**EXPERIMENT 8**

**Hafiz Ahmad 19L-1316**

# Triggers

**Objective**

* Practical implementation of SQL triggers
* Learn how to Create/Alter/drop DML and DDL triggers

**Overview:**

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. An SQL trigger may call stored procedures or user-defined functions to perform additional processing when the trigger is executed.

Unlike stored procedures, an SQL trigger cannot be directly called from an application. Instead, an SQL trigger is invoked by the database management system on the execution of a triggering insert, update, or delete operation. The definition of the SQL trigger is stored in the database management system and is invoked by the database management system, when the SQL table, that the trigger is defined on, is modified.

Triggers can be divided into two main categories

* DML Triggers
* DDL Triggers

## DML Triggers

DML triggers is a special type of stored procedure that automatically takes effect when a data manipulation language (DML) event takes place that affects the table or view defined in the trigger.

DML events include INSERT, UPDATE, or DELETE statements. DML triggers can be used to enforce business rules and data integrity, query other tables, and include complex Transact-SQL statements. The trigger and the statement that fires it are treated as a single transaction, which can be rolled back from within the trigger.

Basically, DML triggers are classified into two main types: -   
  
**(i) After Triggers (For Triggers)   
(ii) Instead Of Triggers**

Using your own database please run the following script to create and populate the table used for today’s lab in order to practice triggers.

create table TriggerEmployee

(Name varchar(50), EmployeeID int primary key,ContactID int ,

ManagerID int ,Gender char(1),Description varchar(50))

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Ahmed',1,2,2,'M','xyz')

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Osama',2,1,2,'M','sadsd')

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Qasim',3,1,2,'M','sadsff')

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Hamid',4,1,2,'M','hamid')

**After Triggers (For Triggers)**

These triggers run after an insert, update or delete on a table. They are **not supported for views.**

Given below we have a trigger that is fired after an update on the table.

CREATE TRIGGER Trigger\_ForUpdate

ON DBO.TriggerEmployee

FOR UPDATE

AS

UPDATE TriggerEmployee

SET Description = 'changed with For update trigger'

To fire the trigger we can perform update on table TriggerEmployee using

UPDATE TriggerEmployee

SET Name = 'New Name'

WHERE EmployeeID= 3

Select data from above table to see that the trigger fired and description got updated.

select \* from TriggerEmployee

Similarly we can make after triggers or For Triggers for insert and delete as well.

**How to use inserted or deleted rows:**

First refresh the TriggerEmployee table by running the script given below:

--refreshing the data

truncate table TriggerEmployee

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Ahmed',1,2,2,'M','xyz')

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Osama',2,1,2,'M','sadsd')

insert into TriggerEmployee (Name, EmployeeID,ContactID,ManagerID,Gender,Description) values ('Qasim',3,1,2,'M','sadsff')

select \* from triggerEmployee

go

Now modify the Trigger\_ForUpdate trigger by running the script given below:

alter TRIGGER Trigger\_ForUpdate

ON DBO.TriggerEmployee

FOR UPDATE

AS

declare @Employeeid int -- variable declaration

-- get the value from the table of employee id that was updated

select @Employeeid=EmployeeID from inserted

UPDATE TriggerEmployee

SET Description = 'changed with For update trigger'

WHERE EmployeeID= @EmployeeID

go

UPDATE TriggerEmployee

SET Name = 'New Name'

WHERE EmployeeID= 2

go

select \*from TriggerEmployee

go

**Instead Of Triggers:**

These can be used as an interceptor for anything that anyone tried to do on our table or view. If you define an *Instead Of trigger* on a table for the Delete operation, then try to delete rows, and they will not actually get deleted (unless you issue another delete instruction from within the trigger).

We have 3 types of instead of triggers.

**(a) INSTEAD OF INSERT Trigger.   
(b) INSTEAD OF UPDATE Trigger.   
(c) INSTEAD OF DELETE Trigger.**

Example of instead of insert trigger

create TRIGGER SampleTrigger\_select

ON DBO.TriggerEmployee

INSTEAD OF INSERT

AS

SELECT \* FROM TriggerEmployee

print 'sorry this table cannot be modified'

go

INSERT INTO TriggerEmployee(EmployeeID,NAME,ContactID,ManagerID,Gender)

VALUES(5, 'Instead of insert ' , 1108, 1, 'M' )

go

SELECT \* FROM TriggerEmployee -- NO CHANGE IN THE TABLE

Go

Instead of update and delete triggers can be made using similar way as well. Below is an example of an intead of delete trigger:

create TRIGGER [dbo].[T\_DeleteEmployee]

ON [dbo].TriggerEmployee

INSTEAD OF delete

AS

IF

(

SELECT COUNT(\*)

FROM TriggerEmployee A

) >0

BEGIN

PRINT 'CAN NOT BE deleted'

END

go

DELETE from TriggerEmployee WHERE EmployeeID= 1

go

select \*From TriggerEmployee

go

## DDL Triggers

DDL triggers, like regular triggers, fire stored procedures in response to an event. However, unlike DML triggers, they do not fire in response to UPDATE, INSERT, or DELETE statements on a table or view. Instead, they fire in response to a variety of Data Definition Language (DDL) events. These events primarily correspond to SQL statements that start with the keywords CREATE, ALTER, and DROP.

Use DDL triggers when you want to do the following:

* You want to prevent certain changes to your database schema.
* You want something to occur in the database in response to a change in your database schema.
* You want to record changes or events in the database schema.

Example of DDL trigger fired on drop\_table event

CREATE TRIGGER [T\_DeleteTable]

ON DatabASE

FOR DROP\_TABLE

AS

PRINT 'CAN NOT BE dropped'

RollBack -- we dont have instead of in DDL so we use rollback

go

To fire the trigger we can try to drop any table of the database on which trigger was defined.

DROP TABLE TriggerEmployee

go

Example of DDL trigger fired on alter\_table event

create TRIGGER [T\_AlterTable\_DT]

ON DatabASE

FOR ALTER\_TABLE

AS

PRINT 'Table alterd succesfully'

go

To fire the trigger we can try to alter any table of the database on which trigger was defined.

ALTER TABLE TriggerEmployee ALTER COLUMN [NAME] VARCHAR(60)

-- select \* from [TriggerEmployee]

go

**InLab Questions:**

Using the company database please write the following triggers. Paste your code as well as screenshot of the result when trigget is fired in the space provided

1. The HR Officer is concerned with database auditing so he decides to maintain a record of changes made to database. Create a table Auditing in database with columns AuditId, description and LastChangeOn using

CREATE TABLE Auditing (

AuditId INT IDENTITY,

Description VARCHAR(100),

LastChangeOn DATE

);

Create trigger on employee table so that when ever any change is made to it via insert, update and delete events the date of change gets stored in the Auditing table.

use [L191316]

CREATE TABLE Auditing (

AuditId INT IDENTITY,

Description VARCHAR(100),

LastChangeOn DATE

);

create trigger AuditEmployeeChanges

on Employee

after insert, update, delete

as

begin

declare @ChangeDescription VARCHAR(100);

set @ChangeDescription = '';

IF EXISTS (SELECT \* FROM inserted)

SET @ChangeDescription = 'Employee Insert / Update';

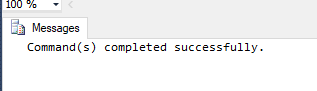
ELSE IF EXISTS (SELECT \* FROM deleted)

SET @ChangeDescription = 'Employee Deleted';

insert into Auditing (Description, LastChangeOn)

values (@ChangeDescription, GETDATE());

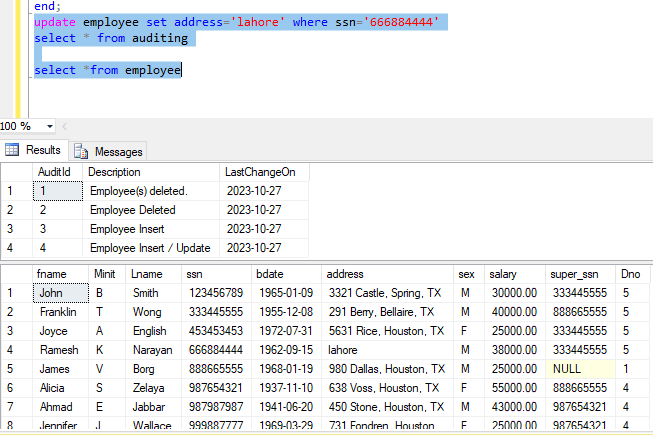
end;



Check whether your trigget worked by running

update employee set address='lahore' where ssn='666884444'

select \* from auditing



1. Though HR officer was pretty much satisfied with your last change but after few months he feels something is missing in auditing table so he ask to add more info about the event that changed the employee table . So now he want to store proper information like

2/09/23 employee updated with id=5

7/10/23 employee deleted with id=1

So make appropriate change in triggers to perform the operation.

alter trigger AuditEmployeeChanges

on Employee

after insert, update, delete

as

begin

declare @ChangeDescription VARCHAR(100);

set @ChangeDescription = '';

IF EXISTS (SELECT \* FROM inserted)

BEGIN

SELECT @ChangeDescription = 'Employee updated with id=' + CAST((SELECT ssn FROM inserted) AS VARCHAR(10));

END

ELSE IF EXISTS (SELECT \* FROM deleted)

BEGIN

SELECT @ChangeDescription = 'Employee deleted with id=' + CAST((SELECT ssn FROM deleted) AS VARCHAR(10));

END

insert into Auditing (Description, LastChangeOn)

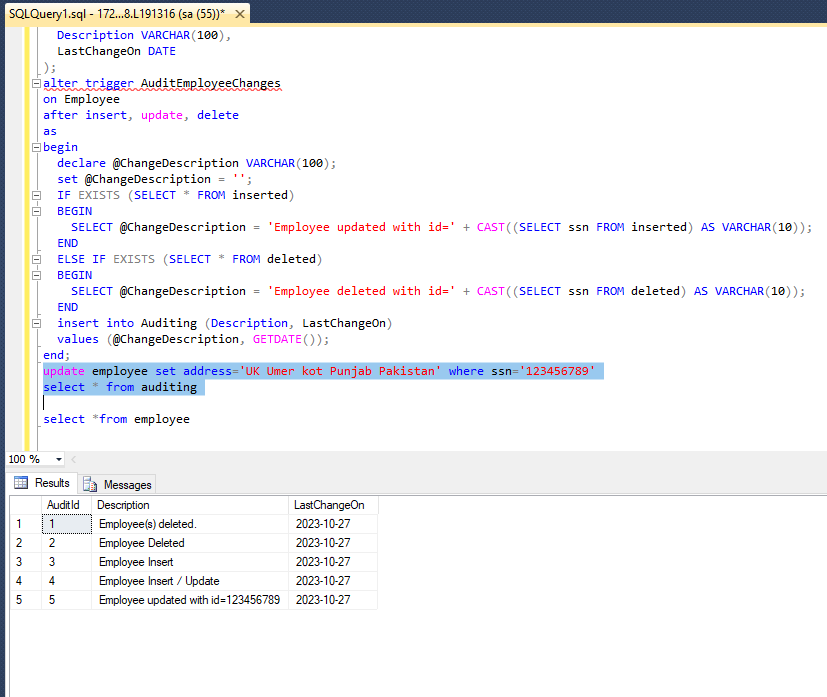
values (@ChangeDescription, GETDATE());

end;

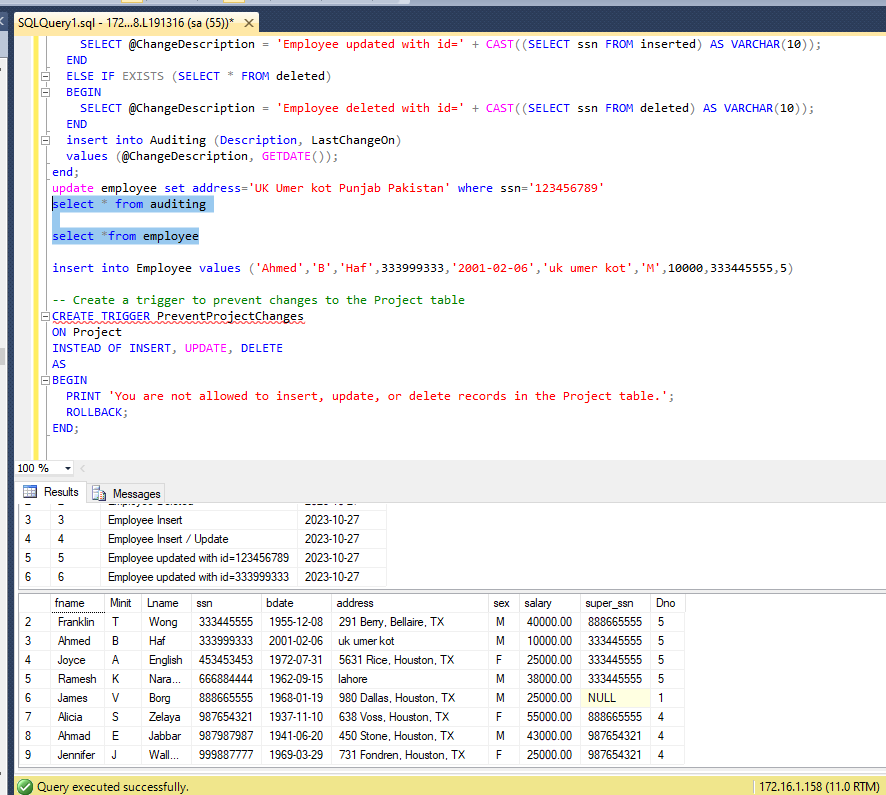
update employee set address='UK Umer kot Punjab Pakistan' where ssn='123456789'

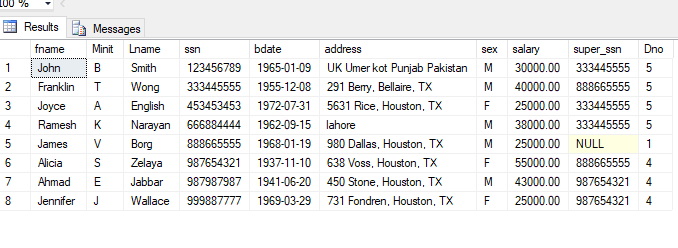
select \* from auditing

select \*from employee

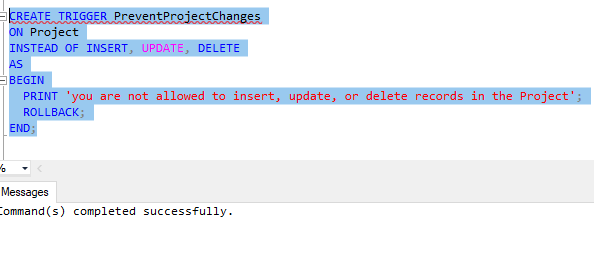


insert into Employee values ('Ahmed','B','Haf',333999333,'2001-02-06','uk umer kot','M',10000,333445555,5)





1. The HR officer also wants to make sure no one insert, update or deletes project information from the database. So create a trigger that will not let anyone to change the project table.



CREATE TRIGGER PreventProjectChanges

ON Project

INSTEAD OF INSERT, UPDATE, DELETE

AS

BEGIN

PRINT 'you are not allowed to insert, update, or delete records in the Project';

ROLLBACK;

END;

Verify the trigget is working by running the following command

delete from project where dnum=5

