National University of Computer and Emerging Sciences

In Lab

"Triggers"

Database Systems Lab

Department of Computer Science

FAST-NU, Lahore, Pakistan

1. Objective

In this lab, the following concepts will be discussed along with the implementation

1. Triggers

2. Triggers

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

a. **DML Triggers**
b. **DDL Triggers**

a. 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

I. After Triggers (For Triggers)

These triggers run after an insert, update or delete on a table.
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'
--WHERE EmployeeID= 1
```

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

```
UPDATE TriggerEmployee
SET Name = 'New Name'
WHERE EmployeeID= 4
```

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

- **INSERTED and DELETED Table:** in the above example we are updating Description in all the rows what if I want update only the Description of row with Employee ID 4 (that was actually updated)?

  For that we use special table "DELETED" and "INSERTED" designed.
  On every **insert**, **delete and update** statement these tables are created, and the effected rows from DML is maintained in these tables temporarily

  More details
  http://technet.microsoft.com/en-us/library/ms191300.aspx

  NOTE THESE TABLES CAN ONLY BE ACCESSED IN BODY OF TRIGGERS
  The tables are accessed like ordinary table

```
select * from inserted
select * from deleted
```

  Following example shows how these tables are created and used in triggers

  Consider the following table **TriggerEmployee**

Name	EmployeeID	ContactID	ManagerID	Gender	Description
Ahmed	1	2	2	M	xyz
Osama	2	1	2	M	sadsd

- **For Insert Statement**

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

On excuted this query following is the data in inserted and deleted table

INSERTED

Name	EmployeeID	ContactID	ManagerID	Gender	Description
Qasim	3	1	2	M	sadsff

DELETED

Name	EmployeeID	ContactID	ManagerID	Gender	Description

**Note that there is no data in deleted table as we have not deleted anything
The inserted table holds the rows that were inserted by this statrment

- **For Delete Statement**

```
delete from  TriggerEmployee
```

On excuted this query following is the data in inserted and deleted table

INSERTED

Name	EmployeeID	ContactID	ManagerID	Gender	Description

DELETED

Name	EmployeeID	ContactID	ManagerID	Gender	Description
Qasim	3	1	2	M	sadsff
Osama	2	1	2	M	sadsd
Ahmed	1	2	2	M	xyz

- **FOR UPDATE STATEMENT**

```
UPDATE TriggerEmployee
SET Name = 'New Name'
WHERE EmployeeID= 1
```

Here we have value in both tables, because update is considered as combination of delete and insert (old row is deleted , replaced by inserting new updated row)

INSERTED

Name	EmployeeID	ContactID	ManagerID	Gender	Description
New Name	1	2	2	M	xyz

DELETED

Name	EmployeeID	ContactID	ManagerID	Gender	Description
Ahmed	1	2	2	M	xyz

EXAMPLE: HOW TO USE DELETED AND INSERTED TABLES

Given the same table

```
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
```

Results | Messages

Name	EmployeeID	ContactID	ManagerID	Gender	Description
Ahmed	1	2	2	M	xyz
New Name	2	1	2	M	changed with For update trigger
Qasim	3	1	2	M	sadsff

Now here only the description of row with Employee ID 2 is changed

II. 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
```

To fire the trigger we can insert a row in table and it will show list of all user instead of inserting into the table

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

Output:

Results Messages

	Name	EmployeeID	ContactID	ManagerID	Gender	Description
1	Ahmed	1	2	2	M	xyz
2	Osama	2	1	2	M	sadsd
3	Qasim	3	1	2	M	sadsff

Update and delete triggers can be made using similar way as well.

b. 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

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
   PRINT 'You must disable Trigger "safety" to drop or alter tables!'
   ROLLBACK
;
```

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

DROP TABLE [Trigger_Project]

Another Example of DDL triggers , whenever alter table query is executed it will add a column “NAME” to the table defined in trigger.

```
-----------------------------------------------------------------
---- trigger for allowing alteration of table with column data type
-----------------------------------------------------------------
CREATE TRIGGER [T_AlterTable_DT]
ON  DatabASE
FOR ALTER_TABLE
AS
    PRINT 'CAN BE ALTERED'

ALTER TABLE [Trigger_Project] ALTER COLUMN [NAME] VARCHAR(60)
```

3. Exercise

Exercise:

Keeping in view the current issues of neon, we are required to redesign the neon system, in which we are keeping track of student and their departments. Along with that we need to maintain the enrollment i.e. a student is enrolled in a particular section of a course. So also need to keep track of all the courses offered by the university along with the respective sections. Students will enroll in sections of a particular course.

Follow the schema specified in **lab.txt** file. Create a database with given information. Insert some rows in each table and do the below questions.

1. The academic 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 a column **audit_id** and **Last_change_on**. Create triggers on student, department and faculty tables so that whenever any change is made on these tables the dates of change get stored in the Auditing table.
2. Though academic officer was pretty much satisfied with your last change but after few months it feels something is missing in auditing table so you are asked to change the structure of the audit table. So now you also have to store the description of changes i.e. which table from the above-mentioned tables was changed. So, make appropriate change in triggers to perform the operation.
3. Now focusing more on the registration system, we firstly need to have some sort of security. So, in order to have some level of abstraction in our course and section tables we need to design a view, so that when students query for enrollment i.e. applying for the registration of a particular section you should be displaying data from that view rather applying queries on multiple tables (course and section).
4. Moreover, we need to have a store procedure of the above functionality in order to reduce the network traffic and faster execution.
5. The academic officer also wants to make sure no one insert, update or deletes department information from the database. As university has only 3 departments CS, EE and MG and that are already present in database. So, create a trigger that will not let anyone to change the department table.