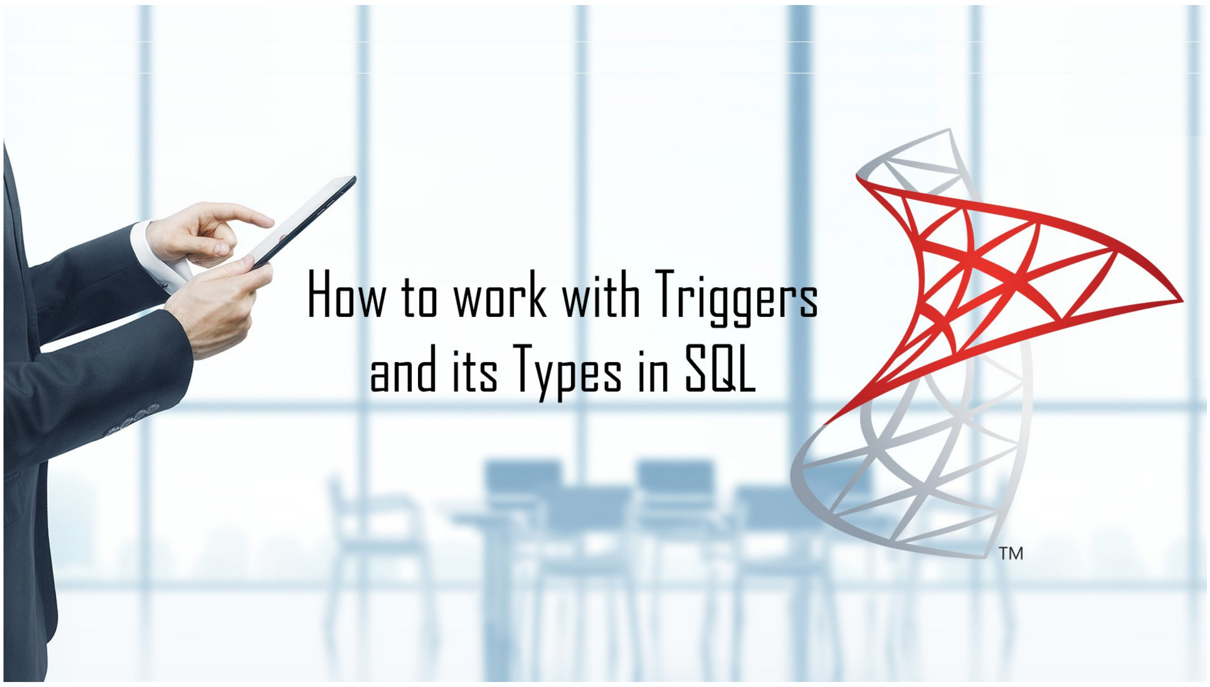
HOW TO WORK WITH TRIGGERS AND ITS TYPES IN SQL?



TRIGGERS

A trigger is a special method of stored procedure and it invokes automatically when an event starts in the database server. DML triggers execute 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.

TYPES OF TRIGGERS

Triggers contain three types as follows;

•DML Triggers
•DDL Triggers
•Logon Triggers

DML TRIGGERS

DML stands for Data Manipulation Language. INSERT, UPDATE, and DELETE statements are DML statements. DML triggers get fired whenever data is modified using INSERT, UPDATE, and DELETE events.

DML triggers can be again classified into 2 types.

1. After triggers (Sometimes called FOR triggers)

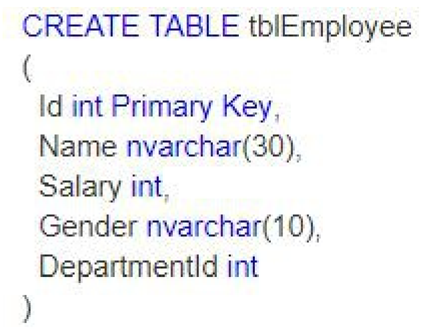
2. Instead of triggers

AFTER TRIGGERS

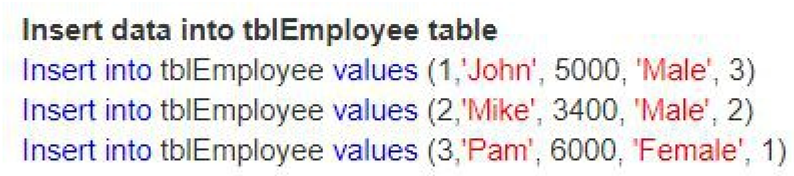
After triggers get fired after only with a condition when a modification action occurs. The INSERT, UPDATE and DELETE commands because of an after trigger gets fired after the execution of a complete statement.

Therefore, we need to use tblEmployee and tblEmployeeAudit tables for further examples as follows;

SQL Script to create tblEmployee table:

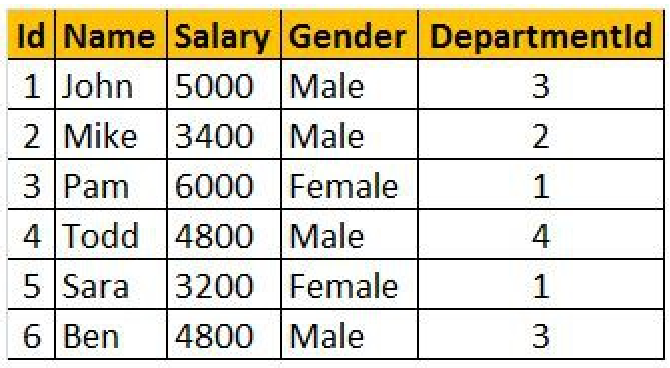


```
CREATE TABLE tblEmployee
(
  Id int Primary Key,
  Name nvarchar(30),
  Salary int,
  Gender nvarchar(10),
  DepartmentId int
)
```



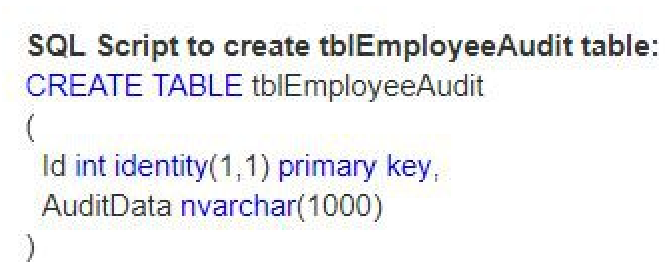
```
Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 5000, 'Male', 3)
Insert into tblEmployee values (2,'Mike', 3400, 'Male', 2)
Insert into tblEmployee values (3,'Pam', 6000, 'Female', 1)
```

Finally, the table will look like as below;



Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mike	3400	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

SQL Script to create tblEmployeeAudit table:



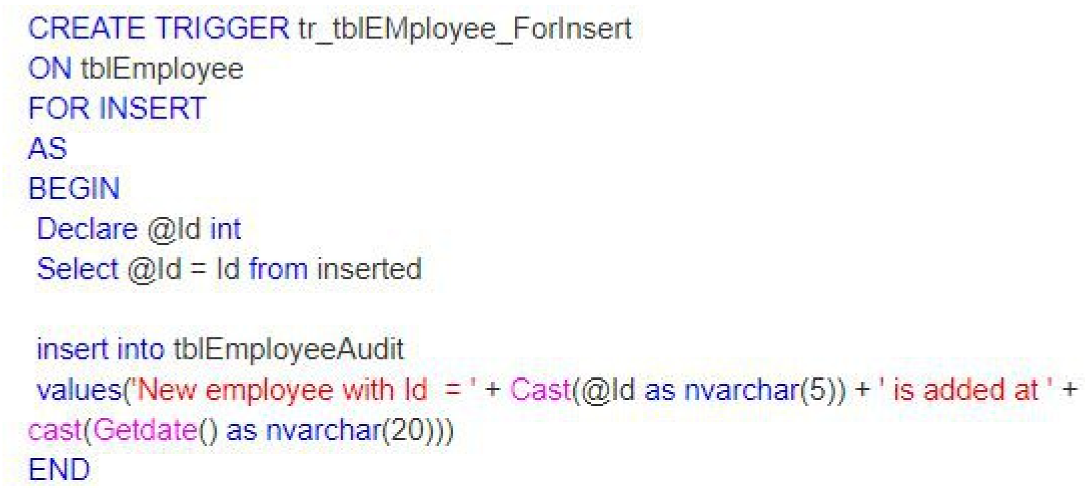
SQL Script to create tblEmployeeAudit table:

```
CREATE TABLE tblEmployeeAudit
(
 Id int identity(1,1) primary key,
 AuditData nvarchar(1000)
)
```

Once a new Employee is added to the table in the database. Now, I want to retrieve the ID, date and time, the new employee is added to the tblEmployeeAudit table. The easiest way to get the same result by using an AFTER TRIGGER for INSERT event.

AFTER TRIGGER FOR INSERTION

Example for AFTER TRIGGER for INSERT event on the tblEmployee table as follows;



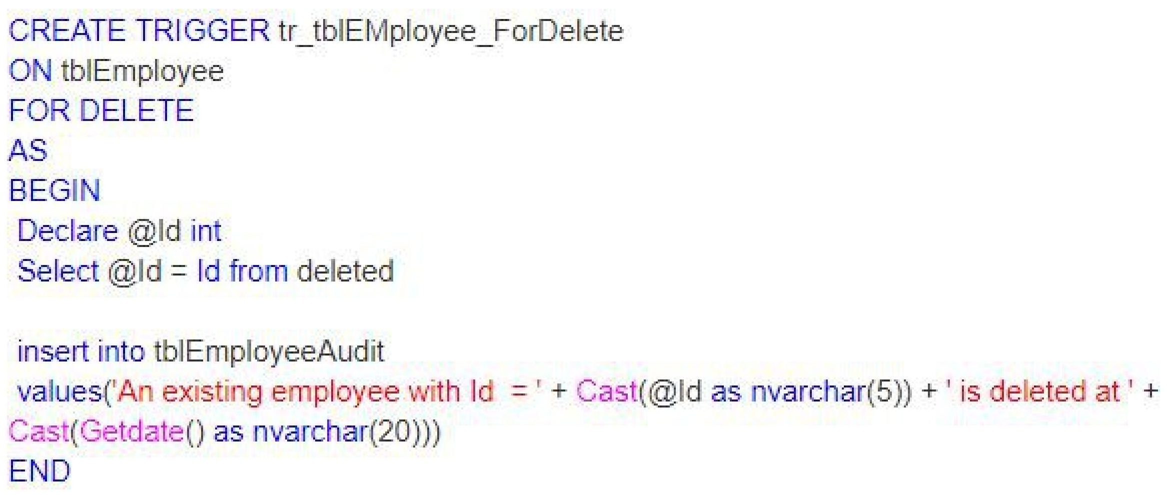
```
CREATE TRIGGER tr_tblEMployee_ForInsert
ON tblEmployee
FOR INSERT
AS
BEGIN
 Declare @Id int
 Select @Id = Id from inserted

 insert into tblEmployeeAudit
 values('New employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is added at ' +
cast(Getdate() as nvarchar(20)))
END
```

After this, we will get the id from inserted table name. So, the question arises, what is the role of an inserted table? An Inserted table defines a table which is mainly used by DML triggers. Once a time, you add a new row to the tblEmployee table, similarly, a row will generate the same copy in the inserted table, where only a trigger can access through the function. Further, you will not be able to access this table outside the context of the trigger function. The structure of the inserted table must be identical to the structure of the tblEmployee table.

So, if we execute the following INSERT command on tblEmployee. Simultaneously, after inserting the row data into the tblEmployee table, then the trigger gets fired as it gets invoked automatically, and the same row inserted into the tblEmployeeAudit table.

Insert into tblEmployee values (7,’Tan’, 2300, ‘Female’, 3)

when a row deletes records from the table tblEmployee.
Example for AFTER TRIGGER for DELETE event a tblEmployee table:

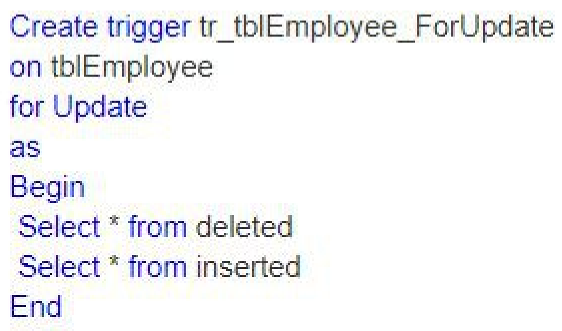


```
CREATE TRIGGER tr_tblEMployee_ForDelete
ON tblEmployee
FOR DELETE
AS
BEGIN
 Declare @Id int
 Select @Id = Id from deleted

 insert into tblEmployeeAudit
 values('An existing employee with Id  = ' + Cast(@Id as nvarchar(5)) + ' is deleted at ' +
Cast(Getdate() as nvarchar(20)))
END
```

AFTER UPDATE TRIGGER

Triggers work with two organized tables, INSERTED and DELETED. Newly updated data stored in the inserted table and old specific data stored in the deleted table. After triggering for UPDATE event makes use of both inserted and deleted tables.

Create AFTER UPDATE trigger script:

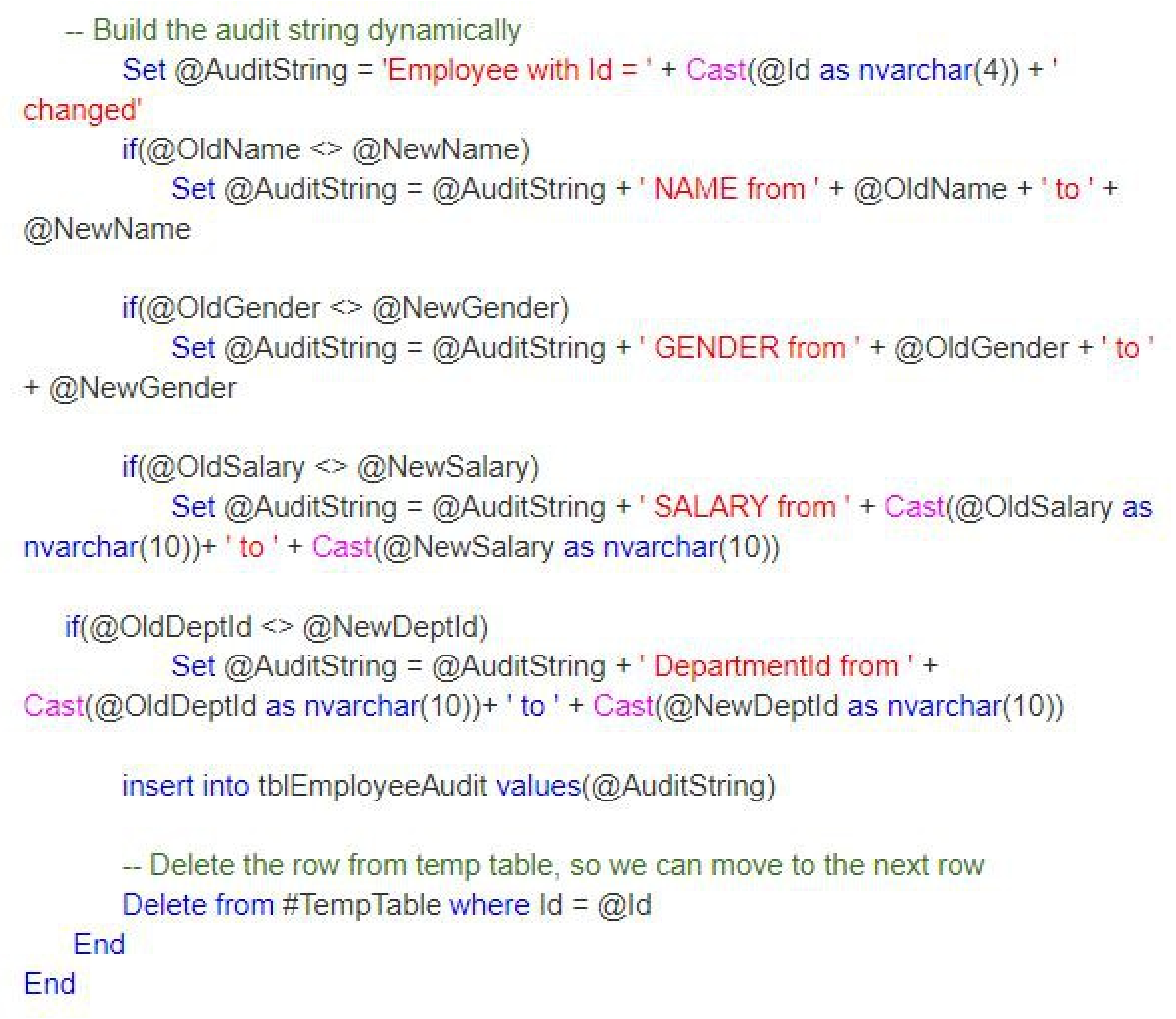


```
Create trigger tr_tblEmployee_ForUpdate
on tblEmployee
for Update
as
Begin
 Select * from deleted
 Select * from inserted
End
```

So, execute the query:
Update tblEmployee set Name = ‘Tods’, Salary = 2000, Gender
= ‘Female’ where Id = 4

Furthermore, after the UPDATE command execution, the AFTER UPDATE trigger gets fired, and you see the contents of INSERTED and DELETED tables and stores the audit data in tblEmployeeAudit table.

```
Alter trigger tr_tblEmployee_ForUpdate
on tblEmployee
for Update
as
Begin
    -- Declare variables to hold old and updated data
    Declare @Id int
    Declare @OldName nvarchar(20), @NewName nvarchar(20)
    Declare @OldSalary int, @NewSalary int
    Declare @OldGender nvarchar(20), @NewGender nvarchar(20)
    Declare @OldDeptId int, @NewDeptId int

    -- Variable to build the audit string
    Declare @AuditString nvarchar(1000)

    -- Load the updated records into temporary table
    Select *
    into #TempTable
    from inserted

    -- Loop thru the records in temp table
    While(Exists(Select Id from #TempTable))
    Begin
        --Initialize the audit string to empty string
        Set @AuditString = ''


     -- Select first row data from temp table
     Select Top 1 @Id = Id, @NewName = Name,
     @NewGender = Gender, @NewSalary = Salary,
     @NewDeptId = DepartmentId
     from #TempTable

     -- Select the corresponding row from deleted table
     Select @OldName = Name, @OldGender = Gender,
     @OldSalary = Salary, @OldDeptId = DepartmentId
     from deleted where Id = @Id
```



```
    -- Build the audit string dynamically
        Set @AuditString = 'Employee with Id = ' + Cast(@Id as nvarchar(4)) + '
changed'
        if(@OldName <> @NewName)
            Set @AuditString = @AuditString + ' NAME from ' + @OldName + ' to ' +
@NewName

        if(@OldGender <> @NewGender)
            Set @AuditString = @AuditString + ' GENDER from ' + @OldGender + ' to '
+ @NewGender

        if(@OldSalary <> @NewSalary)
            Set @AuditString = @AuditString + ' SALARY from ' + Cast(@OldSalary as
nvarchar(10))+ ' to ' + Cast(@NewSalary as nvarchar(10))

    if(@OldDeptId <> @NewDeptId)
            Set @AuditString = @AuditString + ' DepartmentId from ' +
Cast(@OldDeptId as nvarchar(10))+ ' to ' + Cast(@NewDeptId as nvarchar(10))

        insert into tblEmployeeAudit values(@AuditString)

        -- Delete the row from temp table, so we can move to the next row
        Delete from #TempTable where Id = @Id
    End
End
```

INSTEAD OF INSERT TRIGGER

As we well know that AFTER triggers get fired after the triggering action (INSERT, UPDATE or DELETE events), whereas, INSTEAD OF triggers get fired instead of the triggering action (INSERT, UPDATE or DELETE events). In addition, INSTEAD OF Insert triggers makes use for correctly update views that are based on multiple tables.

So, we start with the base demo on Employee and Department tables. So, first, let's create these 2 tables.

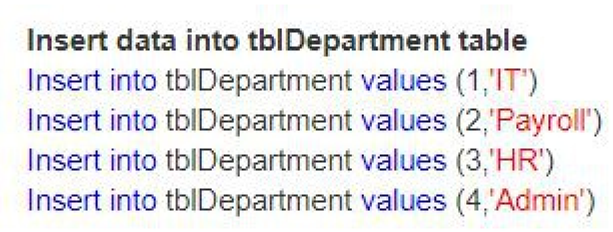
SQL SCRIPT TO CREATE TBLEMPLOYEE TABLE:

```
CREATE TABLE tblEmployee
(
Id int Primary Key,
Name nvarchar(30),
Gender nvarchar(10),
```

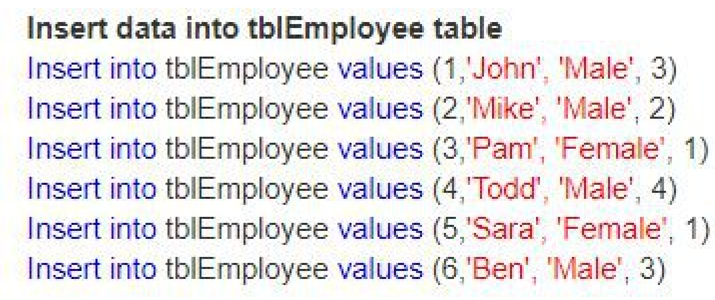
DepartmentId int
)

SQL SCRIPT TO CREATE TBLDEPARTMENT TABLE:

CREATE TABLE tblDepartment
(
DeptId int Primary Key,
DeptName nvarchar(20)
)



```
Insert data into tblDepartment table
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')
```



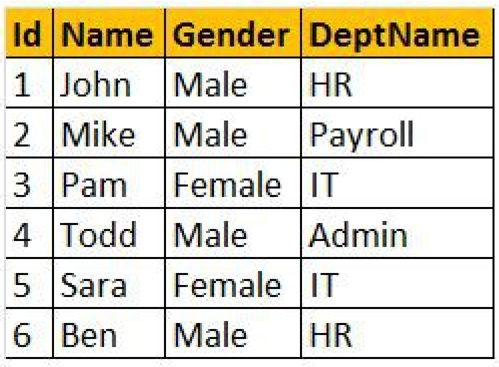
```
Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

So, we have our two required tables, let’s create a view which is based on these two tables, it will fetch the records of Employee Id, Name, Gender and DepartmentName columns. Therefore, the view is based on multiple tables.

SCRIPT TO CREATE A VIEW:

Create view vWEmployeeDetails
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId

Once you execute this line, Select * from vWEmployeeDetails, It will retrieve all the records from the table as follows;



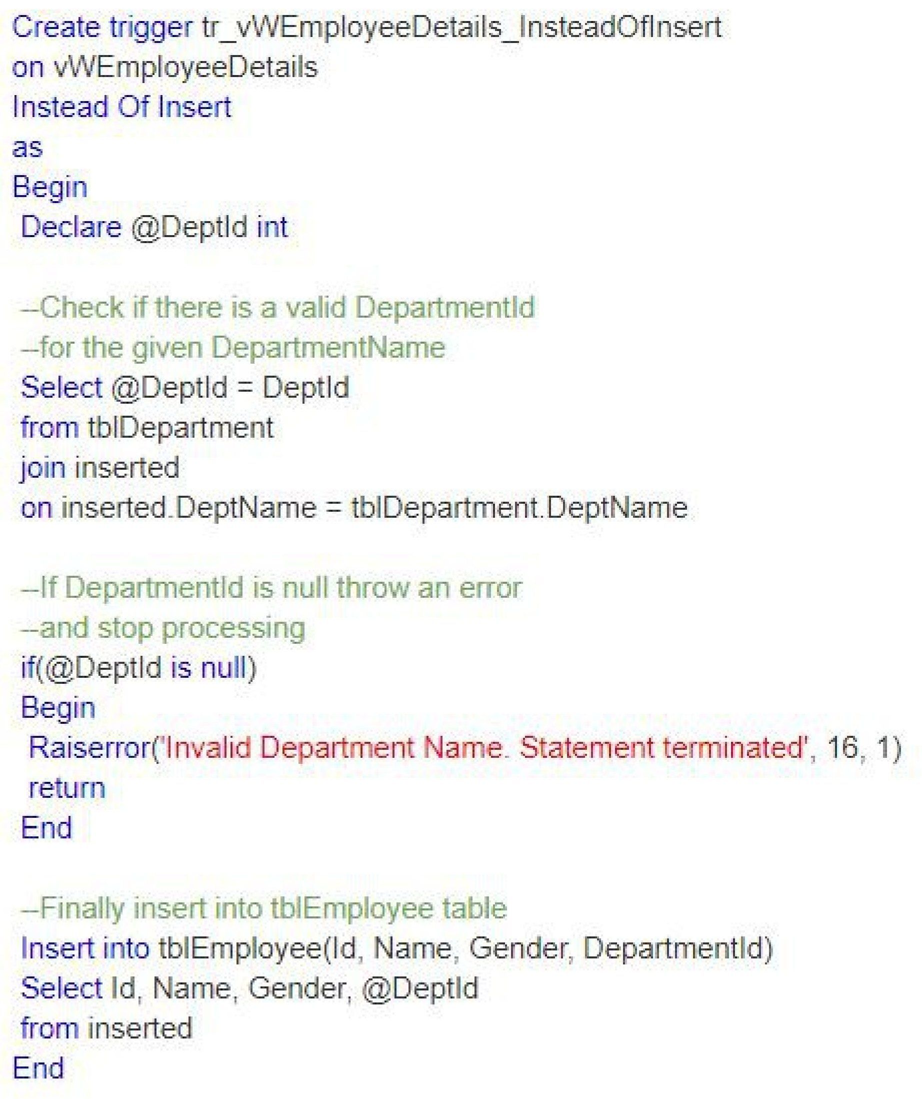
Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

So, let’s insert a single row into the view function, vWEmployeeDetails, by running the following query. At this moment, it will throw an error like “View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.”

Insert into vWEmployeeDetails values (7, ‘Valarie’, ‘Female’, ‘IT’)

Finally, we inserted a row above into a view which is based on multiple tables, it gives an error by default. Now, let’s have a look into this, how INSTEAD OF TRIGGERS give us help in this condition. Since we are facing an error, when we try to insert a single row into the view function, let’s make an INSTEAD OF INSERT trigger on the view vWEmployeeDetails.

SCRIPT TO CREATE INSTEAD OF INSERT TRIGGER:



```
Create trigger tr_vWEmployeeDetails_InsteadOfInsert
on vWEmployeeDetails
Instead Of Insert
as
Begin
 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 tblEmployee(Id, Name, Gender, DepartmentId)
 Select Id, Name, Gender, @DeptId
 from inserted
End
```

So, let’s execute the insertion query as follows:

Insert into vWEmployeeDetails values (7,‘Valarie’, ‘Female’, ‘IT’)

The instead of trigger inserts records wisely, the data records into the tblEmployee table. Since we are inserting the data records into theinserted table, which contains the newly added row data, whereas the deleted table contains empty.

In the trigger action, we used Raiserror() function to show a custom error, once the DepartmentName provides in the insert query, that does not exist. It passes three parameters to the Raiserror() method. The first one parameter shows the error message, and the second parameter shows severity level. Severity level 16, which indicates general errors that only accessible for user correction. The final parameter shows the state parameter.

INSTEAD OF UPDATE TRIGGER

An INSTEAD OF UPDATE triggers gets fired instead of an update event, that can be on a table or a view function. For example, let's understand, an INSTEAD OF UPDATE trigger, and then when you try to make the update into the row within that view function or table, instead of the UPDATE, in this situation, the trigger get invoked automatically. Instead of update trigger based on multiple tables.

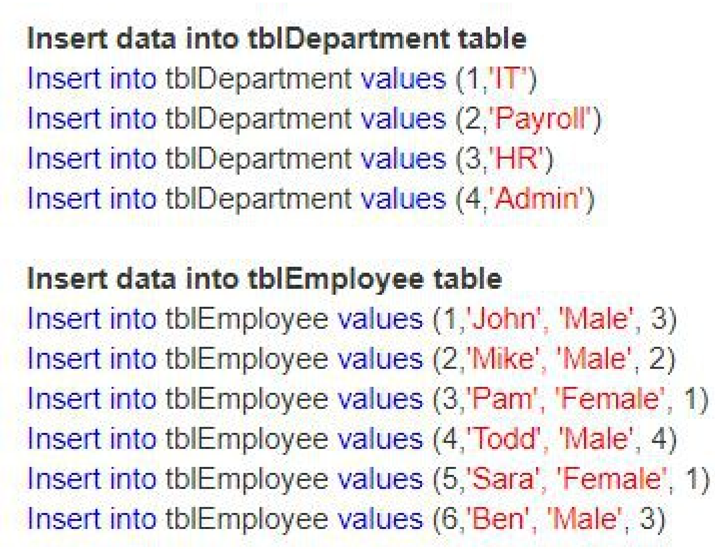
So, let's create both the tables Employee and Department as follows.

SQL SCRIPT TO CREATE TBLEMPLOYEE TABLE:

```
CREATE TABLE tblEmployee
(
Id int Primary Key,
Name nvarchar(30),
Gender nvarchar(10),
DepartmentId int
)
```

SQL SCRIPT TO CREATE TBLDEPARTMENT TABLE

```
CREATE TABLE tblDepartment
(
DeptId int Primary Key,
DeptName nvarchar(20)
)
```



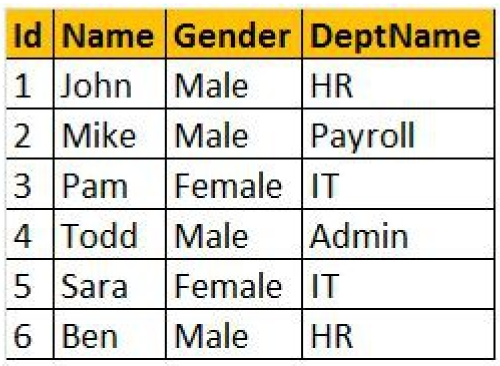
```
Insert data into tblDepartment table
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')

Insert data into tblEmployee table
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

So, we have our required two tables, let's create a view which is based on these two tables, it will fetch the records of Employee Id, Name, Gender and DepartmentName columns which are based on multiple tables.

SCRIPT TO CREATE THE VIEW:

```
Create view vWEmployeeDetails
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Once you execute this line, Select * from vWEmployeeDetails, It will retrieve all the records from the table as follows;



Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

In above example, when we inserted a single row into the view table and got an error statement like- '**View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.**' So, let's quickly update the view function, in addition, it affects both the tables, and if we face the same error statement. Then, the UPDATE command changes its column "**Name**" from the table tblEmployee and column "**DeptName**" from the table tblDepartment. So, when we run this query, we face the same error.

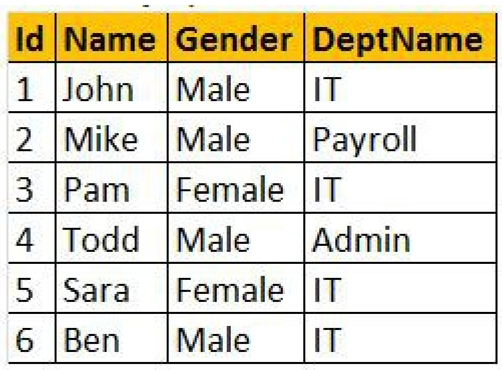
```
Update vWEmployeeDetails
set Name = 'Johny', DeptName = 'IT'
where Id = 1
```

So, let's do some change in the department of **John** from HR to IT. The UPDATE query runs only one table and that is the tblDepartment table. So, the query may succeed. But, there is a condition before executing the query, please make note that employees name JOHN and BEN both are in HR department.

```
Update vWEmployeeDetails
set DeptName = ‘IT’
where Id = 1
```

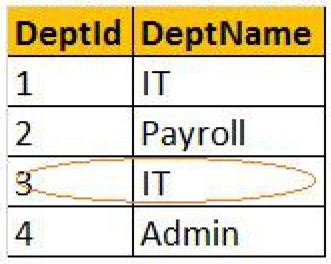
After execution of the query, now select all data records from the view function, and note that BEN’sDeptName has also changed to IT. We also change JOHN’s DeptName. So, the UPDATE command did not work as we expected. Why it happened, because of the UPDATE query command, updated column name DeptName from HR to the IT, in the tblDepartment table. For an update, we need to change the DeptId of JOHN from 3 to 1.

INCORRECTLY UPDATED VIEW



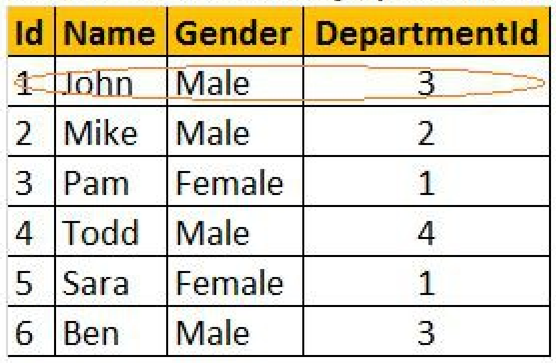
Id	Name	Gender	DeptName
1	John	Male	IT
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	IT

Record with Id = 3, has the DeptName changed from 'HR' to 'IT'



DeptId	DeptName
1	IT
2	Payroll
3	IT
4	Admin

We should have actually updated, JOHN's DepartmentId from 3 to 1



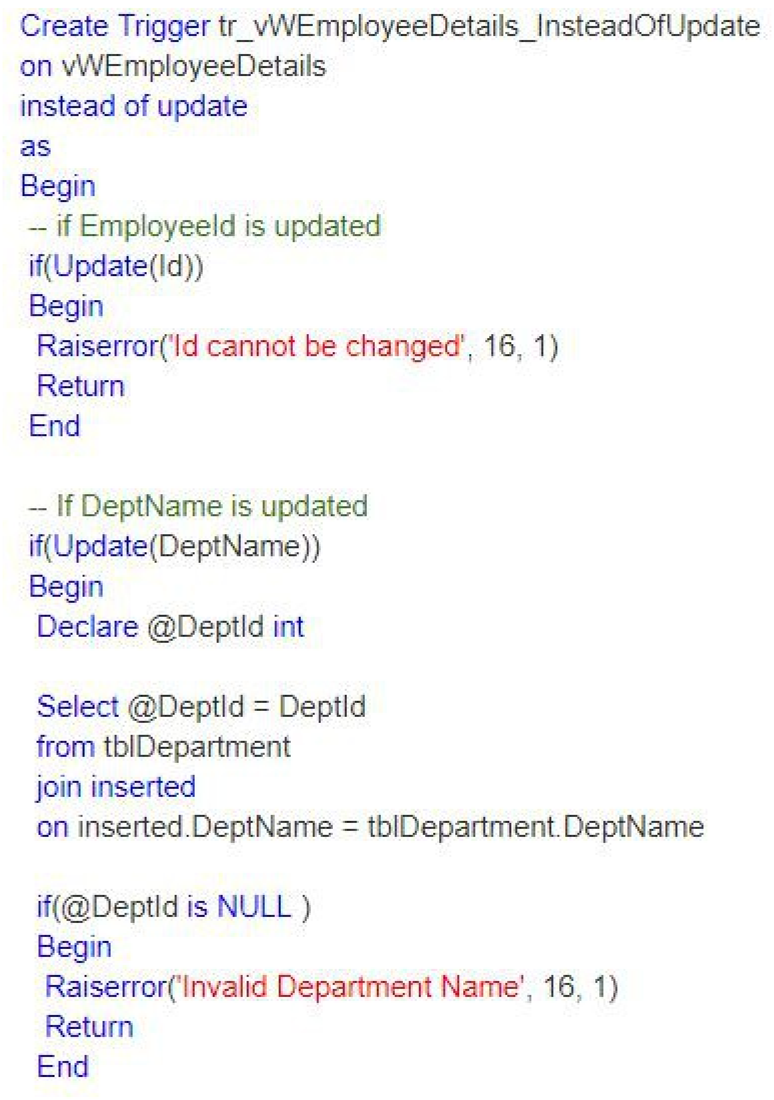
Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

In conclusion, if a view function is based on multiple tables, and if you ever update the view function in triggers, the UPDATE command may not always work as we expect. To resolve this situation, use Instead of Update trigger with a view function.

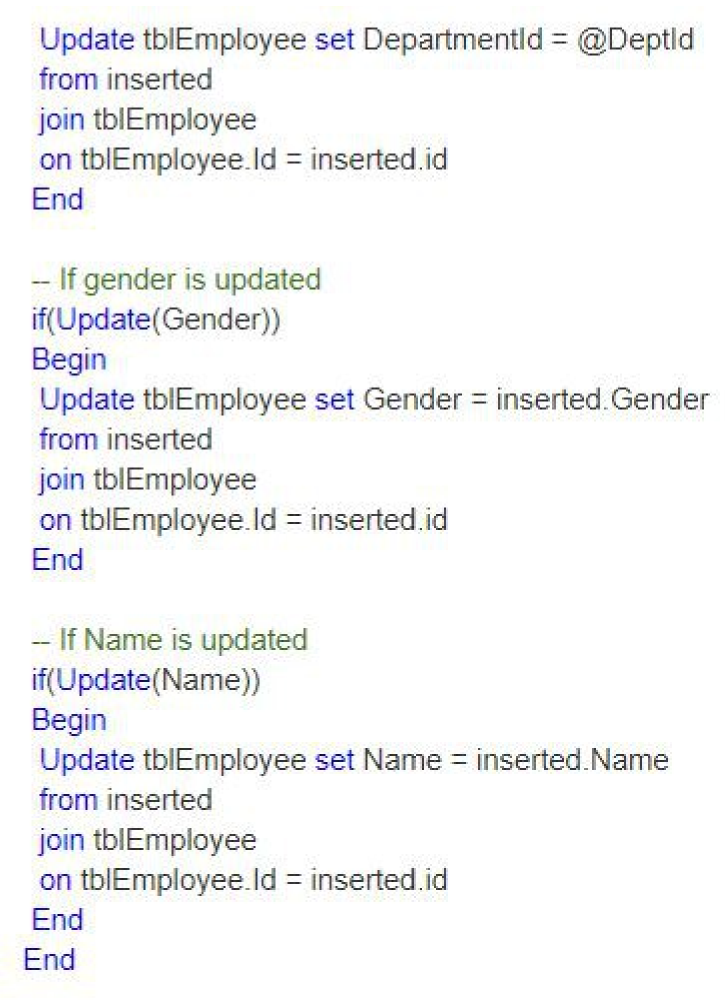
Before we create the trigger, let’s update the DeptName to HR for record with Id = 3.

Update tblDepartment set DeptName = ‘HR’ where DeptId = 3

SCRIPT TO CREATE INSTEAD OF UPDATE TRIGGER:



```
Create Trigger tr_vWEmployeeDetails_InsteadOfUpdate
on vWEmployeeDetails
instead of update
as
Begin
 -- if EmployeeId is updated
 if(Update(Id))
 Begin
  Raiserror('Id cannot be changed', 16, 1)
  Return
 End

 -- If DeptName is updated
 if(Update(DeptName))
 Begin
  Declare @DeptId int

  Select @DeptId = DeptId
  from tblDepartment
  join inserted
  on inserted.DeptName = tblDepartment.DeptName

  if(@DeptId is NULL )
  Begin
   Raiserror('Invalid Department Name', 16, 1)
   Return
  End
```



```
  Update tblEmployee set DepartmentId = @DeptId
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End

 -- If gender is updated
 if(Update(Gender))
 Begin
  Update tblEmployee set Gender = inserted.Gender
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End

 -- If Name is updated
 if(Update(Name))
 Begin
  Update tblEmployee set Name = inserted.Name
  from inserted
  join tblEmployee
  on tblEmployee.Id = inserted.id
 End
End
```

Now, let’s try to update department as “IT” and name “John”.
Update vWEmployeeDetails
set DeptName = ‘IT’
where Id = 1

The UPDATE query works as expected. The INSTEAD OF UPDATE trigger works correctly, and it changes john’s DepartmentId to 1 in the tblEmployee table.

So, let’s try to update in columns Name, Gender, and DeptName. An Update query works properly but it will not give you an error like – ‘**View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.’**
Let’s do the quick update on the views table “vWEmployeeDetails”

Update vWEmployeeDetails
set Name = ‘Johny’, Gender = ‘Female’, DeptName = ‘IT’
where Id = 1

Hence, Update() function used in above example, which gives the result as “True”. It does not matter if you even update with the same value. This is why, I recommend the comparison of values with inserted and deleted tables, instead of using Update() function.

INSTEAD OF DELETE TRIGGER

An INSTEAD OF DELETE trigger gets fired in the state of the DELETE event on a table or a view. Let’s understand with an example. Let’s assume, an INSTEAD OF DELETE trigger on a view or a table, and when you try to update a single row from that view or table, in the state of a real DELETE event, then the trigger automatically gets fired. INSTEAD OF DELETE TRIGGERS only used to delete data in terms of records from a view or a table, which is based on multiple tables.

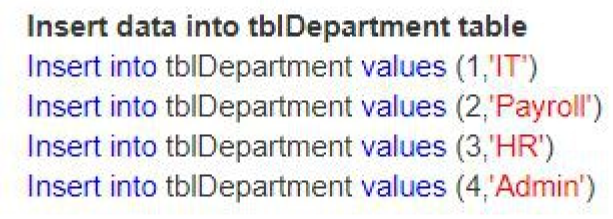
Let’s create two tables Employee and Department.

SQL SCRIPT TO CREATE TBLEMPLOYEE TABLE:

```
CREATE TABLE tblEmployee
(
Id int Primary Key,
Name nvarchar(30),
Gender nvarchar(10),
DepartmentId int
)
```

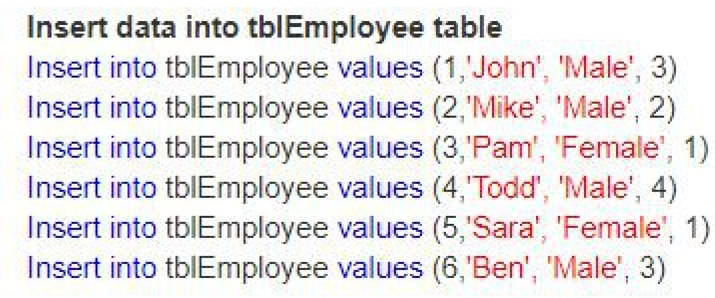
SQL SCRIPT TO CREATE TBLDEPARTMENT TABLE

```
CREATE TABLE tblDepartment
(
DeptId int Primary Key,
DeptName nvarchar(20)
)
```



Insert data into tblDepartment table

```
Insert into tblDepartment values (1,'IT')
Insert into tblDepartment values (2,'Payroll')
Insert into tblDepartment values (3,'HR')
Insert into tblDepartment values (4,'Admin')
```



Insert data into tblEmployee table

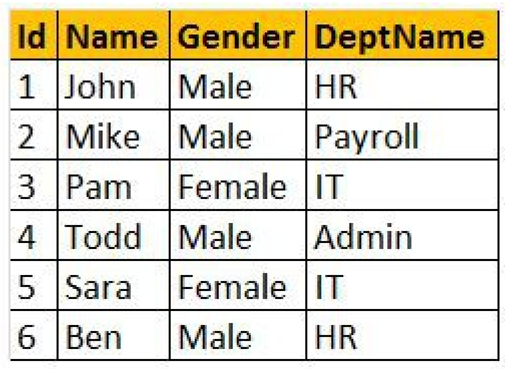
```
Insert into tblEmployee values (1,'John', 'Male', 3)
Insert into tblEmployee values (2,'Mike', 'Male', 2)
Insert into tblEmployee values (3,'Pam', 'Female', 1)
Insert into tblEmployee values (4,'Todd', 'Male', 4)
Insert into tblEmployee values (5,'Sara', 'Female', 1)
Insert into tblEmployee values (6,'Ben', 'Male', 3)
```

Since we now have the required tables, let's create a view based on these tables. And, this view will return Employee Id, Name, Gender and DepartmentName columns. So, the view is based on multiple tables.

SCRIPT TO CREATE THE VIEW:

```
Create view vWEmployeeDetails
as
Select Id, Name, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Once you execute this line, Select * from vWEmployeeDetails, It will retrieve all the records from the table as follows;



Id	Name	Gender	DeptName
1	John	Male	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

In above example, when we inserted a single row into the view table and got an error statement like- '**View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.**'

Although, when we tried to update a view which is based on multiple tables, we faced the same error. To get the error, it will affect both the base tables. If the update query affects only one base table, we do not get the error, but the UPDATE query does not work properly if the “DeptName” column gets updated.

Now, let’s try to delete a row from the view, and we get the same error.

Delete from vWEmployeeDetails where Id = 1

SCRIPT TO CREATE INSTEAD OF DELETE TRIGGER:

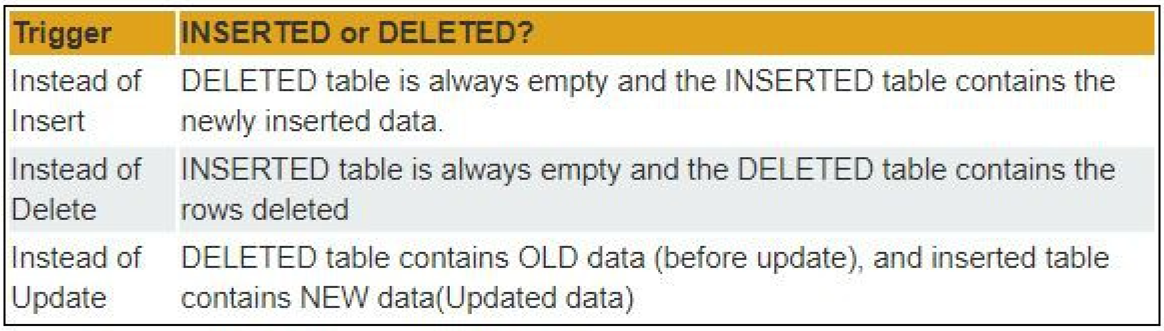
```
Create Trigger tr_vWEmployeeDetails_InsteadOfDelete
on vWEmployeeDetails
instead of delete
as
Begin
Delete tblEmployee
from tblEmployee
join deleted
on tblEmployee.Id = deleted.Id

–Subquery

–Delete from tblEmployee
–where Id in (Select Id from deleted)
End
```

Note: The trigger tr_vWEmployeeDetails_InsteadOfDelete applicable in DELETED table. But Deleted table contains all the rows that we tried to DELETE from the view. So, we join the DELETED table with table tblEmployee to delete the unwanted rows. In such cases, Joins are much faster than the subqueries.

When you execute the following DELETE command, the row gets DELETED as expected from tblEmployee table

Delete from vWEmployeeDetails where Id = 1

A small difference among the triggers as given below;



Trigger	INSERTED or DELETED?
Instead of Insert	DELETED table is always empty and the INSERTED table contains the newly inserted data.
Instead of Delete	INSERTED table is always empty and the DELETED table contains the rows deleted
Instead of Update	DELETED table contains OLD data (before update), and inserted table contains NEW data(Updated data)

DDL TRIGGERS

DDL triggers is an operation that only gets fired in response to DDL events – CREATE, ALTER, and DROP (Table, Function, Index, Stored Procedure) these are the DDL triggers commands or we can say only DDL commands in SQL. System stored procedures, that perform DDL operations can also fire DDL triggers in the SQL commands.

Example – sp_rename system stored procedure.

WHY THE USE OF DDL TRIGGERS.

•Triggers used for improvement in data integrity and liability of the database.
•Triggers maintain **Audit** file and table structure in the database.
Syntax for creating DDL trigger as follows;

```
CREATE TRIGGER [Trigger_Name]
ON [Scope (Server|Database)]
FOR [EventType1, EventType2, EventType3, ...],
AS

BEGIN

  – Trigger Body

END
```

DDL triggers can be created in a database server.

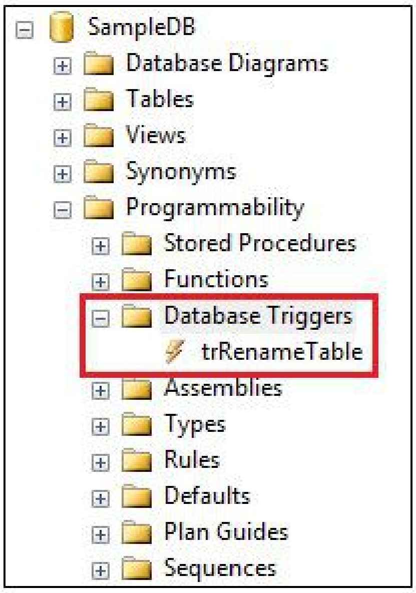
The following trigger will fire in response to CREATE_TABLE DDL event.

```
CREATE TRIGGER trMyFirstTrigger

ON Database

FOR CREATE_TABLE
```

```
AS
BEGIN
  Print ‘New table created’
END
```

HOW TO CHECK TRIGGER HAS BEEN CREATED OR NOT?

1.Open your “Object Explorer window”, then further expand the “SampleDB” database by click on the plus symbol in near the folder name of SampleDB.
2.Then, expand Programmability folder.
3.Finally, expand Database Triggers folder. Please have a look at given image for reference as follows;



Please note: If you do not find the trigger that you just created, I suggest you refresh the Database Triggers folder.

When you execute the given code to just create the table, then the trigger automatically get fired and in return, it prints the message – **New table created**

```
Create Table Test (Id int)
```

The above trigger gets fired only for one DDL event which is “Create Table”. Furthermore, if you want that trigger to get fired with multiple events, let’s understand an example when you ever make your table in alter or drop commands, then separate the events by using a comma as follows.

ALTER TRIGGER trMyFirstTrigger

ON Database

FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE

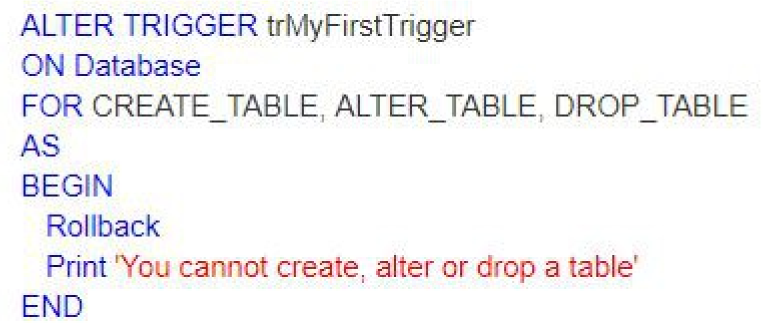
AS

BEGIN

  Print '**A table has just been created, modified or deleted**'
END

So, now if you want to create, alter or drop a table, the trigger automatically gets fired and then you get the message like as – "**A table has just been created, modified or deleted**".

The second DDL triggers perform some code to DDL events. So, let's look at an example of how to save users from creating, altering or dropping tables. To get this you need to modify the trigger as given below.



```
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
  Rollback
  Print 'You cannot create, alter or drop a table'
END
```

To create, alter or drop a table, you either have to disable or delete the trigger. How do we get this, let's understand the following checkpoints for disabling and enabling it.

HOW TO DISABLE TRIGGER?

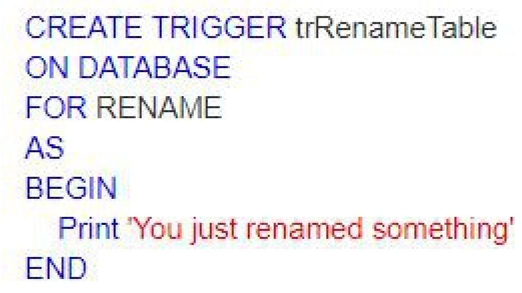
•First, Open your object explorer window on the left corner of your window screen, then do Right-click on the trigger and select option "**Disable**" from the context menu folder.
•Further, you can also disable the trigger by using the T-SQL commands like as follows DISABLE TRIGGER trMyFirstTrigger ON DATABASE

HOW TO ENABLE TRIGGER?

•First, Open your object explorer window on the left corner of your window screen, then do Right-click on the trigger and select option "**Enable**" from the context menu folder.
•Further, you can also enable the trigger by using the T-SQL commands like as follows ENABLE TRIGGER trMyFirstTrigger ON DATABASE

HOW TO DELETE TRIGGER?

•First, Open your object explorer window on the left corner of your window screen, then do Right-click on the trigger and select option "**Delete**" from the context menu folder.
•Further, you can also delete the trigger by using the T-SQL commands like as follows DROP TRIGGER trMyFirstTrigger ON DATABASE
System stored procedures that perform DDL operations can also fire DDL triggers. A given trigger gets fired once you rename a database object using "sp_rename" system stored procedure name. You can get your output by using this query.



```
CREATE TRIGGER trRenameTable
ON DATABASE
FOR RENAME
AS
BEGIN
   Print 'You just renamed something'
END
```

In the given above code, that completely changes the name of the **TestTable** to **NewTestTable**. When you run this code, it will execute and it will get fire the trigger **trRenameTable.** For this reason, let's have a look at given code as follows;
sp_rename 'TestTable', 'NewTestTable'

In the given above code, that completely changes the name of the "**Id"** column in **NewTestTable**to **NewId**.When you run this code, it will execute and it will get fire the trigger **trRenameTable.** For this reason, let's have a look at given code as follows;
sp_rename 'NewTestTable.Id' , 'NewId', 'column'

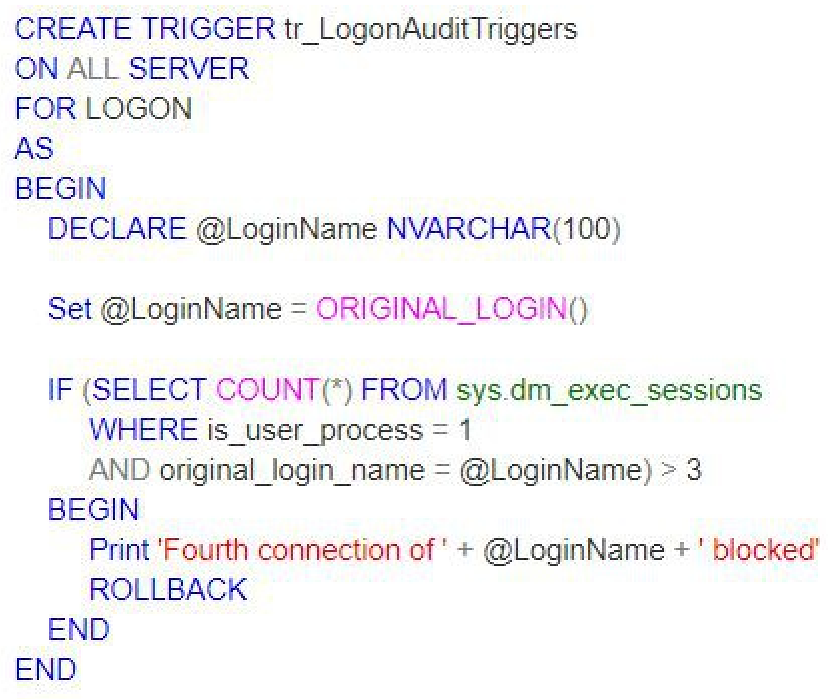
LOGON TRIGGERS

Logon triggers is an operation that only gets executed automatically in response to a LOGON event. These triggers get fired only after the log finishes in the authentication phase but user session must be established before. Moreover, you will learn about why we use logon triggers.

WHY WE USE FOR LOGON TRIGGERS

Below are some meaningful cases in logon triggers

1.For tracking login activity in the database.
2.To restrict logins authority to SQL Server programs.
3.To restrict the number of sessions for a specific login.
Let’s understand with a Logon trigger example: Trigger sets limits of the maximum number of open connections with a user to 3.



```
CREATE TRIGGER tr_LogonAuditTriggers
ON ALL SERVER
FOR LOGON
AS
BEGIN
   DECLARE @LoginName NVARCHAR(100)

   Set @LoginName = ORIGINAL_LOGIN()

   IF (SELECT COUNT(*) FROM sys.dm_exec_sessions
       WHERE is_user_process = 1
       AND original_login_name = @LoginName) > 3
   BEGIN
       Print 'Fourth connection of ' + @LoginName + ' blocked'
       ROLLBACK
   END
END
```

As a result, trigger error message gets straight to the error log window. Then, execute the given command to read the error log.

Execute sp_readerrorlog

CONCLUSION

In this article, I described all the various types of Triggers like DDL, DML, and Logon. It is very useful to maintain the data integrity constraints in the database in the absence of SQL constraints keys (primary key and foreign key). Triggers is much useful feature of SQL/T-SQL and you can also use it in Oracle.

Moreover, Triggers control update format on which update allows the database. Triggers play a vital role in calling stored procedures. It is useful in the corporate sector to maintain the track of all the employees' record which need to be changed like (update, deletion, and insertion) in the tables.

Connect with source url:-

https://www.loginworks.com/blogs/work-triggers-types-sql/