Joins in SQL Server

The SQL Server Joins are used to retrieve the data from two or more related tables. In general, tables are related to each other using the primary key and foreign key relationship but it is not mandatory. The tables involved in the joins must have a common field. And based on that common field the SQL Server JOINS retrieves the records.

Again the ANSI format joins classified into three types such as
1.Inner join
2.Outer join
3.Cross join

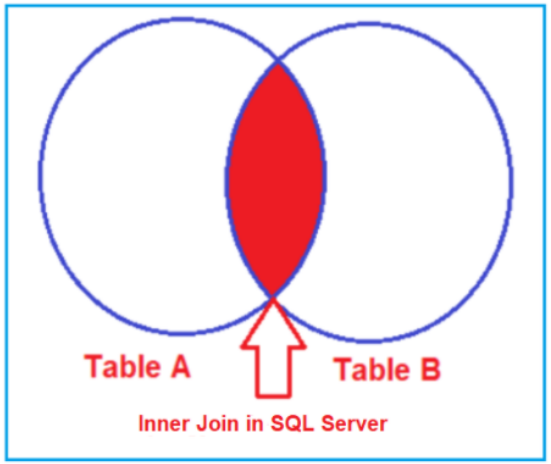
Further, the outer join is divided into three types are as follows
1.Left outer join
2.Right outer join
3.Full outer join

Inner Join in SQL Server

The Inner Join in SQL Server is used to return only the matching rows from both the tables involved in the join by removing the non-matching records. The following diagram shows the pictorial representation of SQL Server Inner Join.

```
Syntax : -
Select * from  table1 A inner join table2 B
A.ColumnName=B.ColumnName

Example - SELECT Id as EmployeeID, Name, Department, City, Title as Project,
ClientId FROM Employee INNER JOIN Projects
ON Employee.Id = Projects.EmployeeId;
```



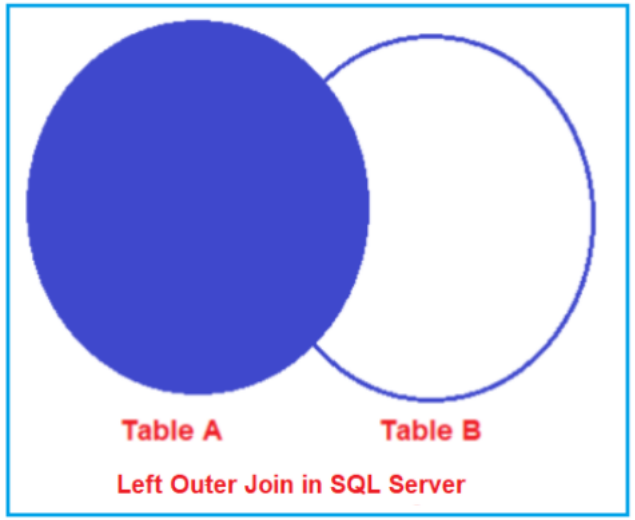
Left Outer Join in SQL Server

The LEFT OUTER JOIN in SQL Server is used to retrieve all the matching rows from both the tables involved in the join as well as non-matching rows from the left side table. In this case, the un-matching data will take a null value.\

The question that should come to your mind is, which is the left table and which is the right table? The answer is, the table mentioned to the left of the LEFT OUTER JOIN keyword is the left table, and the table mentioned to the right of the LEFT OUTER JOIN keyword is the right table. The following diagram is the pictorial representation of SQL Server Left Outer Join.

```
Syntax : -
Select * from  table1 A left join table2 B
A.ColumnName=B.ColumnName
```

```
Example - SELECT Id as EmployeeID, Name, Department, City, Title as Project, ClientId FROM Employee LEFT
JOIN Projects
ON Employee.Id = Projects.EmployeeId;
```



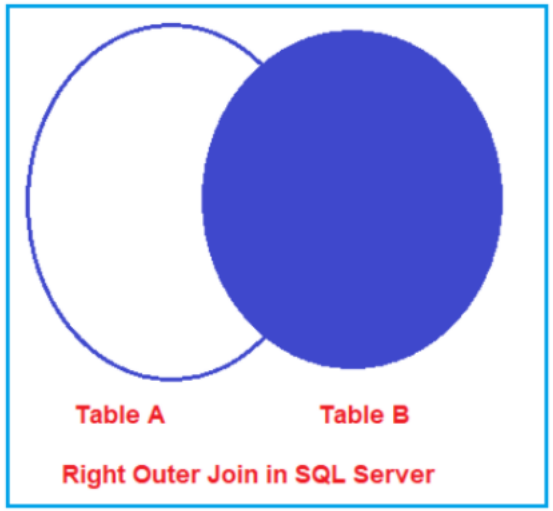
Right Outer Join in SQL Server

The RIGHT OUTER JOIN in SQL Server is used to retrieve all the matching rows from both the tables involved in the join as well as non-matching rows from the right-side table. In this case, the un-matching data will take NULL values.

The question that should come to your mind is, which is the left table and which is the right table? The answer is, the table mentioned to the left of the RIGHT OUTER JOIN keyword is the left table, and the table mentioned to the right of the RIGHT OUTER JOIN keyword is the right table. The following diagram is the pictorial representation of SQL Server Right Outer Join.

```
Syntax : -
Select * from  table1 A right join table2 B
A.ColumnName=B.ColumnName
```

```
Example - SELECT Id as EmployeeID, Name, Department, City, Title as Project, ClientId FROM Employee right
JOIN Projects
ON Employee.Id = Projects.EmployeeId;
```

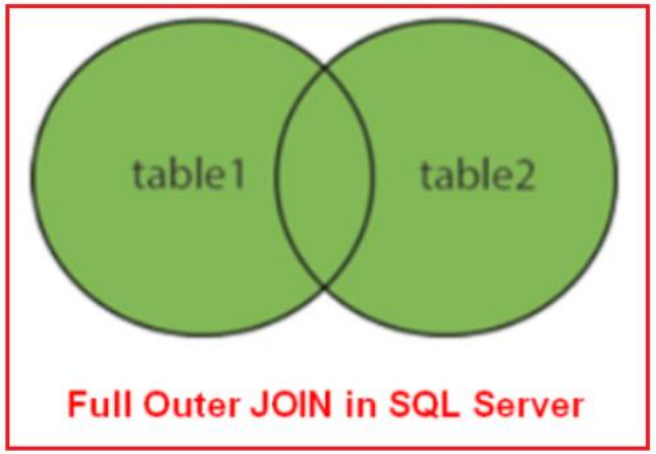


Full Outer Join in SQL Server

The Full Outer Join in SQL Server is used to retrieve all the matching records as well as all the non-matching records from both the tables involved in the JOIN. The Un-matching data in such cases will take the NULL values. The following diagram shows the pictorial representation of Full Outer Join in SQL Server.

```
Syntax : -
Select * from  table1 A full outer join table2 B
A.ColumnName=B.ColumnName
```

```
Example - SELECT Id as EmployeeID, Name, Department, City, Title as Project, ClientId FROM Employee Full
Outer JOIN Projects
ON Employee.Id = Projects.EmployeeId;
```



Cross Join in SQL Server

The CROSS JOIN is created by using the CROSS JOIN keyword. The CROSS JOIN does not contain an ON clause. In Cross Join, each record of a table is joined with each record of the other table. In SQL Server, the Cross Join should not have either an ON or where clause.

```
SELECT Employee.Id as EmployeeId, Name, Department, City, Title as Project
FROM Employee
CROSS JOIN Projects;
```

What is a Stored Procedure in SQL Server?

A SQL Server Stored Procedure is a database object which contains pre-compiled queries (a group of T-SQL Statements). In other words, we can say that the Stored Procedures are a block of code designed to perform a task whenever we called.

In SQL Server, you can create a stored procedure by using the **CREATE PROCEDURE** or **CREATE PROC** statement. Again, you can create a procedure with or without parameters. Please have a look at the below image for the **Syntax of Stored Procedure**.

```
-- Syntax for creating a procedure without parameter
CREATE/ALTER PROCEDURE ProcedureName
AS
BEGIN
        Procedure Body
END

-- Syntax for creating a procedure with parameters
CREATE/ALTER PROCEDURE ProcedureName
(
        @Param1 DataType,
        @Param2 DataType,
        @Paramn DataType,
)
AS
BEGIN
        Procedure Body
END
```

How to call a Stored Procedure in SQL Server?

```
EXECUTE ProcedureName VALUES
OR
EXEC ProcedureName VALUES
OR
ProcedureName VALUES
```

Stored Procedure in SQL Server Without Parameter

```
CREATE PROCEDURE spDisplayWelcome
AS
BEGIN
PRINT 'WELCOME TO PROCEDURE in SQL Server'
END
```

Create a stored procedure to get the names, gender, and the dob of all employees from the table Employee table.

```
CREATE PROCEDURE spGetEmployee
AS
BEGIN
Select Name, Gender, DOB from Employee
END
-- To Execute the Procedure
EXEC spGetEmployee
```

How to change the name and body of a stored procedure in SQL Server?

```
CREATE PROCEDURE spGetEmployee
As
BEGIN
SELECT Name,Gender, DOB FROM Employee
END
-- How to change the body of a stored procedure
-- User Alter procedure to change the body
ALTER PROCEDURE spGetEmployee
AS
BEGIN
SELECT Name, Gender, DOB
FROM Employee
ORDER BY Name
END
-- To change the procedure name from spGetEmployee to spGetEmployee1
-- Use sp_rename system defined stored procedure
EXEC sp_rename 'spGetEmployee', 'spGetEmployee1'
```

How to Drop a Stored Procedure?

In order to drop a stored procedure, all you need to use the following syntax

DROP PROCEDURE ProcedureName

Example: **Drop proc spGetEmployee1** or **Drop Procedure spGetEmployee1**

Different Types of Parameters in SQL Server Stored Procedure.

```
-- Create a Procedure
CREATE PROCEDURE spUpdateEmployeeByID
(
@ID INT,
@Name VARCHAR(50),
@Gender VARCHAR(50), @DOB DATETIME,
@DeptID INT
)
AS
BEGIN
UPDATE Employee SET
Name = @Name,
Gender = @Gender,
DOB = @DOB,
DeptID = @DeptID
WHERE ID = @ID
END
GO
```

1.Input parameters
2.Output parameters

```
CREATE PROCEDURE spGetEmployeesByGenderAndDepartment
@Gender VARCHAR(20),
@DeptID INT
AS
BEGIN
SELECT Name, Gender, DOB, DeptID
FROM Employee
WHERE Gender = @Gender AND DeptID = @DeptID
END
GO
```

```
-- Executing the Procedure
-- If you are not specifying the Parameter Names then the order is important
EXECUTE spUpdateEmployeeByID 3, 'Palak', 'Female', '1994-06-17 10:53:27.060', 3
-- If you are specifying the Parameter Names then order is not mandatory
EXECUTE spUpdateEmployeeByID @ID =3, @Gender = 'Female', @DOB = '1994-06-17 10:53:27.060', @DeptID = 3,
@Name = 'Palak'
```

Stored Procedure with Output Parameter

```
CREATE PROCEDURE spGetEmployeeCountByGender
@Gender VARCHAR(30),
@EmployeeCount INT OUTPUT
AS
BEGIN
SELECT @EmployeeCount = COUNT(ID)
FROM Employee
WHER Gender = @Gender
END
```

Execution

```
DECLARE @EmployeeTotal INT
EXECUTE spGetEmployeeCountByGender 'Male', @EmployeeTotal OUTPUT
PRINT @EmployeeTotal
```

What is a function in SQL Server?

A function in SQL Server is a subprogram that is used to perform an action such as complex calculation and returns the result of the action as a value. There are two types of functions in SQL Server, such as

1.System Defined Function
2.User-Defined Function

```
Select LTRIM('   Hello')
Select RTRIM('Hello   ')
Select LTRIM(RTRIM('   Hello   '))
Select LOWER('CONVERT This String Into Lower Case')

Select UPPER('CONVERT This String Into upperCase')

Select REVERSE('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
Select LEN(' Functions   ')
Select LEFT('ABCDE', 3)
Select RIGHT('ABCDE', 3)
Select CHARINDEX('@','hina@aaa.com',1)
Select SUBSTRING('info@dotnettutorials.net',6, 19)

SELECT Name, Department, Salary,
ROW_NUMBER() OVER (ORDER BY Department) AS RowNumber
FROM Employees
```

ROW_NUMBER function in SQL Server

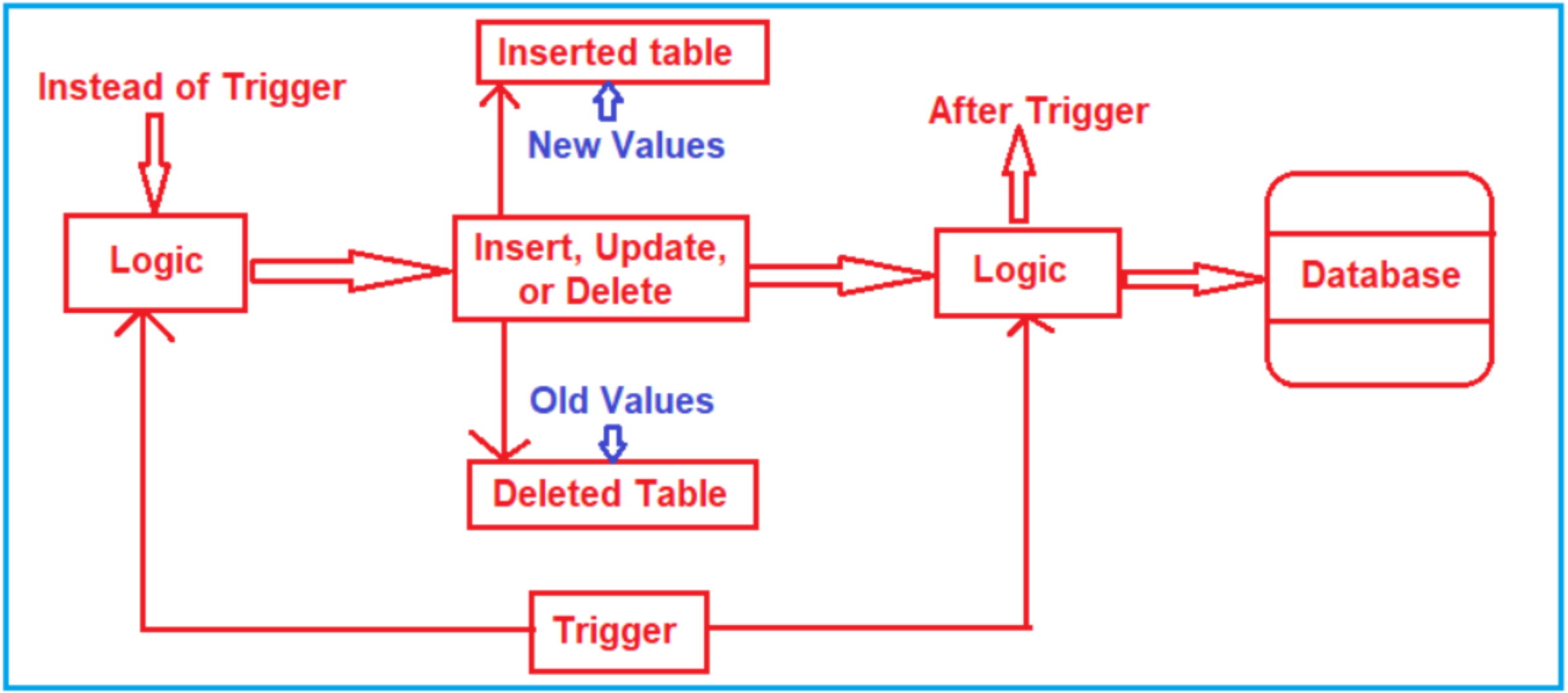
```
SELECT Name, Department, Salary,
ROW_NUMBER() OVER
(
PARTITION BY Department
ORDER BY Name
) AS RowNumber
FROM Employees
```

Delete the duplicate records from the Employees table.

```
WITH DeleteDuplicateCTE AS
(
     SELECT *, ROW_NUMBER() OVER(PARTITION BY ID ORDER BY ID) AS RowNumber
     FROM Employees
)
DELETE FROM DeleteDuplicateCTE WHERE RowNumber > 1
```

What are Triggers in SQL Server?

Triggers are nothing but they are logic's like stored procedures that can be executed automatically before the Insert, Update or Delete happens in a table or after the Insert, Update, or Delete happens in a table. In simple words, we can say that, if you want to execute some pre-processing or post-processing logic before or after the Insert, Update, or Delete in a table then you need to use Triggers in SQL Server.



Types of User-Defined Function:

1. Scalar Valued Functions
2. Inline Table-Valued Functions
3. Multi-Statement Table-Valued Functions

SQL Server Scalar Valued Functions

The user-defined function which returns only a single value (scalar value) is known as the Scalar Valued Function. Scalar Value Functions in SQL Server may or may not have parameters that are optional but always return a single (scalar) value which is mandatory. The returned value which is return by the SQL Server Scalar Function can be of any data type, except text, ntext, image, cursor, and timestamp. Following is the syntax to create a User-Defined Scalar Value Function in SQL Server.

What are Inline Table-Valued functions in SQL Server?

In the case of an Inline Table-Valued Function, the body of the function will have only a **Single Select Statement** prepared with the "**RETURN**" statement. And here, we need to specify the Return Type as TABLE by using the **RETURNS TABLE** statement. The following image shows the syntax of the Inline Table-Valued Function and how to call an Inline Table-Valued Function in SQL Server.

Multi-Statement Table Valued Function in SQL Server

In this article, I am going to discuss the **Multi-Statement Table Valued Function in SQL Server** with Examples. Please read our previous article where we discussed **Inline Table-Valued Function in SQL Server**. There are two types of Table-Valued Functions in SQL Server i.e. Inline Table-Valued and Multi-Statement Table-Valued Function. In our previous article, we already discussed the Inline Table-Valued Function and in this article, we are going to discuss Multi-Statement Table-Valued Function in SQL Server.

```
CREATE FUNCTION CalculateAge
(
@DOB DATE
)
RETURNS INT
AS
BEGIN
DECLARE @AGE INT
SET @AGE = DATEDIFF(YEAR, @DOB, GETDATE())-
CASE
WHEN (MONTH(@DOB) > MONTH(GETDATE())) OR
(MONTH(@DOB) = MONTH(GETDATE()) AND
DAY(@DOB) > DAY(GETDATE()))
THEN 1
ELSE 0
END
RETURN @AGE
END
```

```
SELECT dbo.CalculateAge ('02/29/1988') AS AGE
```

```
CREATE FUNCTION FN_GetStudentDetailsByID
(
@ID INT
)
RETURNS TABLE
AS
RETURN (SELECT * FROM Student WHERE ID = @ID)

SELECT * FROM FN_GetStudentDetailsByID(2)

CREATE FUNCTION FN_GetStudentDetailsByBranch
(
@Branch VARCHAR(50)
)
RETURNS TABLE
AS
RETURN (SELECT * FROM Student WHERE Branch = @Branch)
```

```
-- Multi-statement Table Valued function:
CREATE FUNCTION MSTVF_GetEmployees()
RETURNS @Table Table (ID int, Name nvarchar(20), DOB Date)
AS
BEGIN
INSERT INTO @Table
SELECT ID, Name, Cast(DOB AS Date)
FROM Employee
Return
End

SELECT * FROM MSTVF_GetEmployees()
```

What are the differences between Inline and Multi-Statement Table-Valued Functions in SQL Server?

1.In an **Inline Table-Valued Function**, the returns clause cannot define the structure of the table that the function is going to return whereas in the **Multi-Statement Table-Valued Function** the returns clause defines the structure of the table that the function is going to return.

2.The **Inline Table-Valued Function** cannot have **BEGIN and END** blocks whereas the **Multi-Statement Table-Valued Function** has the **Begin and End** blocks.

3.It is possible to update the underlying database table using the inline table-valued function but it is not possible to update the underlying database table using the multi-statement table-valued function.

4.Inline Table-Valued functions are better for performance than the Multi-Statement Table-Valued function. So, if the given task can be achieved using an Inline Table-Valued Function, then it is always preferred to use Inline Table-valued Function over the Multi-Statement Table-Valued function.

Reason For Better Performance: Internally SQL Server treats an Inline Table-Valued function much like a view and treats a Multi-Statement Table-Valued function as a stored procedure.

1.Instead of Triggers: The Instead Of triggers are going to be executed instead of the corresponding DML operations. That means instead of the DML operations such as Insert, Update, and Delete, the Instead Of triggers are going to be executed.

2.After Triggers: The After Triggers fires in SQL Server execute after the triggering action. That means once the DML statement (such as Insert, Update, and Delete) completes its execution, this trigger is going to be fired.

1.DML Triggers – Data Manipulation Language Triggers.

2.DDL Triggers – Data Definition Language Triggers

3.CLR triggers – Common Language Runtime Triggers

4.Logon triggers

What are DML Triggers in SQL Server?

As we know DML Stands for Data Manipulation Language and it provides Insert, Update and Delete statements to perform the respective operation on the database tables or view which will modify the data. The triggers which are executed automatically in response to DML events (such as Insert, Update, and Delete statements) are called DML Triggers.

```
CREATE TRIGGER trAllDMLOperationsOnEmployee
ON Employee
FOR INSERT, UPDATE, DELETE
AS
BEGIN
PRINT 'YOU CANNOT PERFORM DML OPERATION'
ROLLBACK TRANSACTION
END
```