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:

57.4M

1.2K

Features

1. **CREATE TRIGGER schema.trigger\_name**
2. **ON table\_name**
3. **AFTER  {INSERT, UPDATE, DELETE}**
4. **[NOT FOR REPLICATION]**
5. **AS**
6. **{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:

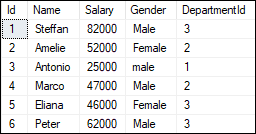
1. **CREATE** **TABLE** Employee
2. (
3. Id **INT** **PRIMARY** **KEY**,
4. **Name** **VARCHAR**(45),
5. Salary **INT**,
6. Gender **VARCHAR**(12),
7. DepartmentId **INT**
8. )

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

1. **INSERT** **INTO** **Employee** **VALUES (1,'Steffan', 82000, 'Male', 3),**
2. **(2,'Amelie', 52000, 'Female', 2),**
3. **(3,'Antonio', 25000, 'male', 1),**
4. **(4,'Marco', 47000, 'Male', 2),**
5. **(5,'Eliana', 46000, 'Female', 3)**

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

1. **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:

1. **CREATE TABLE Employee\_Audit\_Test**
2. **(**
3. **Id int IDENTITY,**
4. **Audit\_Action text**
5. **)**

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:

1. **CREATE TRIGGER trInsertEmployee**
2. **ON Employee**
3. **FOR INSERT**
4. **AS**
5. **BEGIN**
6. **Declare @Id int**
7. **SELECT @Id = Id from inserted**
8. **INSERT INTO Employee\_Audit\_Test**
9. **VALUES ('New employee with Id = ' + CAST(@Id AS VARCHAR(10)) + ' is added at ' + CAST(Getdate() AS VARCHAR(22)))**
10. **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

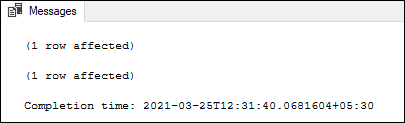
We are going to **create another trigger to store transaction records of each delete operation** on the Employee table into the Employee\_Audit\_Test table. We can create the delete trigger using the below statement:

1. **CREATE TRIGGER trDeleteEmployee**
2. **ON Employee**
3. **FOR DELETE**
4. **AS**
5. **BEGIN**
6. **Declare @Id int**
7. **SELECT @Id = Id from deleted**
8. **INSERT INTO Employee\_Audit\_Test**
9. **VALUES ('An existing employee with Id = ' + CAST(@Id AS VARCHAR(10)) + ' is deleted at ' + CAST(Getdate() AS VARCHAR(22)))**
10. **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

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.