# **Trigger**

* **A trigger is a database object that runs automatically when an event occurs.**

## **In SQL Server, there are 3 types of triggers.**

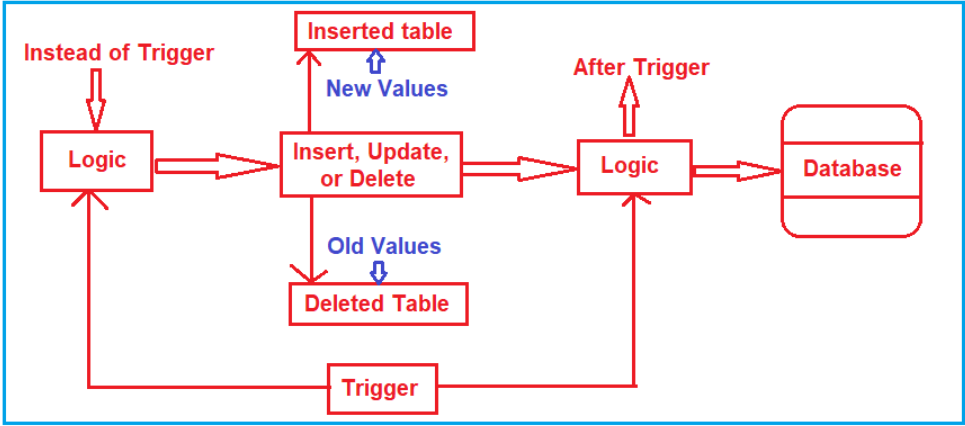
1. **DML Trigger**

* These are fired automatically in response to DML events (Insert, Update, delete)
* DML trigger can be classified into 2 types: ***After Tigger*** and **Instead Of** trigger.
* **After Tigger**, fire after the triggering action. The Insert, update and delete statements, causes an *after trigger* to fire the respective statement complete.
* **Instead Tigger**, fires instead the triggering action. The Insert, update and delete statements, causes an *instead trigger* to fire the respective statement complete.

1. **DDL Trigger**
2. **Logon Trigger**

**Note: We can find newly created trigger. [ Server > Databases > Database Name > Tables > Triggers].**

Whenever you fire any *INSERT, UPDATE, and DELETE* statement on a table, all the new records are actually going to the inserted table i.e. all the updated and new records are present in the inserted table. On the other hand, all the old values are present in the deleted table.



## **After Trigger**

* Sometime called as FOR triggers.
* These kinds of triggers fire after the execution of an action query that can be either DDL statements like Create, Alter and Drop or DML statements like Insert, Update and Delete.
* Whenever you fire any *INSERT, UPDATE, and DELETE* statement on a table, all the new records are actually going to the inserted table i.e. all the updated and new records are present in the ***inserted*** table. On the other hand, all the old values are present in the ***deleted*** table.

## **Syntax for Trigger**

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

**ON** table\_name

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

**[NOT FOR REPLICATION]**

**AS**

**{**sql\_statements**}**

**Syntax:**

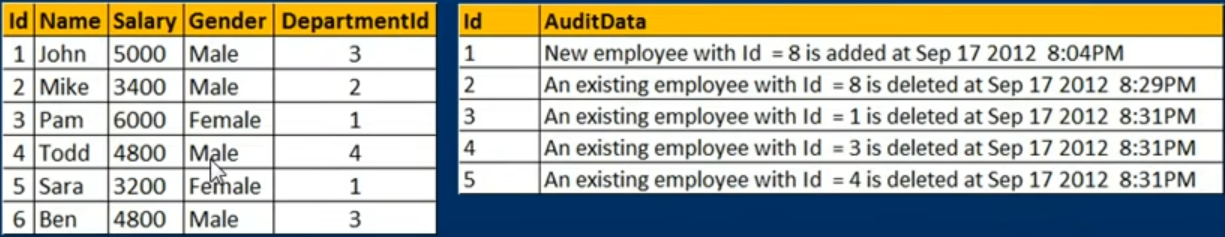
* The ***schema\_name*** is the name of the schema to which the new trigger belongs. The schema name is optional.
* The ***trigger\_name*** is the user-defined name for the new trigger.
* The ***table\_name*** is the table to which the trigger applies.
* The event is listed in the AFTER clause. The event could be INSERT, UPDATE, or **DELETE**. A single trigger can fire in response to one or more actions against the table.
* The **NOT FOR REPLICATION** option instructs SQL Server not to fire the trigger when data modification is made as part of a replication process.
* The ***sql\_statements*** is one or more Transact-SQL used to carry out actions once an event occurs.

## **The content of the INSERTED and DELETED tables (Virtual Table) before and after each event:**

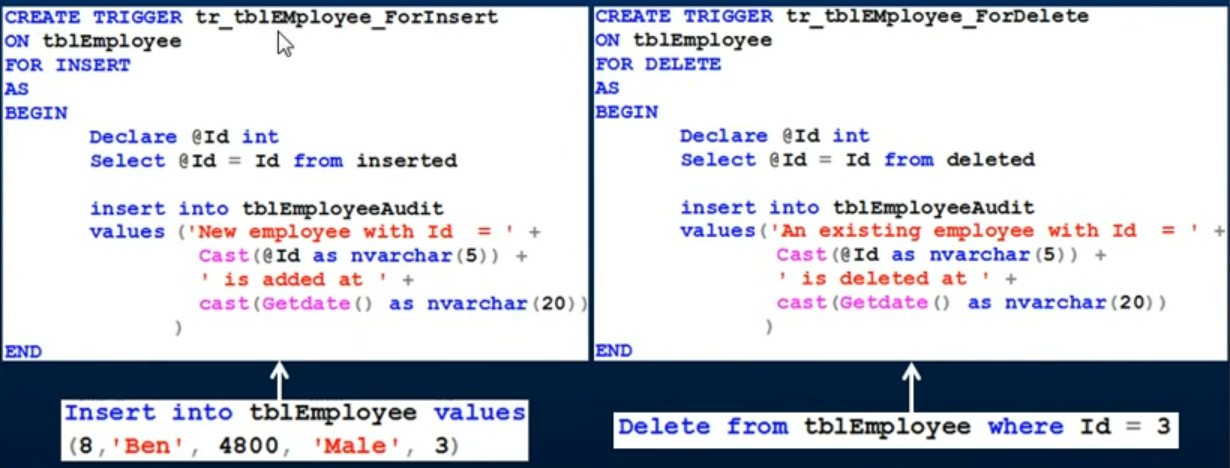
|  |  |  |
| --- | --- | --- |
| **DML event** | **INSERTED table holds** | **DELETED table holds** |
| INSERT | rows to be inserted | empty |
| UPDATE | new rows modified by the update | existing rows modified by the update |
| DELETE | empty | rows to be deleted |

### **Practical**

1. **Suppose, we add/remove a new employee in Employee tables, we want to some audit information to be captured automatically in Audit Table.**



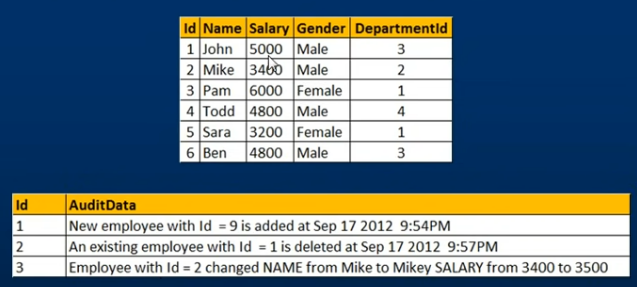
1. **We want to capture employee id with date and time of data inserted or deleted.**

****

* Naming convention to use at starting ‘tr’.

## **After Update Trigger**

**SQL Server provides two virtual tables that are available specifically for triggers called *INSERTED* and *DELETED* tables.**

**SQL Server uses these tables to capture the data of the modified row before and after the event occurs.**

* The After trigger for UPDATE event, makes use of both inserted and deleted tables. The inserted table contains the updated data and the deleted table contains the old data.

### **Practical**

# **Instead of Trigger**

* An ***INSTEAD OF*** trigger is a trigger that allows you to skip an *INSERT*, *DELETE*, or *UPDATE* statement to a table or a view and execute other statements defined in the trigger instead.
* The actual insert, delete, or update operation does not occur at all.
* In other words, an INSTEAD OF trigger skips a DML statement and execute other statements.
* **Instead of Triggers usually used to update views correctly that are based on multiple tables.**

## **Syntax of Instead Trigger**

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

**ON {**table\_name | view\_**name }**

**INSTEAD OF {[*INSERT*] [,] [*UPDATE*] [,] [*DELETE*] }**

**AS**

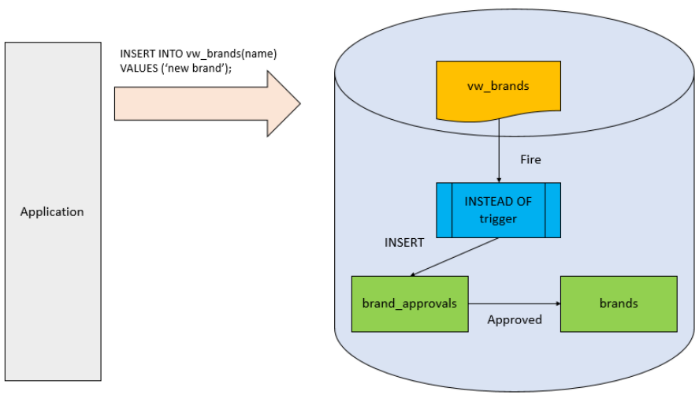
**{**sql\_statements**}**

In this syntax:

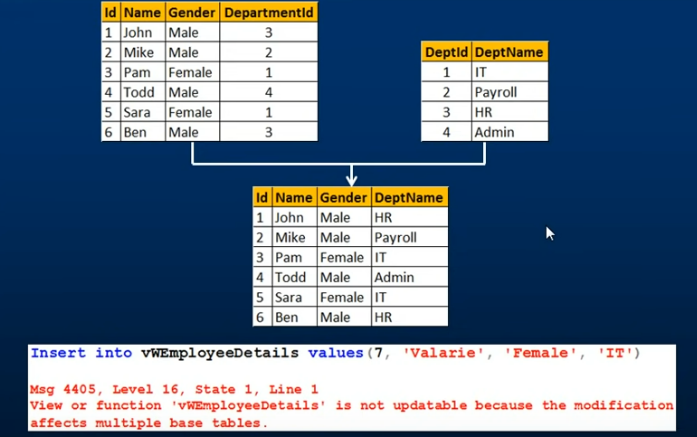
* First, specify the name of the trigger and optionally the name of the schema to which the trigger belongs in the **CREATE TRIGGER** clause.
* Second, specify the name of the table or view which the trigger associated with.
* Third, specify an event such as ***INSERT, DELETE, or UPDATE*** which the trigger will fire in the **INSTEAD OF** clause. The trigger may be called to respond to one or multiple events.
* Fourth, place the trigger body after the **AS** keyword. A trigger’s body may consist of one or more Transact-SQL statements.

**Example**

Suppose, an application needs to insert new brands into the ***production.brands*** table. However, the new brands should be stored in another table called ***production.brand\_approvals*** for approval before inserting into the ***production.brands*** table.

To accomplish this, you create a view called ***production.vw\_brands*** for the application to insert new brands. If brands are inserted into the view, an INSTEAD OF trigger will be fired to insert brands into the ***production.brand\_approvals*** table.

## **Practical**



# **Instead Of Insert**

* An ***INSTEAD OF*** trigger is a trigger that allows you to skip an [INSERT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/), [DELETE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-delete/), or [UPDATE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-update/) statement to a table or a view and execute other statements defined in the trigger instead.
* The actual insert, delete, or update operation does not occur at all.
* In other words, an INSTEAD OF trigger skips a DML statement and execute other statements.

## **Syntax**

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

**ON** *{table\_name | view\_name }*

**INSTEAD OF** *{[INSERT] [,] [UPDATE] [,] [DELETE] }*

**AS**

**BEGIN**

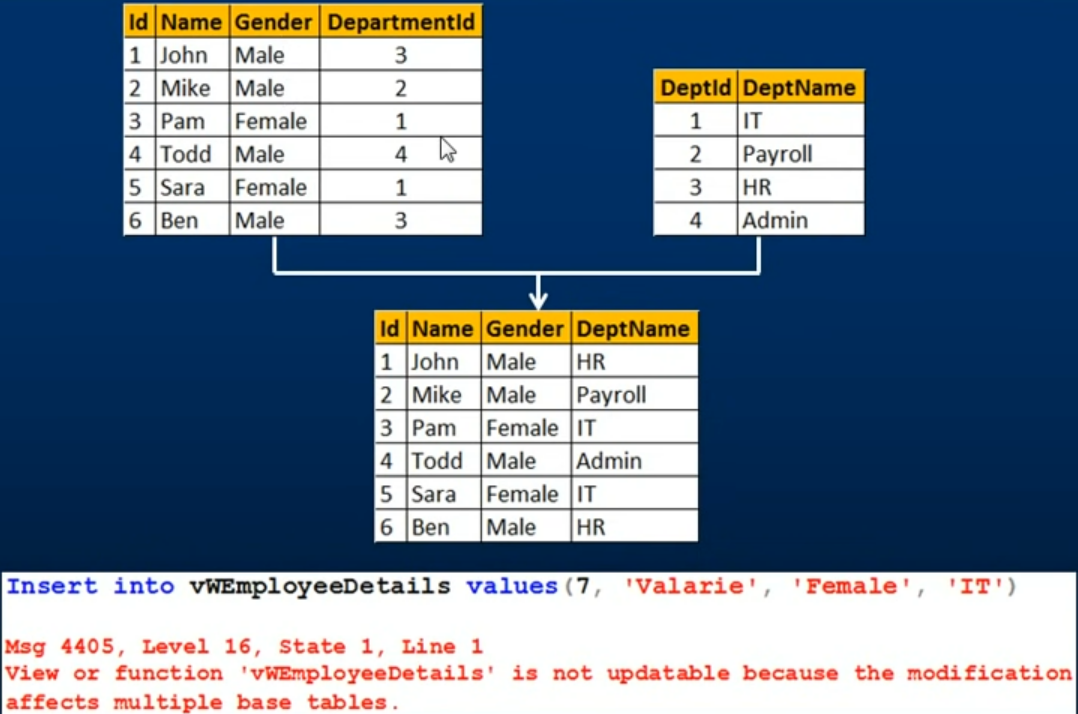
*{sql\_statements}*

END

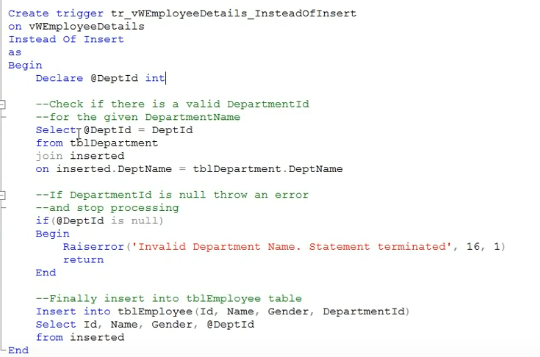
EN

**In this syntax:**

* First, specify the name of the trigger and optionally the name of the schema to which the trigger belongs in the CREATE TRIGGER clause.
* Second, specify the name of the table or view which the trigger associated with.
* Third, specify an event such as INSERT, DELETE, or UPDATE which the trigger will fire in the INSTEAD OF clause. The trigger may be called to respond to one or multiple events.
* Fourth, place the trigger body after the AS keyword. A trigger’s body may consist of one or more Transact-SQL statements.



* We have two tables here, Tbl\_Employees and tbl\_Department.
* Now we create a view based on these two tables so we want Id, Name, Gender from tbl\_employee table and DepartName from Tbl\_department table.
* So, to create a view we need to join these two table, So, this view is based on multiple base tables.
* as we know View is a virtual table meaning it is nothing more than stored SQL Query, it doesn’t really contain any data.
* The view actually gets its data from its underlying based tables.
* Now, when we try to insert a row into this View, behind the scenes insert that row into these base tables.
* SQL server has confusion which column should go into which table or base table should be this row should be inserted because we already have a row with ID, so should another row should be inserted.
* That why, if insert statement affecting multiple base tables, SQL server will throw an error stating so.
* **Instead of triggers are used to update/insert in VIEW correctly that are based on multiple base tables.**

****