022;
GO

IF OBJECT_ID('Purchasing.LowCredit', 'TR') IS NOT NULL
    DROP TRIGGER Purchasing.LowCredit;
GO

-- 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 (ROWCOUNT_BIG() = 0)
    RETURN;
    IF EXISTS (SELECT 1
        FROM inserted AS i
        INNER JOIN Purchasing.Vendor AS v
            ON v.BusinessEntityID = i.VendorID
            WHERE v.CreditRating = 5)
BEGIN
    RAISERROR ('A vendor''s credit rating is too low to accept new purchase orders.',
16, 1);
    ROLLBACK;
    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. Use a database-scoped DDL trigger

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

SQL

```
CREATE TRIGGER safety
    ON DATABASE
    FOR DROP_SYNONYM
    AS IF (@@ROWCOUNT = 0)
           RETURN;
       RAISERROR ('You must disable Trigger "safety" to remove synonyms!', 10, 1);
       ROLLBACK;
GO

DROP TRIGGER safety
    ON DATABASE;
GO
```

E. Use 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.

SQL

```
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. Use 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. Change `<password>` to a strong password.

SQL

```
USE master;
GO

CREATE LOGIN login_test
    WITH PASSWORD = '<password>' 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. View 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. The trigger, `safety`, is created in example D. Use a database-scoped DDL trigger.

SQL

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

Related content

- 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
- sys.dm_sql_referencing_entities
- sys.sql_expression_dependencies
- sp_help
- sp_helptrigger
- sp_helptext
- sp_rename
- sp_settriggerorder
- UPDATE - Trigger Functions (Transact-SQL)
- Get Information About DML Triggers
- Get Information About DDL Triggers
- sys.triggers
- sys.trigger_events
- sys.sql_modules
- sys.assembly_modules
- sys.server_triggers
- sys.server_trigger_events
- sys.server_sql_modules
- sys.server_assembly_modules

Last updated on 09/30/2025