**Triggers in SQL Server**

A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.

A **trigger is called a special procedure** because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

The following are the main characteristics that distinguish triggers from stored procedures:

* We cannot manually execute/invoked triggers.
* Triggers have no chance of receiving parameters.
* A transaction cannot be committed or rolled back inside a trigger.

Syntax of Trigger

We can create a trigger in [SQL Server](https://www.javatpoint.com/sql-server-tutorial) by using the **CREATE TRIGGER** statement as follows:

**CREATE** **TRIGGER** **schema**.trigger\_name

**ON** table\_name

**AFTER**  {**INSERT**, **UPDATE**, **DELETE**}

[NOT **FOR** REPLICATION]

**AS**

{SQL\_Statements}

The parameter descriptions of this syntax illustrate below:

**schema:** It is an optional parameter that defines which schema the new trigger belongs to.

**trigger\_name:** It is a required parameter that defines the name for the new trigger.

**table\_name:** It is a required parameter that defines the table name to which the trigger applies. Next to the table name, we need to write the AFTER clause where any events like INSERT, UPDATE, or DELETE could be listed.

**NOT FOR REPLICATION:** This option tells that [SQL](https://www.javatpoint.com/sql-tutorial) Server does not execute the trigger when data is modified as part of a replication process.

**SQL\_Statements:** It contains one or more SQL statements that are used to perform actions in response to an event that occurs.

When we use triggers?

Triggers will be helpful when we need to execute some events automatically on certain desirable scenarios. **For example**, we have a constantly changing table and need to know the occurrences of changes and when these changes happen. If the primary table made any changes in such scenarios, we could create a trigger to insert the desired data into a separate table.

Example of Trigger in SQL Server

Let us understand how we can work with triggers in the SQL Server. We can do this by first creating a table named '**Employee'** using the below statements:

**CREATE** **TABLE** Employee

(

  Id **INT** **PRIMARY** **KEY**,

**Name** **VARCHAR**(45),

  Salary **INT**,

  Gender **VARCHAR**(12),

  DepartmentId **INT**

)

Next, we will insert some record into this table as follows:

**INSERT** **INTO** Employee **VALUES** (1,'Steffan', 82000, 'Male', 3),

(2,'Amelie', 52000, 'Female', 2),

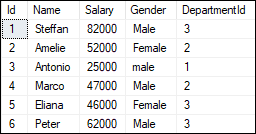
(3,'Antonio', 25000, 'male', 1),

(4,'Marco', 47000, 'Male', 2),

(5,'Eliana', 46000, 'Female', 3)

We can verify the insert operation by using the SELECT statement. We will get the below output:

**SELECT** \* **FROM** Employee;



We will also create another table named '**Employee\_Audit\_Test'** to automatically store transaction records of each operation, such as INSERT, UPDATE, or DELETE on the Employee table:

**CREATE** **TABLE** Employee\_Audit\_Test

(

Id **int** IDENTITY,

Audit\_Action text

)

Now, we will **create a trigger that stores transaction records of each insert operation** on the Employee table into the Employee\_Audit\_Test table. Here we are going to create the insert trigger using the below statement:

**CREATE** **TRIGGER** trInsertEmployee

**ON** Employee

**FOR** **INSERT**

**AS**

**BEGIN**

**Declare** @Id **int**

**SELECT** @Id = Id **from** inserted

**INSERT** **INTO** Employee\_Audit\_Test

**VALUES** ('New employee with Id = ' + CAST(@Id **AS** **VARCHAR**(10)) + ' is added at ' + CAST(Getdate() **AS** **VARCHAR**(22)))

**END**

After creating a trigger, we will try to add the following record into the table:

1. **INSERT** **INTO** Employee **VALUES** (6,'Peter', 62000, 'Male', 3)

If no error is found, execute the SELECT statement to check the audit records. We will get the output as follows:

Triggers in SQL Server

We are going to **create another trigger to store transaction records of each delete operation** on the Employee table into the Employee\_Audit\_Test table. We can create the delete trigger using the below statement:

**CREATE** **TRIGGER** trDeleteEmployee

**ON** Employee

**FOR** **DELETE**

**AS**

**BEGIN**

**Declare** @Id **int**

**SELECT** @Id = Id **from** deleted

**INSERT** **INTO** Employee\_Audit\_Test

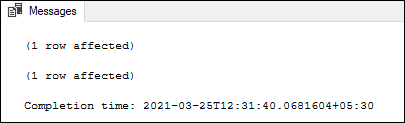
**VALUES** ('An existing employee with Id = ' + CAST(@Id **AS** **VARCHAR**(10)) + ' is deleted at ' + CAST(Getdate() **AS** **VARCHAR**(22)))

**END**

After creating a trigger, we will delete a record from the Employee table:

1. **DELETE** **FROM** Employee **WHERE** Id = 2;

If no error is found, it gives the message as below:



Finally, execute the SELECT statement to check the audit records:

Triggers in SQL Server

In both the triggers code, you will notice these lines:

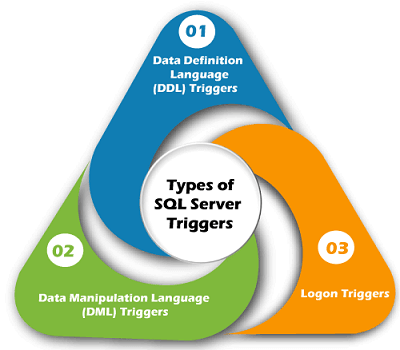
1. **SELECT** @Id = Id **from** inserted
2. **SELECT** @Id = Id **from** deleted

Here inserted and deleted are special tables used by the SQL Server. The inserted table keeps the copy of the row when you insert a new row into the actual table. And the deleted table keeps the copy of the row you have just deleted from the actual table.

Types of SQL Server Triggers

We can categorize the triggers in SQL Server in mainly three types:

1. Data Definition Language (DDL) Triggers
2. Data Manipulation Language (DML) Triggers
3. Logon Triggers



DDL Triggers

DDL triggers are fired in response to the DDL events, such as CREATE, ALTER, and DROP statements. We can create these triggers at the database level or server level, depending on the type of DDL events. It can also be executed in response to certain system-defined stored procedures that do DDL-like operations.

**The DDL triggers are useful in the following scenario:**

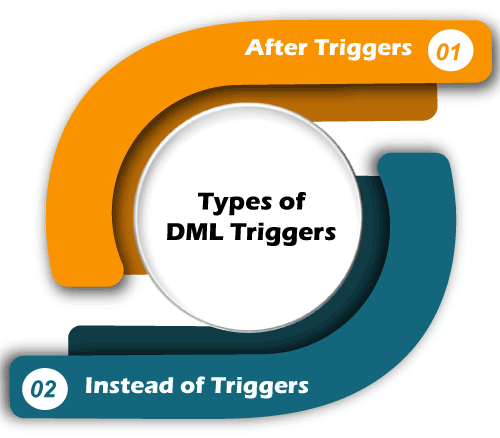
* When we need to prevent the database schema from changing
* When we need to audit changes made in the database schema
* When we need to respond to a change made in the database schema

DML Triggers

DML triggers are fired in response to DML events like INSERT, UPDATE, and DELETE statements in the user's table or view. It can also be executed in response to DML-like operations performed by system-defined stored procedures.

**The DML triggers can be classified into two types:**

* After Triggers
* Instead Of Triggers

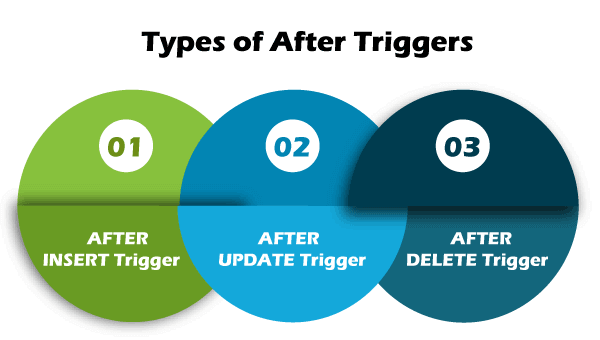


After Triggers

After trigger fires, when SQL Server completes the triggering action successfully, that fired it. Generally, this trigger is executed when a table completes an insert, update or delete operations. It is not supported in views. Sometimes it is known as FOR triggers.

**We can classify this trigger further into three types:**

1. AFTER INSERT Trigger
2. AFTER UPDATE Trigger
3. AFTER DELETE Trigger



**Example:** When we insert data into a table, the trigger associated with the insert operation on that table will not fire until the row has passed all constraints, such as the primary key constraint. SQL Server cannot fire the AFTER trigger when the data insertion failed.

**The following is illustration of the After Triggers syntax in SQL Server:**

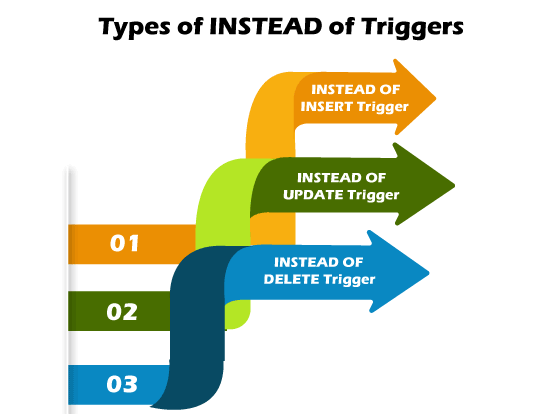
1. **CREATE** **TRIGGER** schema\_name.trigger\_name
2. **ON** table\_name
3. **AFTER** {**INSERT** | **UPDATE** | **DELETE**}
4. **AS**
5. **BEGIN**
6. -- Trigger Statements
7. -- Insert, Update, Or Delete Statements
8. **END**

Instead of Triggers

Instead of Trigger fires before SQL Server begins to execute the triggering operation that triggered it. It means that no condition constraint check is needed before the trigger runs. As a result, even if the constraint check fails, this trigger will fire. It is the opposite of the AFTER trigger. We can create the INSTEAD OF triggers on a table that executes successfully but doesn't contain the table's actual insert, update, or delete operations.

**We can classify this trigger further into three types:**

1. INSTEAD OF INSERT Trigger
2. INSTEAD OF UPDATE Trigger
3. INSTEAD OF DELETE Trigger



**Example:** When we insert data into a table, the trigger associated with the insert operation on that table will fire before the data has passed all constraints, such as the primary key constraint. SQL Server also fires the Instead of Trigger if the data insertion fails.

**The following is an illustration of the Instead of Triggers syntax in SQL Server:**

1. **CREATE** **TRIGGER** schema\_name.trigger\_name
2. **ON** table\_name
3. **INSTEAD** **OF** {**INSERT** | **UPDATE** | **DELETE**}
4. **AS**
5. **BEGIN**
6. -- trigger statements
7. -- Insert, Update, or Delete commands
8. **END**

Logon Triggers

Logon triggers are fires in response to a **LOGON event**. The LOGON event occurs when a user session is generated with an SQL Server instance, which is made after the authentication process of logging is completed but before establishing a user session. As a result, the SQL Server error log will display all messages created by the trigger, including error messages and the PRINT statement messages. If authentication fails, logon triggers do not execute. These triggers may be used to audit and control server sessions, such as tracking login activity or limiting the number of sessions for a particular login.

How to SHOW Triggers in SQL Server?

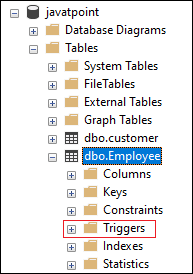
When we have many databases with multiple tables, the show or list trigger comes in handy. When the table names in multiple databases are the same, this query is extremely helpful. Using the following command, we can see a list of all the triggers available in SQL Server:

1. **SELECT** **name**, is\_instead\_of\_trigger
2. **FROM** sys.triggers
3. **WHERE** type = 'TR';

If we are using the **SQL Server Management Studio**, it is very easy to show or list all triggers available in any specific table. We can do this using the following steps:

* Go to the **Databases** menu, select desired database, and then expand it.
* Select the **Tables** menu and expand it.
* Select any specific table and expand it.

We will get various options here. When we choose the **Triggers** option, it displays all the triggers available in this table.

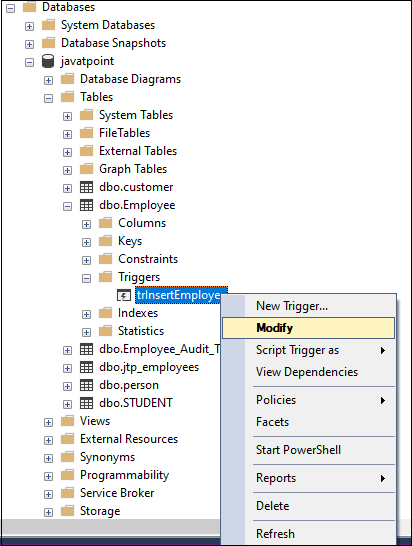


How to UPDATE Triggers in SQL Server?

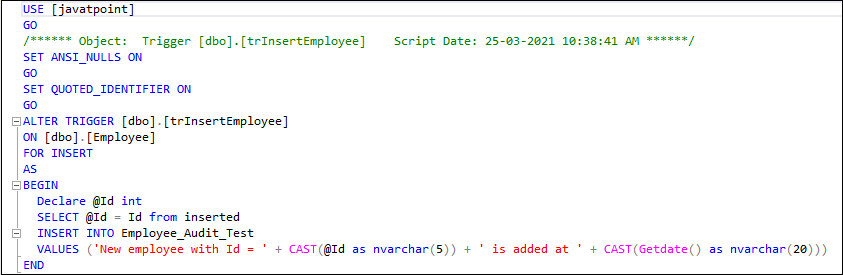
The data stored in the table can be changed over a period of time. In that case, we also need to make changes in the triggers. We can do this in two ways into the SQL Server. The first one is to use the SQL Server Management Studio, and the second one is the Transact-SQL Query.

Modify Triggers using SSMS

First, open the Management Studio to modify the trigger. Next, go to the database and then the table where the trigger is stored. Now, right-click on the trigger that you are going to change or update. It will open the context menu where you will choose **Modify** option:



When you select the Modify option, you will see a new query window with automatically generated ALTER TRIGGER code. We can change it according to our needs.



Modify Triggers using SQL Command

We can use the **ALTER TRIGGER** statement to modify the triggers in MS SQL. The following statement allows us to do modifications to the triggers:

1. **ALTER** **TRIGGER** [dbo].[triggers\_in\_sql]
2. **ON**  [dbo].[EmployeeTable]
3. **AFTER** **INSERT**
4. **AS**
5. **BEGIN**
6. -- Modify as per your needs
7. **END**

How to DELETE Triggers in SQL Server?

We can remove an existing trigger in SQL Server using the **DROP TRIGGER** statement. We must be very careful while removing a trigger from the table. Because once we have deleted the trigger, it cannot be recovered. If a trigger is not found, the DROP TRIGGER statement throws an error.

**The following syntax removes DML triggers:**

1. **DROP** **TRIGGER** [IF EXISTS] schema\_name.trigger\_name;

If we want to remove more than one trigger at once, we must separate the trigger using the comma operator:

1. **DROP** **TRIGGER** schema\_name.trigger\_name1, trigger\_name2.....n;

We can use the DROP TRIGGER statement in the below format to delete one or more LOGON triggers:

1. **DROP** **TRIGGER** [ IF EXISTS ] trigger\_name1, trigger\_name2.....n
2. **ON** { **DATABASE** | ALL SERVER };

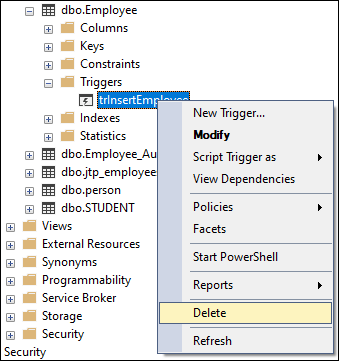
We can use the DROP TRIGGER statement in the below format to delete one or more DDL triggers:

1. **DROP** **TRIGGER** [ IF EXISTS ] trigger\_name1, trigger\_name2.....n
2. **ON** ALL SERVER;

If we are using the SQL Server Management Studio, it makes it very easy to delete triggers from the table. We can do this using the following steps:

* Go to the **Databases** -> **Tables** menu and expand it.
* Select any specific table, expand it, and choose the **Triggers** option

This option displays all the available triggers in this table. Now, right-click on any specific trigger that you want to remove and choose the **Delete** option from the drop-down menu.



Advantages of Triggers

The following are the advantages of using triggers in SQL Server:

* Triggers set database object rules and roll back if any change does not satisfy those rules. The trigger will inspect the data and make changes if necessary.
* Triggers help us to enforce data integrity.
* Triggers help us to validate data before inserted or updated.
* Triggers help us to keep a log of records.
* Triggers increase SQL queries' performance because they do not need to compile each time they are executed.
* Triggers reduce the client-side code that saves time and effort.
* Triggers are easy to maintain.

Disadvantages of Triggers

The following are the disadvantages of using triggers in SQL Server:

* Triggers only allow using extended validations.
* Triggers are invoked automatically, and their execution is invisible to the user. Therefore, it isn't easy to troubleshoot what happens in the database layer.
* Triggers may increase the overhead of the database server.
* We can define the same trigger action for multiple user actions such as INSERT and UPDATE in the same CREATE TRIGGER statement.
* We can create a trigger in the current database only, but it can reference objects outside the current database.