# **Trigger**

## **Introduction**

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

**These are logic’s like stored procedures that can be executed automatically before the Insert, Update or Delete happens in a table or after the Insert, Update, or Delete happens in a table.**

* In simple words, we can say that, if you want to execute some ***pre-processing or post-processing logic*** before or after the *Insert, Update, or Delete* in a table then you need to use Triggers in SQL Server.

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

1. **DML Trigger – Data Manipulation Language Triggers.**

* 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 Trigger***, fire after the triggering action. The Insert, update and delete statements, causes an after trigger to fire the respective statement complete.
* ***Instead Trigger***, fires instead the triggering action. The Insert, update and delete statements, causes an instead trigger to fire the respective statement complete.

1. **DDL Trigger -- Data Definition Language Triggers**

* The DDL triggers in SQL Server are fired in response to a variety of data definition language (DDL) events such as Create, Alter, Drop, Grant, Deny, and Revoke (Table, Function, Index, Stored Procedure, etc…).
* That means DDL triggers in SQL Server are working on a database.

1. **Logon Trigger**

* The Logon Triggers in SQL Server are the special kind of stored procedure or we can also say a special type of operation which fire or executed automatically in response to a LOGON event and moreover, we can define more than one Logon trigger on the server.
* The Logon triggers are fired only after the successful authentication but before the user session is actually established.
* If the authentication is failed then the logon triggers will not be fired.

1. **CLR Trigger – Common Language Runtime Triggers**

* We can create a database object inside SQL Server that is programmed in an assembly created in the Microsoft .NET Framework common language runtime (CLR).
* Database objects that can leverage the rich programming model provided by the CLR include DML triggers, DDL triggers, stored procedures, functions, aggregate functions, and types.

**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.

**Triggers are used for several purposes:**

1. **Produce additional checking during insert, update or delete operations on the affected table.**
2. **They allow us to encode complex default values that cannot be handled by default constraints.**
3. **Implement referential integrity across databases.**

* You can read more about this in this tip: SQL Server Referential Integrity across Databases Using Triggers.

1. **They allow us to control what actually happens when one performs an insert, update, or delete on a view that accesses multiple tables.**
2. **You can calculate aggregated columns in a table using triggers.**
3. **Log table modifications.**

* Some tables have sensitive data such as customer email, employee salary, etc., that you want to log all the changes.
* In this case, you can create the UPDATE trigger to insert the changes into a separate table.

1. **Enforce complex integrity of data.**

* In this scenario, you may define triggers to validate the data and reformat the data if necessary.
* For example, you can transform the data before insert or update using a BEFORE INSERT or BEFORE UPDATE trigger.

# **DML Trigger**

**This type of trigger is the most known and used by developers.**

**DML stands for Data Manipulation Language and refers to the SQL instructions that changes data**.

* Those instructions are INSERT, UPDATE and DELETE.
* Basically, DML triggers can be defined as pieces of code written mostly in Transact SQL language whose execution is not performed manually by the user and instead is run automatically in response to DML events.

**DML Triggers are associated to a table or view and to any of the DML events (*INSERT, UPDATE and DELETE*).**

**Something to remark is that a trigger can only be associated with one single table or view, but can be associated to more than one DML event.**

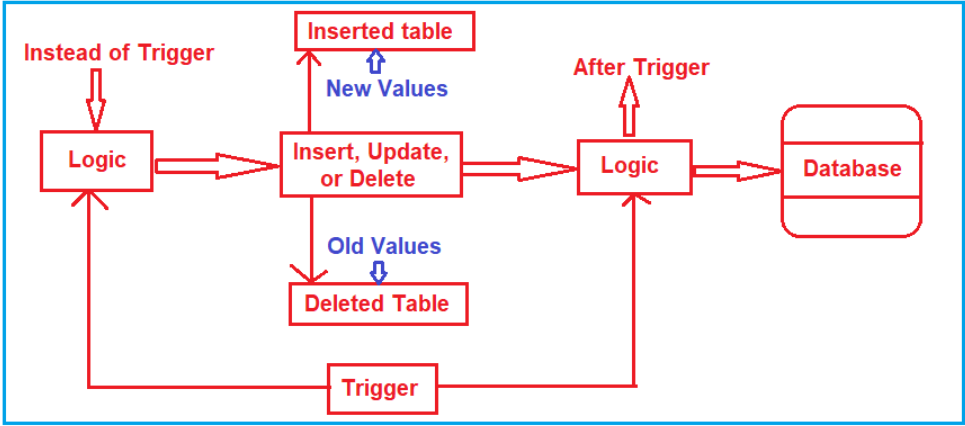
* For instance, you could have a trigger that is associated with a table that fires in response to INSERT and UPDATE events.

**There are two types of triggers. They are as follows:**

1. ***Instead of Triggers***: The Instead Of triggers are going to be executed instead of the corresponding DML operations.

* That means *instead of the DML operations such as Insert, Update, and Delete,* the Instead Of triggers are going to be executed.

1. ***After Triggers***: The After Triggers fires in SQL Server execute after the triggering action.

* ****That means once the DML statement (such as Insert, Update, and Delete) completes its execution, this trigger is going to be fired.

## **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:**

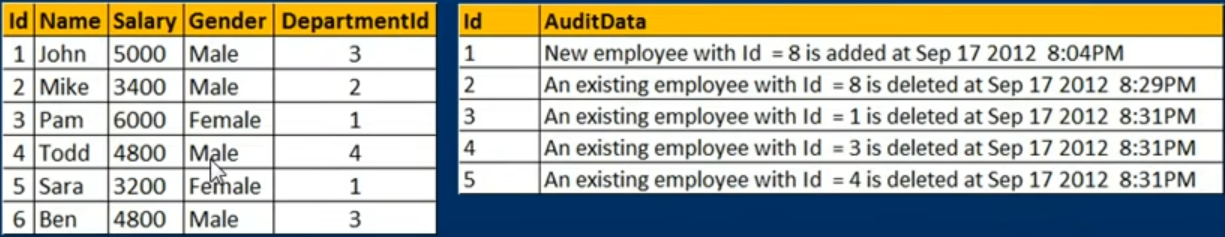
* The ***schema\_name*** is the name of the schema to which the new trigger belongs. The schema name is optional.
* The ***trigger\_name*** is the user-defined name for the new trigger.
* The ***table\_name*** is the table to which the trigger applies.
* The event is listed in the AFTER clause. The event 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:**

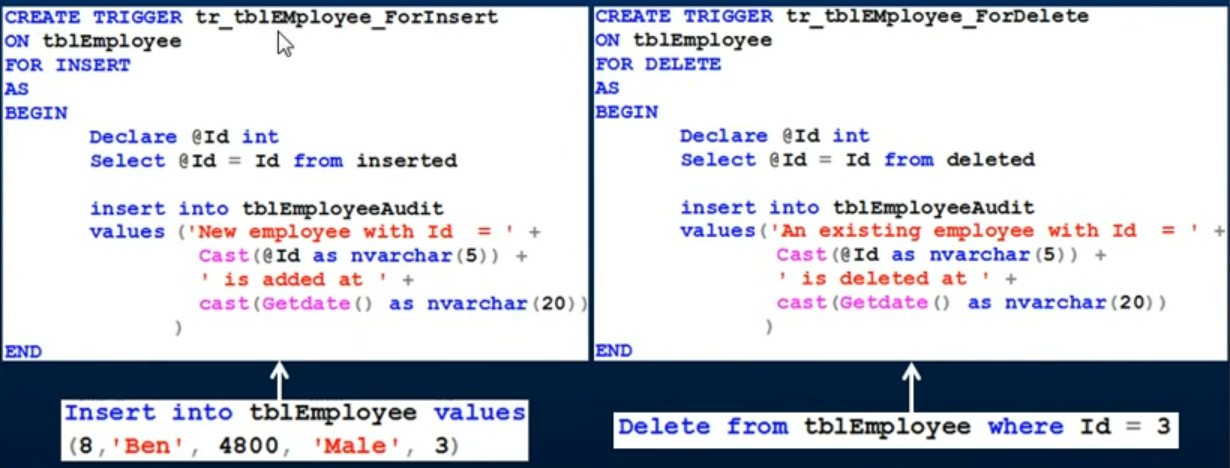
|  |  |  |
| --- | --- | --- |
| DML event | INSERTED table holds | DELETED table holds |
| INSERT | rows to be inserted | empty |
| UPDATE | new rows modified by the update | existing rows modified by the update |
| DELETE | empty | rows to be deleted |

### **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’***.
* We create a trigger for a table and for a specific event (like insert, etc).
* For example, we want to create a trigger on this table ***tbl\_employee*** for insert event.
* Because as soon as a new row is inserted into ***tbl\_employee*** we want to log the audit information.

**I want to capture the information and we can also make use of store procedures to do this task.**

* However, the easiest way to achieve that by associate an ***After Update trigger*** with the ***tbl\_Employee*** as soon as the insert statement is fired.
* I want this after insert trigger to be fired which will capture the ID of the record that you have inserted into the table and then the date and time, and insert another row into this ***TBL employee audit table.***

-- Create After trigger to insert

alter trigger tr\_tbl\_EmployeeDetails\_ForInsert

on tblEmployeeDetails

for insert

as

BEGIN

-- check id from inserted table and store in id variable

Declare @Id int

select @Id=id from inserted

-- insert the record into audit table

insert into tbl\_employeeAuditTable values ( 'New Employee with id: ' +

CAST(@Id as nvarchar(5)) +

' is added at ' +

cast(getdate() as nvarchar(max))

)

END

-- insert data in employee table and check audit table

insert into tblEmployeeDetails (Name,salary,Gender,DepartmentId)values ('Riya',6000,'Female',1)

Select \* from tbl\_employeeAuditTable

go

--Prerequisite

create database Test

go

use test

Create TABLE tblEmployeeDetails ( Id int IDENTITY(1,1) primary key ,

[Name] nvarchar(20),

salary int,

Gender nvarchar(10),

DepartmentId int

)

-- drop table tblEmployeeDetails

-- ALTER TABLE tblEmployeeDetails ADD NewColumn INT IDENTITY(1,1)

--insert into tblEmployeeDetails (Name,salary,Gender,DepartmentId)values ('shubham',5000,'Male',3),

('shivam',null,'Male',2),

('shivani',4000,'Female',3),

('Nikhil',5100,'Male',4),

('Hemant',3500,'Male',1),

('Rahul',3500,'Male',3)

select \* from test..tblEmployeeDetails

Create table tbl\_employeeAuditTable ( id int primary key identity(1,1),

AuditData nvarchar(max)

)

Select \* from tbl\_employeeAuditTable

go

-- #2. Create After Delete trigger

alter trigger tr\_tbl\_EmployeeDetails\_ForDelete

on tblEmployeeDetails

for delete

as

BEGIN

-- check id from inserted table and store in id variable

Declare @Id int

select @Id=id from deleted

-- insert the record into audit table

insert into tbl\_employeeAuditTable values ( 'New Employee with id: ' +

CAST(@Id as nvarchar(5)) +

' is removed at ' +

cast(getdate() as nvarchar(max))

)

END

-- delete data in employee table and check audit table

delete from tblEmployeeDetails where id=6

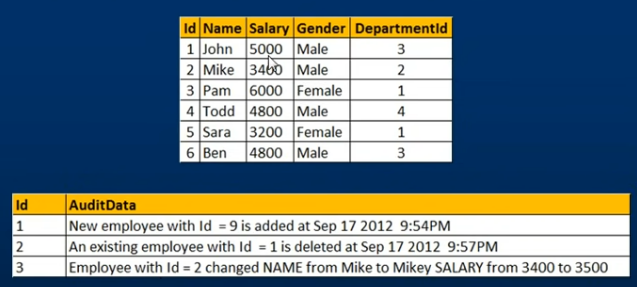
select \* from test..tblEmployeeDetails

Select \* from tbl\_employeeAuditTable

go

## **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

--Prerequisite

create database Test

go

use test

Create TABLE tblEmployeeDetails ( Id int IDENTITY(1,1) primary key ,

[Name] nvarchar(20),

salary int,

Gender nvarchar(10),

DepartmentId int

)

-- drop table tblEmployeeDetails

-- ALTER TABLE tblEmployeeDetails ADD NewColumn INT IDENTITY(1,1)

--insert into tblEmployeeDetails (Name,salary,Gender,DepartmentId)values ('shubham',5000,'Male',3),

('shivam',null,'Male',2),

('shivani',4000,'Female',3),

('Nikhil',5100,'Male',4),

('Hemant',3500,'Male',1),

('Rahul',3500,'Male',3)

select \* from test..tblEmployeeDetails

Create table tbl\_employeeAuditTable ( id int primary key identity(1,1),

AuditData nvarchar(max)

)

Select \* from tbl\_employeeAuditTable

go

-- Create/Alter a update trigger

Create trigger tr\_tblEmployee\_ForUpdate

on tblEmployeeDetails

for update

as

Begin

-- declare varable to store old and new data

Declare @Id int

Declare @OldName nvarchar(20), @NewName nvarchar(20)

Declare @OldSalary nvarchar(20), @NewSalary nvarchar(20)

Declare @OldGender nvarchar(20), @NewGender nvarchar(20)

Declare @OldDeptId nvarchar(20), @NewDeptId nvarchar(20)

-- create a string for Audit Table dynamically

Declare @AuditString nvarchar(1000)

-- insert row from inserted virtual table to Temp table

Select \*

into #TempTable

from inserted

-- while loop is used in case if user update multiple record at a time

-- if select query returns no record, exists functiion return false and while loop get skiped.

While(Exists(Select Id from #TempTable))

Begin

set @AuditString = ''

-- set variable for inserted row form temp table

Select top 1 @Id=Id, @NewName = Name,

@NewGender = Gender, @NewSalary = Salary,

@NewDeptId = DepartmentId

from #TempTable

-- fetch record based on @id variable from deleted and set old data into variables

Select @OldName = Name,

@OldGender = Gender, @OldSalary = Salary,

@OldDeptid = DepartmentId

from deleted where id = @id

Set @AuditString = 'Employee with Id = ' + Cast(@id as nvarchar(4)) + ' changed '

if(@OldName <> @NewName)

set @AuditString = @AuditString + ' Name From ' + @OldName + ' to ' + @NewName

if(@OldSalary <> @NewSalary)

set @AuditString = @AuditString + ' Salary From ' + @OldSalary + ' to ' + @NewSalary

if(@OldGender <> @NewGender)

set @AuditString = @AuditString + ' Gender From ' + @OldGender + ' to ' + @NewGender

if(@OldDeptId <> @NewDeptId)

set @AuditString = @AuditString + ' DeptId From ' + @OldDeptId + ' to ' + @NewDeptId

-- insert the @AuditString into Audit Table

insert into tbl\_employeeAuditTable (AuditData) values (@AuditString)

-- delete the variable @id record from temp data, to prevent infinite loop and in case of multiple update scenario.

delete from #TempTable where Id = @Id

End

End

go

-- update and check the audit data

Update tblEmployeeDetails set Name='Ramesh', salary=5500, Gender='Male' where Id=5

Select \* from tbl\_employeeAuditTable

-- drop trigger tr\_tblEmployee\_ForUpdate

## **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**

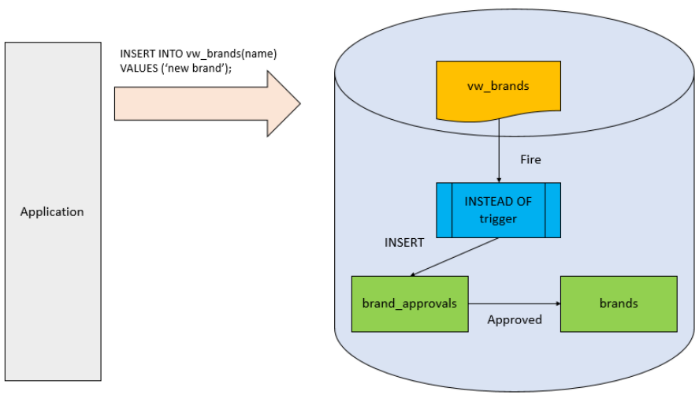
**{**sql\_statements**}**

**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**

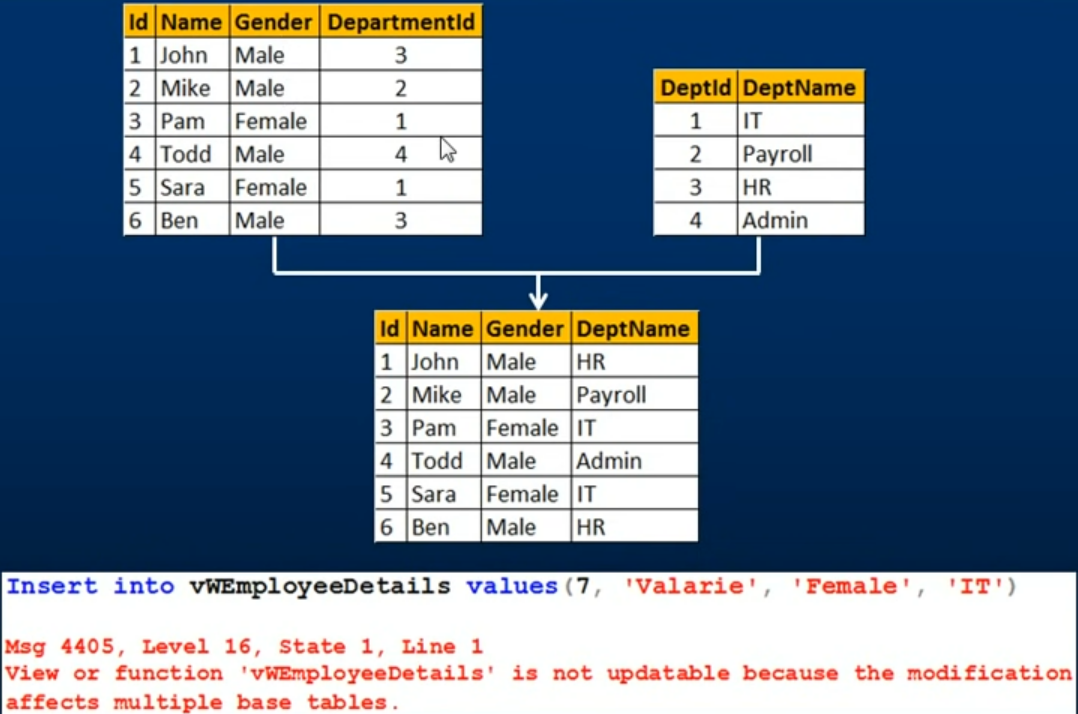
*{sql\_statements}*

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.
* This view is based on multiple tables and your insert statement into this view will affect both these tables.
* If we look at the error message view of function view employee details is not updatable because the modification affects multiple base tables.

**Instead of triggers are used to update/insert in VIEW correctly that are based on multiple base tables.**

--Prerequisite

create database Test

go

use test

go

-- create Employee Table

Create TABLE tblEmployeeDetails ( Id int primary key ,

[Name] nvarchar(20),

salary int,

Gender nvarchar(10),

DepartmentId int

)

--insert into tblEmployeeDetails (Id,Name,salary,Gender,DepartmentId)values (1,'shubham',5000,'Male',3),

(2,'shivam',1000,'Male',2),

(3,'shivani',4000,'Female',3),

(4,'Nikhil',5100,'Male',4),

(5,'Hemant',3500,'Male',1),

(6, 'Rahul',3500,'Male',3)

truncate table tblEmployeeDetails

-- create department table

Create TABLE tblDepartment( DeptId int IDENTITY(1,1) primary key ,

DeptName nvarchar(50)

)

insert into tblDepartment(DeptName) values ('IT'),('Payroll'),('HR'),('Admin')

-- create a view

Create view vw\_EmployeeDetails

AS

Select Id,Name, Gender, de.DeptName

from tblEmployeeDetails em inner join tblDepartment de on em.DepartmentId = de.DeptId

go

-- select view

select \* from vw\_EmployeeDetails

-- insert a row in view

Insert into vw\_EmployeeDetails (Id,Name, Gender, DeptName)values ((select max(id) + 1 from vw\_EmployeeDetails),'Vari','Female', 'IT')

Insert into vw\_EmployeeDetails (Id,Name, Gender, DeptName)values (50,'Vari','Female', 'I')

-- Error : View or function 'vw\_EmployeeDetails' is not updatable because the modification affects multiple base tables.

go

-- create instead trigger for insert

Create trigger tr\_vwEmployeeDetails\_InsteadofInsert

on vw\_EmployeeDetails

Instead of Insert

AS

Begin

-- create a variable which hold the deptid

Declare @DeptId int

--Check if there is a valid DepartmentId

--for the given DepartmentName

Select @DeptId = DeptId

from tblDepartment

join inserted

on inserted.DeptName = tblDepartment.DeptName

--If DepartmentId is null throw an error

--and stop processing

if (@DeptId is null)

Begin

Raiserror('Invalid Department Name. Statement terminated', 16, 1)

return

End

--Finally insert into tblEmployee table

Insert into tblEmployeeDetails(Id, Name, Gender, DepartmentId)

Select Id, Name, Gender, @DeptId

from inserted

end