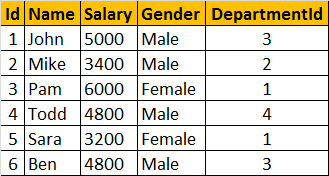
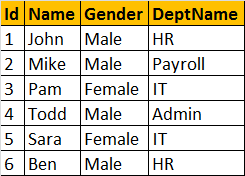
**DML Triggers**

**In general, a trigger is a special kind of stored procedure** that automatically executes when an event occurs in the database server.  
  
**DML stands for Data Manipulation Language.** INSERT, UPDATE, and DELETE statements are DML statements. DML triggers are fired, when ever data is modified using INSERT, UPDATE, and DELETE events.  
  
**DML triggers can be again classified into 2 types.**  
1. After triggers (Sometimes called as FOR triggers)  
2. Instead of triggers  
  
**After triggers, as the name says, fires after the triggering action**. The INSERT, UPDATE, and DELETE statements, causes an after trigger to fire after the respective statements complete execution.  
  
**On ther hand, as the name says, INSTEAD of triggers, fires instead of the triggering action**. The INSERT, UPDATE, and DELETE statements, can cause an INSTEAD OF trigger to fire INSTEAD OF the respective statement execution.  
  
**We will use tblEmployee and tblEmployeeAudit**tables for our examples  
  
**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)  
  
**tblEmployee**  
  
  
  
**SQL Script to create tblEmployeeAudit table:**  
CREATE TABLE tblEmployeeAudit  
(  
  Id int identity(1,1) primary key,  
  AuditData nvarchar(1000)  
)  
  
**When ever, a new Employee is added**, we want to capture the ID and the date and time, the new employee is added in tblEmployeeAudit table. The easiest way to achieve this, is by having an AFTER TRIGGER for INSERT event.  
  
**Example for AFTER TRIGGER for INSERT event on tblEmployee table:**  
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  
  
**In the trigger, we are getting the id from inserted table.** So, what is this inserted table? INSERTED table, is a special table used by DML triggers. When you add a new row into tblEmployee table, a copy of the row will also be made into inserted table, which only a trigger can access. You cannot access this table outside the context of the trigger. The structure of the inserted table will be identical to the structure of tblEmployee table.  
  
**So, now if we execute the following INSERT statement on tblEmployee.** Immediately, after inserting the row into tblEmployee table, the trigger gets fired (executed automatically), and a row into tblEmployeeAudit, is also inserted.  
**Insert into tblEmployee values (7,'Tan', 2300, 'Female', 3)**  
  
**Along, the same lines, let us now capture audit information, when a row is deleted**from the table, tblEmployee.  
**Example for AFTER TRIGGER for DELETE event on 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  
  
**The only difference here is that**, we are specifying, the triggering event as **DELETE** and retrieving the deleted row ID from **DELETED** table. DELETED table, is a special table used by DML triggers. When you delete a row from tblEmployee table, a copy of the deleted row will be made available in DELETED table, which only a trigger can access. Just like INSERTED table, DELETED table cannot be accessed, outside the context of the trigger and, the structure of the DELETED table will be identical to the structure of tblEmployee table.

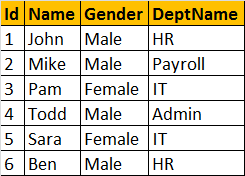
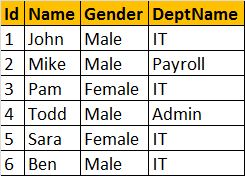
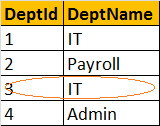
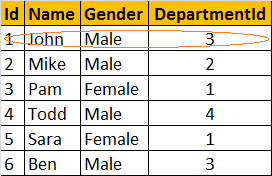
### After update trigger

**Triggers make use of 2 special tables**, INSERTED and DELETED. The inserted table contains the updated data and the deleted table contains the old data. The After trigger 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  
  
**Now, execute this query:**  
Update tblEmployee set Name = 'Tods', Salary = 2000,   
Gender = 'Female' where Id = 4  
  
**Immediately after the UPDATE statement execution**, the AFTER UPDATE trigger gets fired, and you should see the contenets of INSERTED and DELETED tables.  
  
**The following AFTER UPDATE trigger, audits employee information upon UPDATE**, 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

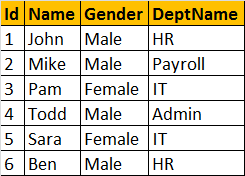
### Instead of insert trigger

**INSTEAD OF triggers**, specifically INSTEAD OF INSERT trigger. We know that, AFTER triggers are fired after the triggering event(INSERT, UPDATE or DELETE events), where as, INSTEAD OF triggers are fired instead of the triggering event(INSERT, UPDATE or DELETE events). In general, INSTEAD OF triggers are usually used to correctly update views that are based on multiple tables.   
  
**We will base our demos 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)  
  
**Since, we now have the required tables**, let's create a view based on these tables. The view should return Employee Id, Name, Gender and DepartmentName columns. So, the view is obviously 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  
  
**When you execute**, **Select \* from vWEmployeeDetails**, the data from the view, should be as shown below  
  
  
**Now, let's try to insert a row into the view, vWEmployeeDetails**, by executing the following query. At this point, an error will be raised stating 'View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.'  
Insert into vWEmployeeDetails values(7, 'Valarie', 'Female', 'IT')  
  
**So, inserting a row into a view that is based on multipe tables**, raises an error by default. Now, let's understand, how INSTEAD OF TRIGGERS can help us in this situation. Since, we are getting an error, when we are trying to insert a row into the view, let's create 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  
  
**Now, let's execute the insert query:**  
Insert into vWEmployeeDetails values(7, 'Valarie', 'Female', 'IT')  
  
**The instead of trigger correctly inserts**, the record into tblEmployee table. Since, we are inserting a row, the **inserted** table, contains the newly added row, where as the **deleted** table will be empty.  
  
**In the trigger, we used Raiserror() function**, to raise a custom error, when the DepartmentName provided in the insert query, doesnot exist. We are passing 3 parameters to the Raiserror() method. The first parameter is the error message, the second parameter is the severity level. Severity level 16, indicates general errors that can be corrected by the user. The final parameter is the state.

### Instead of update triggers

**INSTEAD OF UPDATE** trigger. An INSTEAD OF UPDATE triggers gets fired instead of an update event, on a table or a view. For example, let's say we have, an INSTEAD OF UPDATE trigger on a view or a table, and then when you try to update a row with in that view or table, instead of the UPDATE, the trigger gets fired automatically. INSTEAD OF UPDATE TRIGGERS, are of immense help, to correctly update a view, that is based on multiple tables.  
  
**Let's create the required Employee and Department tables**, that we will be using for this demo.  
  
**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. The view should return Employee Id, Name, Gender and DepartmentName columns. So, the view is obviously 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  
  
**When you execute, Select \* from vWEmployeeDetails**, the data from the view, should be as shown below  
  
  
**In Part 45, we tried to insert a row into the view**, and we got an error stating - 'View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.'  
  
**Now, let's try to update the view**, in such a way that, it affects, both the underlying tables, and see, if we get the same error. The following UPDATE statement changes **Name column** from **tblEmployee** and **DeptName column** from **tblDepartment**. So, when we execute this query, we get the same error.  
Update vWEmployeeDetails   
set Name = 'Johny', DeptName = 'IT'  
where Id = 1  
  
**Now, let's try to change, just the department of John from HR to IT**. The following UPDATE query, affects only one table, tblDepartment. So, the query should succeed. But, before executing the query, please note that, employees **JOHN** and **BEN** are in **HR** department.  
Update vWEmployeeDetails   
set DeptName = 'IT'  
where Id = 1  
  
**After executing the query**, select the data from the view, and notice that **BEN's** **DeptName** is also changed to **IT**. We intended to just change **JOHN's DeptName**. So, the UPDATE didn't work as expected. This is because, the UPDATE query, updated the **DeptName from HR to IT**, in tblDepartment table. For the UPDATE to work correctly, we should change the **DeptId** of **JOHN** from 3 to 1.  
  
**Incorrectly Updated View**  
  
  
**Record with Id = 3, has the DeptName changed from 'HR' to 'IT'**  
  
  
**We should have actually updated, JOHN's DepartmentId from 3 to 1**  
  
  
  
**So, the conclusion is that, if a view is based on multiple tables**, and if you update the view, the UPDATE may not always work as expected. To correctly update the underlying base tables, thru a view, INSTEAD OF UPDATE TRIGGER can be used.  
  
**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 JOHN's Department to IT.**  
Update vWEmployeeDetails   
set DeptName = 'IT'  
where Id = 1  
  
**The UPDATE query works as expected.** The INSTEAD OF UPDATE trigger, correctly updates, JOHN's DepartmentId to 1, in tblEmployee table.  
  
**Now, let's try to update Name, Gender and DeptName.** The UPDATE query, works as expected, without raising the error - 'View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables.'  
Update vWEmployeeDetails   
set Name = 'Johny', Gender = 'Female', DeptName = 'IT'   
where Id = 1  
  
**Update**() function used in the trigger, returns true, even if you update with the same value. For this reason, I recomend to compare values between inserted and deleted tables, rather than relying on Update() function. The Update() function does not operate on a per row basis, but across all rows.

### Instead of delete trigger

**INSTEAD OF DELETE trigger**. An INSTEAD OF DELETE trigger gets fired instead of the DELETE event, on a table or a view. For example, let's say we have, an INSTEAD OF DELETE trigger on a view or a table, and then when you try to update a row from that view or table, instead of the actual DELETE event, the trigger gets fired automatically. INSTEAD OF DELETE TRIGGERS, are used, to delete records from a view, that is based on multiple tables.  
  
Let's create the required Employee and Department tables, that we will be using for this demo.  
  
**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. The view should return Employee Id, Name, Gender and DepartmentName columns. So, the view is obviously 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  
  
**When you execute, Select \* from vWEmployeeDetails**, the data from the view, should be as shown below  
  
  
**In**[**Part 45**](http://csharp-video-tutorials.blogspot.com/2012/09/instead-of-insert-trigger-part-45.html), we tried to insert a row into the view, and we got an error stating - 'View or function vWEmployeeDetails is not updatable because the modification affects multiple base tables'. Along, the same lines, in [**Part 46**](http://csharp-video-tutorials.blogspot.com/2012/09/instead-of-update-triggers-part-46.html), when we tried to update a view that is based on multiple tables, we got the same error. To get the error, the UPDATE should affect both the base tables. If the update affects only one base table, we don't get the error, but the UPDATE does not work correctly, if the **DeptName**column is 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  
  
**Notice that, the trigger tr\_vWEmployeeDetails\_InsteadOfDelete**, makes use of DELETED table. DELETED table contains all the rows, that we tried to DELETE from the view. So, we are joining the DELETED table with tblEmployee, to delete the rows. You can also use sub-queries to do the same. In most cases JOINs are faster than SUB-QUERIEs. However, in cases, where you only need a subset of records from a table that you are joining with, sub-queries can be faster.  
  
**Upon executing the following DELETE statement**, the row gets DELETED as expected from tblEmployee table  
Delete from vWEmployeeDetails where Id = 1

|  |  |
| --- | --- |
| **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) |