National University of Computer and Emerging Sciences

Lab Manual 8

"Triggers"

Database Systems

Spring 2022

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

Objectives

The purpose this lab is to know how the triggers work, types of triggers, how to create a trigger and what are the uses of triggers.

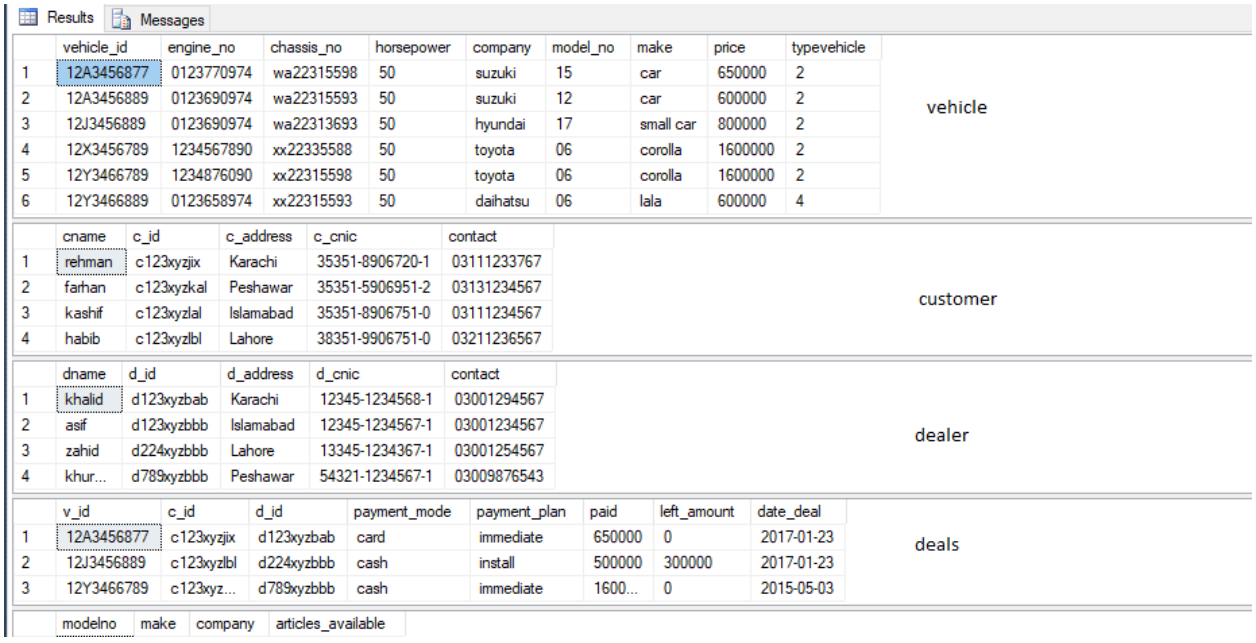
Triggers

Triggers are special kind of stored procedures that automatically execute when a DML or DDL statement associated with the trigger is executed. Each trigger will be associated with one DML or DDL statement. Unlike stored procedure triggers cannot be executed directly by application/user, they will ONLY be executed by DBMS in reaction to DML or DDL statement with which the trigger was associated.

Triggers can be divided in two categories depending on the type of statement they are associated with as follow:

- DML triggers
- DDL triggers

We shall use the schema from the last lab's manual. Which is as follows.



Results | Messages

vehicle

	vehicle_id	engine_no	chassis_no	horsepower	company	model_no	make	price	typevehicle
1	12A3456877	0123770974	wa22315598	50	suzuki	15	car	650000	2
2	12A3456889	0123690974	wa22315593	50	suzuki	12	car	600000	2
3	12J3456889	0123690974	wa22313693	50	hyundai	17	small car	800000	2
4	12X3456789	1234567890	xx22335588	50	toyota	06	corolla	1600000	2
5	12Y3466789	1234876090	xx22315598	50	toyota	06	corolla	1600000	2
6	12Y3466889	0123658974	xx22315593	50	daihatsu	06	lala	600000	4

customer

	cname	c_id	c_address	c_cnic	contact
1	rehman	c123xyzjix	Karachi	35351-8906720-1	03111233767
2	farhan	c123xyzkal	Peshawar	35351-5906951-2	03131234567
3	kashif	c123xyzlal	Islamabad	35351-8906751-0	03111234567
4	habib	c123xyzlbl	Lahore	38351-9906751-0	03211236567

dealer

	dname	d_id	d_address	d_cnic	contact
1	khalid	d123xyzbab	Karachi	12345-1234568-1	03001294567
2	asif	d123xyzbbb	Islamabad	12345-1234567-1	03001234567
3	zahid	d224xyzbbb	Lahore	13345-1234367-1	03001254567
4	khur...	d789xyzbbb	Peshawar	54321-1234567-1	03009876543

deals

	v_id	c_id	d_id	payment_mode	payment_plan	paid	left_amount	date_deal
1	12A3456877	c123xyzjix	d123xyzbab	card	immediate	650000	0	2017-01-23
2	12J3456889	c123xyzlbl	d224xyzbbb	cash	install	500000	300000	2017-01-23
3	12Y3466789	c123xyz...	d789xyzbbb	cash	immediate	1600...	0	2015-05-03

	modelno	make	company	articles_available

	v_id	c_id	d_id	payment_mode	payment_plan	paid	left_amount	date_deal
1	12A3456877	c123xyzjix	d123xyzbab	card	immediate	650000	0	2017-01-23
2	12J3456889	c123xyzlbl	d224xyzbbb	cash	install	500000	300000	2017-01-23
3	12Y3466789	c123xyz...	d789xyzbbb	cash	immediate	1600...	0	2015-05-03

Deals

	modelno	make	company	articles_available
1	12	car	suzuki	35
2	15	car	suzuki	20
3	17	sm...	hyundai	3
4	6	lala	daihatsu	0

Inventory

	c_id	d_id	make	company	model	dateorder	status_order	date_completeion
1	c123xyzjix	d123xyzbab	car	suzuki	12	2017-10-02	1	2017-10-02
2	c123xyzlbl	d224xyzbbb	lala	daihatsu	6	2017-09-02	0	2017-09-02

Orders

DML Triggers:

DML is the data modification language that uses queries like INSERT, UPDATE, DELETE. The DML triggers are used to handle these kind of queries.

Syntax of (DML) Triggers:

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ (FOR | AFTER) | Instead of }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS { sql_statement  [ ; ] [ ,...n ] }

<dml_trigger_option> ::=
    [ NATIVE_COMPILATION ]
    [ SCHEMABINDING ]
    [ EXECUTE AS Clause ]
```

‘Instead of’ and ‘After’ Triggers:

You must have noticed the word after and instead of in the syntax of the trigger creation given above.

The instead of trigger is shot by stopping the action on which the trigger is created. However, the after (or for) trigger is shot after the action on which the trigger is created is complete.

DML trigger Option:

The DML trigger option given in the above syntax is used to specify the action on which the trigger is to be shot.

Now let us look at the triggers a little closely how they practically work.

```
go
create trigger the_trigger on customer
after insert
as begin
print 'your data has been inserted'
end
```

The trigger has been made as to be shot on the insert query on the customer table. Now if we insert anything in the customer table as soon as a new customer has been added in the table this trigger shall be shot.

```
select * from customer
insert into customer (cname, c_id, c_address, c_cnic, contact) values('talha', 'c168fghjas', 'Lalamusa', '35876-0987654-2','03213221234')
```

```
%
Messages
your data has been inserted

(1 row(s) affected)
```

In the above example you can clearly see that we have just inserted a new entry in the table customer but the trigger has been shot that too after the trigger has been shot.

```
go
create trigger the_del_trig
on customer
instead of delete
as begin
print'you can not delete this data'
end
go
```

```
100 %
Messages
Command(s) completed successfully.
```

Getting the affected tables inside the triggers:

The triggers we have seen above are simple one, what if you want the value of effect rows from DML and use them in triggers.

Example: Whenever a customer is inserted in, it should automatically convert that name of that instructor in Upper Case.

For example: customer with Name "ali ahmed" should be inserted as "ALI AHMED"

For that we use special table "DELETED" and "INSERTED" designed for DML triggers.

DML triggers use the **deleted** and **inserted** logical (conceptual) tables. They are structurally similar to the table on which the trigger is defined, that is, the table on which the user action is tried. The **deleted** and **inserted** tables hold the old values or new values of the rows that may be changed by the DML action.

NOTE: These tables are only accessible in triggers

1. For Trigger or After Trigger:

A For or After trigger is fired after the intended action takes place.

1.1 Insert Trigger

A for trigger defined on a table for insert operation is fired after the insert statement is executed. Consider the following For trigger for insert operation on Customers Table.

```
Create Trigger ForTriggerForInsertCustomer
On Customers
For Insert
AS
Begin
declare @customerNo varchar(2),
              @name varchar(30),
              @city varchar(3),
              @phone varchar(11)

Select @customerNo= Inserted.customerNo, @name=Inserted.name, @city=Inserted.city,
       @phone= Inserted.phone
From Inserted

print ('New Customer Inserted whose customerNo is '+@customerNo+'. Name is '+@name+'. City is
'+@city+'. Phone is '+@phone+'.')
End
```

The above trigger will be called after the insert operation on customers. The insert operation will be carried out as usual and after that the trigger will be called.

```
Insert into Customers values('C0', 'Ali', 'Lhr', '12345')
```

When the above insert statement is executed, we get the following output:

```
New Customer Inserted whose customerNo isC0. Name is Ali. City is Lhr. Phone is 12345.

(1 row affected)
```

1.2 Delete Trigger

A for trigger defined on a table for delete operation is fired when the delete statement is executed. Consider the following For trigger for delete operation on Customers Table:

```
Create Trigger ForTriggerForDeleteCustomer
On Customers
For Delete
AS
Begin
declare @customerNo varchar(2),
              @name varchar(30),
              @city varchar(3),
              @phone varchar(11)

Select @customerNo= Deleted.customerNo,
       @name=Deleted.name,
          @city=Deleted.city,
          @phone= Deleted.phone
          From Deleted

print ('A Customer deleted whose customerNo is '+@customerNo+'. Name is '+@name+'. City
is '+@city+'. Phone is '+@phone+'.
')
End
```

The above trigger will be called after the delete operation on customers. The delete operation will be carried out as usual and after that the trigger will be called.

When the following delete operation is called:

```
Delete From Customers
where CustomerNo='C0'
```

We get the output as follows:

```
A Customer deleted whose customerNo is C0. Name is Ali. City is Lhr. Phone is 12345.

(1 row affected)
```

1.3 Update Trigger

A for trigger defined on a table for update operation is fired when the update statement is executed. Consider the following For trigger for update operation on Customers Table:

```
Create Trigger ForTriggerForUpdateCustomer
On Customers
For Update
AS
Begin
declare @oldCustomerNo varchar(2),
              @oldName varchar(30),
              @oldCity varchar(3),
              @oldPhone varchar(11),

              @newCustomerNo varchar(2),
              @newName varchar(30),
              @newCity varchar(3),
              @newPhone varchar(11)

Select @oldCustomerNo= Deleted.customerNo,
       @oldName=Deleted.name,
          @oldCity=Deleted.city,
          @oldPhone= Deleted.phone
          From Deleted

Select @newCustomerNo= Inserted.customerNo,
       @newName=Inserted.name,
          @newCity=Inserted.city,
          @newPhone= Inserted.phone
          From Inserted

declare @messageString varchar(100)
set @messageString='The record of a customer updated. The fields that have been updated
are: '

       if @oldCustomerNo!=@newCustomerNo
       Begin
       set @messageString=@messageString+'Customer No,'
       End

       if @oldName!= @newName
       Begin
       set @messageString= @messageString+'Name, '
       End

       if @oldPhone!= @newPhone
       Begin
       set @messageString= @messageString+'Phone, '
       End

       if @oldCity!= @newCity
       Begin
       set @messageString= @messageString+'City'
       End

print (@messageString)
End
```

When the following update statement is run:

```
Update Customers
```

```
set Name='Imran', city='Khr'
where CustomerNo='C2'
```

We get the output as follows:

```
The record of a customer updated. The fields that have been updated are: Name, City

(1 row affected)
```

2. Instead of Trigger

AS the name suggest, this type of trigger, when defined on insert, update, or delete operation, is called instead of the insert, update, and delete operation itself. So if we want the original insert, update or delete operation to be carried out successfully, we must do it in the trigger.

2.1 Insert Trigger

```
Create Trigger InsteadTriggerForInsertCustomer
On Customers
Instead of Insert
AS
Begin
declare @customerNo varchar(2),
              @name varchar(30),
              @city varchar(3),
              @phone varchar(11)

Select @customerNo= Inserted.customerNo, @name=Inserted.name, @city=Inserted.city,
       @phone= Inserted.phone
From Inserted

Insert into Customers values (@customerNo, @name, @city, @phone)

print ('New Customer Inserted (by the trigger) whose customerNo is '+@customerNo+'. Name
is '+@name+'. City is '+@city+'. Phone is '+@phone+'.')
End
```

When we run the following insert statement:

```
Insert into Customers values ('C0', 'Ali', 'Khr', '01234')
```

The output we get is as follows:

```
(1 row affected)
New Customer Inserted (by the trigger) whose customerNo isC0. Name is Ali. City is Khr. Phone is 01234.

(1 row affected)
```

2.2 Delete Trigger

```
create Trigger InsteadTriggerForDeleteCustomer
On Customers
Instead of Delete
AS
```

```
Begin
declare @customerNo varchar(2),
              @name varchar(30),
              @city varchar(3),
              @phone varchar(11)

Select @customerNo= Deleted.customerNo,
       @name=Deleted.name,
          @city=Deleted.city,
          @phone= Deleted.phone
          From Deleted


          delete from Customers
          where CustomerNo=@customerNo

print('Customer whose customer id is '+@customerNo+' deleted by trigger sucessfully')
End
```

When we run the following statement:

```
Delete From Customers
where CustomerNo='C2'
```

The output we get is as follows (The actual insert will be carried out by trigger itself):

```
(1 row affected)
Customer whose customer id is C2 deleted by trigger sucessfully

(1 row affected)
```

2.3 Update Trigger

```
Create Trigger InsteadTriggerForUpdateCustomer
On Customers
Instead of Update
AS
Begin
declare @oldCustomerNo varchar(2),
              @oldName varchar(30),
              @oldCity varchar(3),
              @oldPhone varchar(11),

              @newCustomerNo varchar(2),
              @newName varchar(30),
              @newCity varchar(3),
              @newPhone varchar(11)

Select @oldCustomerNo= Deleted.customerNo,
       @oldName=Deleted.name,
          @oldCity=Deleted.city,
          @oldPhone= Deleted.phone
          From Deleted

Select @newCustomerNo= Inserted.customerNo,
```

```
        @newName=Inserted.name,
           @newCity=Inserted.city,
           @newPhone= Inserted.phone
           From Inserted

           update Customers
           set CustomerNo=@newCustomerNo, Name=@newName, city= @newCity, Phone= @newPhone
           where CustomerNo= @oldCustomerNo

declare @messageString varchar(100)
set @messageString='The record of a customer updated by the trigger. The fields that have
been updated are: '

        if @oldCustomerNo!=@newCustomerNo
        Begin
        set @messageString=@messageString+'Customer No,'
        End

        if @oldName!= @newName
        Begin
        set @messageString= @messageString+'Name, '
        End

        if @oldPhone!= @newPhone
        Begin
        set @messageString= @messageString+'Phone, '
        End

        if @oldCity!= @newCity
        Begin
        set @messageString= @messageString+'City'
        End

print (@messageString)
End
```

When the following update statement is called:

```
Update Customers
set Name='Imran', city='Khr'
where CustomerNo='C2'
```

We get the output as follows (The actual update will be done by the trigger itself)

```
(0 rows affected)
The record of a customer updated by the trigger. The fields that have been updated are:

(0 rows affected)
```

3. Rollback in For Trigger

If you do not want to carry out the insert, update or delete operation due to any reason. Then you can either write an instead trigger which will do the insert, update or delete operation only when the conditions are met. Alternatively, you can do rollback in for trigger if the conditions are not met.

```
Create Trigger ForTriggerOnInsertCustomer
on Customers
For Insert
AS
Begin
declare @name varchar(10)

Select @name=name from Inserted

if @name is NULL
Begin
rollback
End

End
```

The above trigger does not allow inserting the customer whose name is null. So, when the following insert statement is run, the insert operation will be rolled back by the trigger:

```
Insert into Customers values('C0', NULL, 'Lhr', '123')
```

We get the output as follows:

```
Msg 3609, Level 16, State 1, Line 129
The transaction ended in the trigger. The batch has been aborted.
```

4. Same Trigger For All three Actions (Insert, Update and Delete)

So far we have seen triggers on a single action a single trigger can be used to cater multiple actions.

Before moving forward you must know that only one trigger can be made on a particular action on a table so in order to make a new trigger on the same action we should either alter the previous trigger or drop the previous trigger or disable it. The syntax for these are as follows.

```
ALTER <TriggerName>

On <view/table>

After/Instead of  <insert/update/delete>

As

begin

      <Body>

end
```

```
Create Trigger DisallowChangeOnCustomers
On Customers
```

```
Instead of Insert, Update, Delete
As
Begin
print ('Customer table is not allowed to be modified')
End
```

5. Drop, Enable and Disable a Trigger

```
Drop trigger <TriggerName>

Enable trigger <TriggerName> on <ObjectName>

Disable trigger <TriggerName> on <ObjectName>
```

```
disable trigger InsteadTriggerForInsertCustomer on Customers
```

6. DDL Triggers

DDL triggers, fire in response to a DDL statement to which they are associated. DDL event primarily correspond to SQL statements that start with the keywords CREATE, ALTER, and DROP. These triggers are current databases.

There triggers are also of two types, FOR and AFTER, first one executes instead of the DDL statement it is associated with and second one executes after the DDL statement, it is associate with is successfully executed.

(For in DML FOR is same as Instead of in DDL)

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.

Syntax Of DDL

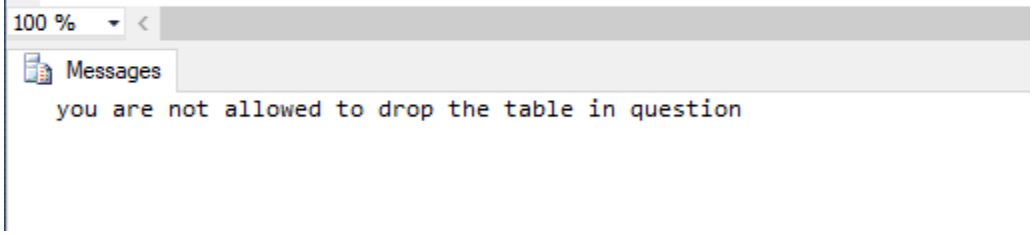
```
CREATE TRIGGER trigger_name

ON DATABASE

{ FOR | AFTER } { event_type} [ ,...n ]

AS

Begin
```

```
        <Body>
End
```

Triggers:

```
go
create trigger ddl_trig
on database
for
drop_table
as begin
print'you are not allowed to drop the table in question'
end
```


```
drop table deals
```



References:

https://msdn.microsoft.com/en-us/library/bb522542.aspx