T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ

COM5041 - DATABASE DESIGN AND DEVELOPMENT

LAB 08 – Data manipulation triggers and Stored procedures

After completing this Lab, you will be able to

- Explain the different types of triggers
- Create, alter, and drop triggers
- Explain practical uses of triggers
- Understand stored procedures
- Work with stored procedures
- Create, alter, and delete stored procedures
- Use different types of parameters with stored procedures

PROCEDURE 1 – How to create triggers.

Data manipulation triggers are sets of T-SQL statements that perform a specific action. They are often referred to as a special kind of stored procedure. Unlike stored procedures, triggers are executed only when a user or application attempts to modify data using Data Manipulation Language (DML). DML includes INSERTs, UPDATEs, and DELETEs against views and tables.

The **CREATE TRIGGER** statement allows you to create a new trigger that is fired automatically whenever an event such as INSERT, DELETE, or UPDATE occurs against a table.

You can use the following pseudocode sample script for creating a DML trigger:

```
CREATE TRIGGER [schema_name. ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
<dml_trigger_option> ::=
[ ENCRYPTION ]
[ EXECUTE AS Clause ]
<method_specifier> ::=
assembly_name.class_name.method_name
```

WITH <Options>: In this argument you can specify additional options for the creation of the trigger.

FOR or AFTER [INSERT, UPDATE, DELETE]: These types of triggers are executed after the firing statement ends (either an insert, update or delete).

INSTEAD OF [INSERT, UPDATE, DELETE]: Contrary to the FOR (AFTER) type, the INSTEAD OF triggers executes instead of the firing statement. In other words, this type of trigger replaces the firing statement. This is very useful in cases where you need to have cross database referential integrity.

[INSERT], [UPDATE], [DELETE]: The DML event (or list of events) that will cause the trigger to fire.

ENCRYPTION: Encrypts the code of the Trigger.

EXECUTE AS: Changes the security context on which the trigger will execute.

Note that; Constraints, including foreign-key cascade operations, are checked prior to executing either type of trigger. If it is an INSTEAD OF trigger, the constraint is checked after the trigger completes. If it is an AFTER trigger, the constraint is checked prior to executing the trigger. Regardless of the type of trigger, when the trigger event is rolled back, it causes a constraint violation. For INSTEAD OF triggers, the entire event is undone or rolled back, and in the case of FOR triggers, nothing is executed.

Step 1 - Create a trigger that checks the modified date. The trigger ensures that during the insert of a new department, the modified date is the current day. If it is not, the row is updated, setting ModifiedDate to the current date and time.

```
USE AdventureWorks2019;
GO
CREATE TRIGGER HumanResources.iCheckModifiedDate
ON HumanResources.Department
FOR INSERT
AS
BEGIN
DECLARE @modifieddate datetime, @DepartmentID int
SELECT @modifieddate = modifieddate, @DepartmentID = departmentid FROM inserted;
IF(DATEDIFF(Day, @modifieddate, getdate()) > 0)
BEGIN
UPDATE HumanResources.Department
SET ModifiedDate = GETDATE()
WHERE DepartmentID = @DepartmentID
END
END
```

Note that; In the code of the trigger, specifically the SELECT statement, a logical table called inserted is referenced. There are actually two tables of this type - the second table is deleted. These tables are available within the context of the trigger. You cannot modify the structure or contents of these tables. The inserted table stores a copy of a row or rows that were inserted, or a copy of the new values for rows that were updated. During an update, the inserted table stores the new or updated data. The second (deleted) table stores the data before an update and a row or rows that were deleted in a DELETE statement.

Step 2 – Let's insert a new data in Department table.

```
INSERT INTO HumanResources.Department
VALUES('Executive Marketing', 'Executive General and Administration', '2021-11-21');
```

Step 3 - Execute the query and review the results.

```
SELECT * FROM HumanResources.Department
```

Results | Messages

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research and Development	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production Control	Manufacturing	2008-04-30 00:00:00.000
9	9	Human Resources	Executive General and Ad...	2008-04-30 00:00:00.000
10	10	Finance	Executive General and Ad...	2008-04-30 00:00:00.000
11	11	Information Services	Executive General and Ad...	2008-04-30 00:00:00.000
12	12	Document Control	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality Assurance	Quality Assurance	2008-04-30 00:00:00.000
14	14	Facilities and Maintenance	Executive General and Ad...	2008-04-30 00:00:00.000
15	15	Shipping and Receiving	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and Ad...	2008-04-30 00:00:00.000
17	17	Executive Marketing	Executive General and Ad...	2023-11-28 22:40:40.603

** In the last row of the result set, you will see the newly inserted row. Notice that the modified date is not the value that was specified in the insert. It should be the current date and time of the insert.

PROCEDURE 2 – How to alter triggers.

ALTER TRIGGER changes properties of an existing trigger. If you need to change the code of a single trigger maybe the easiest way to do so is by using the ALTER TRIGGER statement.

You can use the following pseudocode sample script for altering a DML trigger:

```
ALTER TRIGGER schema_name.trigger_name
ON  ( table | view )
[ WITH <dml_trigger_option> [ ,...n ] ]
 ( FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier>
[ ; ] }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ <EXECUTE AS Clause> ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Step 1 - Alter the HumanResources.iCheckModifedDate trigger that checks the modified date. The trigger ensures that during the insert of a new department, the modified date is the current day. If it is not, the row is updated, setting ModifiedDate to the yesterday's date and time.

```
USE AdventureWorks2019;
GO
ALTER TRIGGER HumanResources.iCheckModifiedDate
ON HumanResources.Department
FOR INSERT
AS
BEGIN
DECLARE @modifieddate datetime, @DepartmentID int
SELECT @modifieddate = modifieddate, @DepartmentID = departmentid FROM inserted;
IF(DATEDIFF(Day, @modifieddate, getdate()) > 0)
BEGIN
UPDATE HumanResources.Department
SET ModifiedDate = DATEADD(day, -1, GETDATE() )
WHERE DepartmentID = @DepartmentID
END
END
```

Step 2 - Let's insert a new data in Department table.

```
INSERT INTO HumanResources.Department
VALUES('Executive Purchasing', 'Executive General and Administration', '2021-11-21');
```

Step 3 - Execute the query and review the results.

```
SELECT * FROM HumanResources.Department
```

Results | Messages

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research ...	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production...	Manufacturing	2008-04-30 00:00:00.000
9	9	Human R...	Executive General and Ad...	2008-04-30 00:00:00.000
10	10	Finance	Executive General and Ad...	2008-04-30 00:00:00.000
11	11	Informatio...	Executive General and Ad...	2008-04-30 00:00:00.000
12	12	Document...	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality Ass...	Quality Assurance	2008-04-30 00:00:00.000
14	14	Facilities a...	Executive General and Ad...	2008-04-30 00:00:00.000
15	15	Shipping a...	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and Ad...	2008-04-30 00:00:00.000
17	17	Executive ...	Executive General and Ad...	2023-11-28 22:40:40.603
18	18	Executive ...	Executive General and Ad...	2023-11-27 22:44:18.670

** In the last row of the results, you should see a modified date that is set to yesterday's date.

PROCEDURE 3 – How to enable, disable or drop triggers.

Step 1 - In some cases, you may not want to delete a trigger, but you want to stop it from firing during a large. DML operation or for testing purposes SQL Server provides you with the ability to disable a trigger, and once you have completed the task, you can enable the trigger again.

- To disable or enable a trigger using T-SQL, use the following code:

```
DISABLE TRIGGER HumanResources.iCheckModifiedDate
ON HumanResources.Department;
```

- To enable a trigger using T-SQL, use the following code:

```
ENABLE TRIGGER HumanResources.iCheckModifiedDate
ON HumanResources.Department;
```

Step 2 - The SQL Server **DROP TRIGGER** statement drops one or more triggers from the database.

You can use the following pseudocode sample script for removing a DML trigger:

```
DROP TRIGGER [ IF EXISTS ] [schema_name.]trigger_name [ ,...n ] [ ; ]
```

- We can delete the HumanResources.iCheckModifedDate trigger using the Drop Trigger statement.

```
DROP TRIGGER HumanResources.iCheckModifiedDate
```

PROCEDURE 4 – Working with stored procedures.

Stored procedures are a set of SQL statements (one or more) typically grouped together to perform a specific routine. Stored procedures can be created in any user-defined database and system database except the resource database. Some of the benefits of using stored procedures are as follows:

- They offer improved performance because of compiled code.
- They are easy to maintain because changes are central instead of inline with code.
- Since database operations can be performed inside the stored procedures, they provide a strong level of security. Instead of access being granted to the underlying object, permission can be granted only to the stored procedure. Essentially, stored procedures create a level of abstraction for permissions—instead of the user being granted SELECT, INSERT, UPDATE, or DELETE rights, the user can be granted EXECUTE rights to a stored procedure.

The basic syntax for creating a stored procedure is follows:

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
[ { @parameter [ type_schema_name. ] data_type }
[ VARYING ] [ = default ] [ OUT | OUTPUT ] [READONLY]
] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
<procedure_option> ::=
[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE AS Clause ]
```

Step 1 - Let's create a simple store procedure that returns the **PurchaseOrderID**, **PurchaseOrderDetailID**, **OrderDate**, **TotalDue**, **ReceivedQty** and **ProductName** information of the products sold.

```
CREATE PROCEDURE sp_PurchaseOrderInformation
AS
BEGIN
SELECT
poh.PurchaseOrderID, pod.PurchaseOrderDetailID,
poh.OrderDate, poh.TotalDue, pod.ReceivedQty, p.Name ProductName
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.PurchaseOrderDetail pod
ON poh.PurchaseOrderID = pod.PurchaseOrderID
INNER JOIN Production.Product p
ON pod.ProductID = p.ProductID
END
GO
```

Step 2 - To execute a stored procedure using T-SQL, you use the **EXECUTE** or **EXEC** keyword.

```
exec sp_PurchaseOrderInformation;
```

Results | Messages

	PurchaseOrderID	PurchaseOrderDetailID	OrderDate	TotalDue	ReceivedQty	ProductName
1	1	1	2011-04-16 00:00:00.000	222,1492	3.00	Adjustable Race
2	2	2	2011-04-16 00:00:00.000	300,6721	3.00	Thin-Jam Hex Nut 9
3	2	3	2011-04-16 00:00:00.000	300,6721	3.00	Thin-Jam Hex Nut 10
4	3	4	2011-04-16 00:00:00.000	9776,2665	550.00	Seat Post
5	4	5	2011-04-16 00:00:00.000	189,0395	2.00	Headset Ball Bearings
6	5	6	2011-04-30 00:00:00.000	22539,0165	550.00	HL Road Rim
7	6	7	2011-04-30 00:00:00.000	16164,0229	468.00	Touring Rim
8	7	8	2011-04-30 00:00:00.000	64847,5328	550.00	LL Crankarm
9	7	9	2011-04-30 00:00:00.000	64847,5328	550.00	ML Crankarm
10	7	10	2011-04-30 00:00:00.000	64847,5328	550.00	HL Crankarm

Step 3 - To change the result set, you issue the **EXECUTE** command as normal, but you add a **WITH RESULT SETS** statement, and within the parentheses you provide a new column definition for each column in the result set.

```
EXEC sp_PurchaseOrderInformation
WITH RESULT SETS
(
(
[Purchase Order ID] int,
[Purchase Order Detail ID] int,
[Order Date] datetime,
[Total Due] Money,
[Received Quantity] float,
[Product Name] varchar(50)
)
)
```

Results | Messages

	Purchase Order ID	Purchase Order Detail ID	Order Date	Total Due	Received Quantity	Product Name
1	1	1	2011-04-16 00:00:00.000	222,1492	3	Adjustable Race
2	2	2	2011-04-16 00:00:00.000	300,6721	3	Thin-Jam Hex Nut 9
3	2	3	2011-04-16 00:00:00.000	300,6721	3	Thin-Jam Hex Nut 10
4	3	4	2011-04-16 00:00:00.000	9776,2665	550	Seat Post
5	4	5	2011-04-16 00:00:00.000	189,0395	2	Headset Ball Bearings
6	5	6	2011-04-30 00:00:00.000	22539,0165	550	HL Road Rim
7	6	7	2011-04-30 00:00:00.000	16164,0229	468	Touring Rim
8	7	8	2011-04-30 00:00:00.000	64847,5328	550	LL Crankarm
9	7	9	2011-04-30 00:00:00.000	64847,5328	550	ML Crankarm
10	7	10	2011-04-30 00:00:00.000	64847,5328	550	HL Crankarm

** In the preceding query, the EXEC keyword is used for a standard stored procedure execution However, the columns and data types have been changed to make the column names user-friendly

Step 4 - Stored procedures can include parameters as part of their code. Creating a stored procedure with parameters allows the calling programs to pass values into the procedure. If your stored procedure contains a default parameter, you are not required to supply the DEFAULT keyword. If you simply execute the stored procedure using the EXEC keyword, the assigned value will be used at run time.

```
--Create Proc with OUTPUT param
CREATE PROC dbo.SampleOutput
@Parameter2 int OUTPUT
as
SELECT @Parameter2 = 10

--Execute Proc with OUTPUT param
DECLARE @HoldParameter2 INT
EXEC dbo.SampleOutput
@HoldParameter2 OUTPUT
SELECT @HoldParameter2
```

Results | Messages

	(No column name)
1	10

** In the first part of the SQL code, a stored procedure is created that has a single output parameter. The output parameter includes the **OUTPUT** keyword. The stored procedure contains a single T-SQL statement that assigns 10 to the parameter. In the second part of the code, a variable is declared that will hold the value of the output parameter. The **EXEC** keyword is used to execute the stored procedure. In addition, the declared variable is specified including the **OUTPUT** keyword. Finally, a **SELECT** statement is issued to display the value of the output.

PROCEDURE 5 – How to alter stored procedures.

Step 1 –We will query the Purchasing.PurchaseOrderHeader, Purchasing.PurchaseOrderDetail and Production.Product tables from the AdventureWorks database, but instead of getting back all records we will limit it to just a particular EmployeeID and OrderYear.

```
ALTER PROCEDURE sp_PurchaseOrderInformation
@EmployeeID int,
@OrderYear int = 2011
AS
BEGIN
SELECT
poh.PurchaseOrderID,
pod.PurchaseOrderDetailID,
poh.OrderDate,
poh.TotalDue,
pod.ReceivedQty,
p.Name ProductName
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.PurchaseOrderDetail pod
ON poh.PurchaseOrderID = pod.PurchaseOrderID
INNER JOIN Production.Product p
ON pod.ProductID = p.ProductID
WHERE
poh.EmployeeID = @EmployeeID AND
YEAR(poh.OrderDate) = @OrderYear
END
```

** In the preceding stored procedure, two parameters were specified: input and default. At the end of the T-SQL query, a WHERE clause was included that limits the result based on the values of the two parameters.

Step 2 – To execute this stored procedure, only the input parameter has been specified, which is fine because the other parameter has a default value. To call this stored procedure we can execute it as follows:

```
EXEC sp_PurchaseOrderInformation
@EmployeeID = 258;
```

Results | Messages

	PurchaseOrderID	PurchaseOrderDetailID	OrderDate	TotalDue	ReceivedQty	ProductName
1	1	1	2011-04-16 00:00:00.000	222,1492	3.00	Adjustable Race
2	11	24	2011-12-14 00:00:00.000	553,8221	3.00	Lock Nut 5
3	11	25	2011-12-14 00:00:00.000	553,8221	3.00	Lock Nut 6
4	11	26	2011-12-14 00:00:00.000	553,8221	3.00	Lock Nut 16
5	11	27	2011-12-14 00:00:00.000	553,8221	3.00	Lock Nut 17

Step 3 - In the previous code, the default value for the OrderYear parameter has been overwritten with 2012. To call this stored procedure we can execute it as follows:

```
EXEC sp_PurchaseOrderInformation
@EmployeeID = 258,
@OrderYear = 2012;
```

Results | Messages

	PurchaseOrderID	PurchaseOrderDetailID	OrderDate	TotalDue	ReceivedQty	ProductName
1	31	77	2012-01-08 00:00:00.000	157,3647	3.00	Keyed Washer
2	41	95	2012-01-16 00:00:00.000	24880,9811	550.00	HL Mountain Seat/Saddle
3	51	116	2012-01-20 00:00:00.000	108,2513	3.00	LL Nipple
4	51	117	2012-01-20 00:00:00.000	108,2513	3.00	HL Nipple
5	61	136	2012-01-24 00:00:00.000	560,3312	3.00	External Lock Washer 3

PROCEDURE 6 – How to drop stored procedures.

Step 1 – To delete a stored procedure, you can use the **DROP PROCEDURE** statement

```
DROP PROCEDURE sp_PurchaseOrderInformation
```

SUMMARY

In this LAB, you learned about triggers, which are associated to a specific table or view. In the event of a DML operation on that table or view, the trigger will execute, performing some action. There are three types of triggers: AFTER, INSTEAD OF, and FOR. This chapter focused on the AFTER and INSTEAD OF triggers and their capabilities. Stored procedures are among the most robust programming tools available in the Microsoft SQL Server database. As you learned in this LAB, you can use stored procedures in several ways to accomplish many things.