T.C. İSTANBUL KÜLTÜR ÜNİVERSİTESİ

COM5041 - DATABASE DESIGN AND DEVELOPMENT

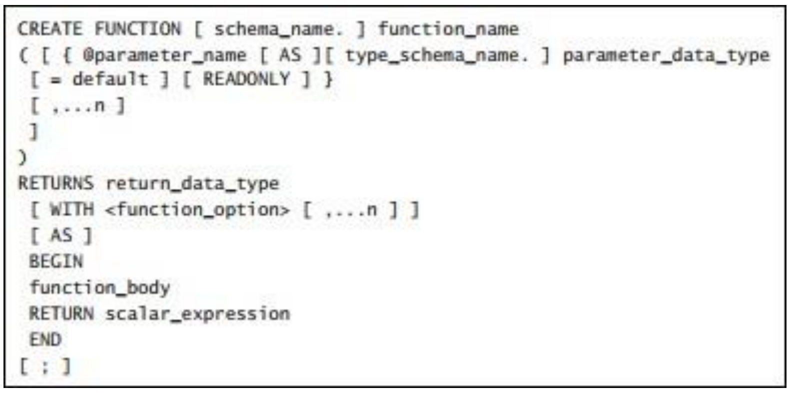
LAB 10 – Working with user-defined functions and XML

After completing this Lab, you will be able to

- Understand user-defined functions.
- Create, alter, and delete functions.
- Understand the difference between scalar and table-valued functions.
- Work with XML in SQL Server.
- Convert tables in SQL into XML.
- Load XML documents into SQL Server
- Create SQL tables from XML documents.

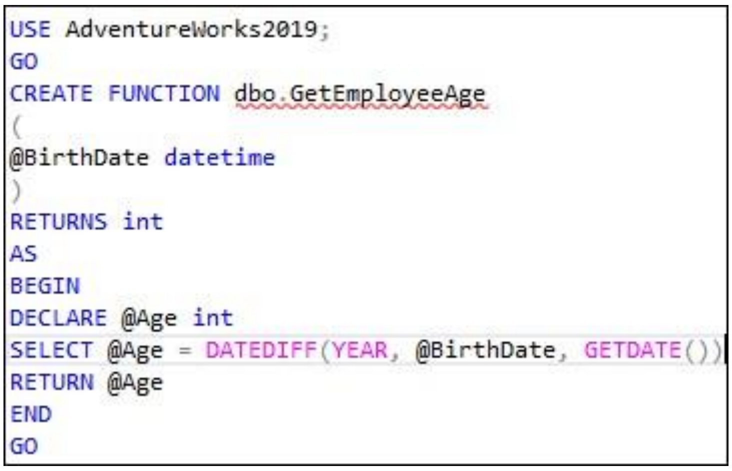
PROCEDURE 1 – How to code user-defined functions

A user-defined scalar function is a routine that returns a single value. These functions are often used to centralize the logic of a complex calculation that may be used by several other database or application resources. The syntax is as follows:



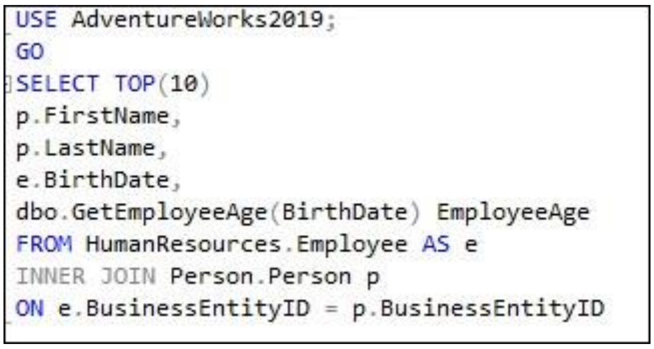
```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
 [ = default ] [ READONLY ] }
 [ ,...n ]
 ]
)
RETURNS return_data_type
 [ WITH <function_option> [ ,...n ] ]
 [ AS ]
 BEGIN
 function_body
 RETURN scalar_expression
 END
[ ; ]
```

Step 1 - Let's define a function that returns the age of employees.



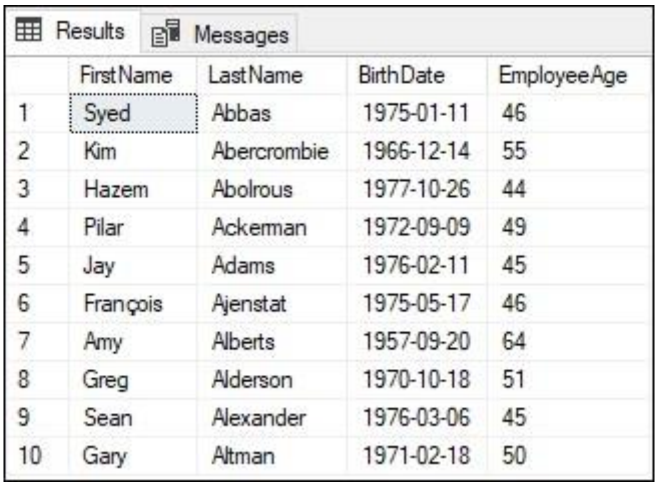
```
USE AdventureWorks2019;
GO
CREATE FUNCTION dbo.GetEmployeeAge
(
@BirthDate datetime
)
RETURNS int
AS
BEGIN
DECLARE @Age int
SELECT @Age = DATEDIFF(YEAR, @BirthDate, GETDATE())
RETURN @Age
END
GO
```

Step 2 – We create a query that returns the FirstName, LastName, BirthDate and Age information of the first 10 employees.



```
USE AdventureWorks2019;
GO
SELECT TOP(10)
p.FirstName,
p.LastName,
e.BirthDate,
dbo.GetEmployeeAge(BirthDate) EmployeeAge
FROM HumanResources.Employee AS e
INNER JOIN Person.Person p
ON e.BusinessEntityID = p.BusinessEntityID
```

** Execute the query and view the result set.

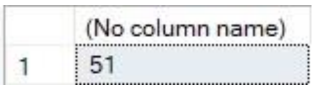


	FirstName	LastName	BirthDate	EmployeeAge
1	Syed	Abbas	1975-01-11	46
2	Kim	Abercrombie	1966-12-14	55
3	Hazem	Abolrous	1977-10-26	44
4	Pilar	Ackerman	1972-09-09	49
5	Jay	Adams	1976-02-11	45
6	François	Ajenstat	1975-05-17	46
7	Amy	Alberts	1957-09-20	64
8	Greg	Alderson	1970-10-18	51
9	Sean	Alexander	1976-03-06	45
10	Gary	Altman	1971-02-18	50

Step 3 - A scalar function can be included in a SELECT statement. The parameters can be a column, constant, or expression.

```
SELECT dbo.GetEmployeeAge ('5/26/1972')
```

** Execute the query and view the result set.



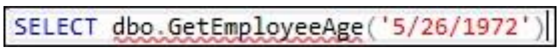
	(No column name)
1	51

Step 4- A parameter, in the scope of T-SQL function programming, is an input value that can be passed from the calling function into the code. A parameter can be set to a constant, a column from a table, an expression, and other values. Functions can contain two types of parameters: INPUT and DEFAULT. If you have a single input parameter and any other combination of other types of parameters, you must ensure that the order in which they are passed corresponds to the order in which they are specified in the function. In the following function, there are two parameters:

```
USE AdventureWorks2019;
GO
IF(OBJECT_ID('dbo.GetEmployeeAge')) IS NOT NULL
DROP FUNCTION dbo.GetEmployeeAge
GO
CREATE FUNCTION dbo.GetEmployeeAge
(
@BirthDate datetime = '5/26/1972',
@Temp datetime = NULL
)
RETURNS int
AS
BEGIN
DECLARE @Age int
SELECT @Age = DATEDIFF(Year, @BirthDate, GETDATE())
RETURN @Age
END
GO
```

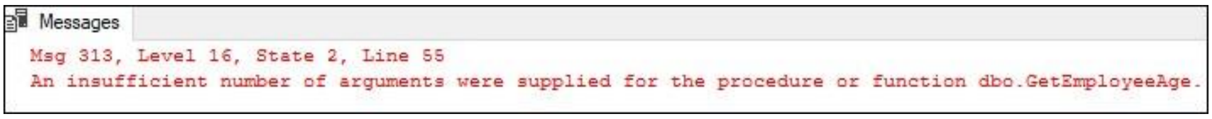
Note that; The Object_Id function is a function used when we want to check whether a table exists in SQL Server.

- In the following code, the first SELECT statement calls the function with a single parameter.



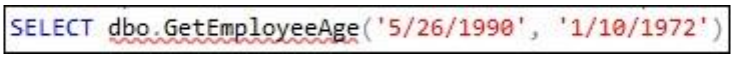
```
SELECT dbo.GetEmployeeAge('5/26/1972')
```

** Execute the query and view the result set.



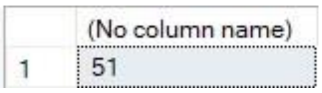
```
Messages
Msg 313, Level 16, State 2, Line 55
An insufficient number of arguments were supplied for the procedure or function dbo.GetEmployeeAge.
```

- The SELECT statement has two parameters. Since both values are provided, it succeeds.



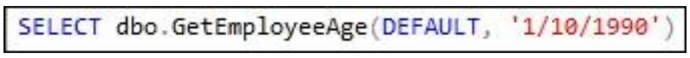
```
SELECT dbo.GetEmployeeAge('5/26/1990', '1/10/1972')
```

** Execute the query and view the result set.



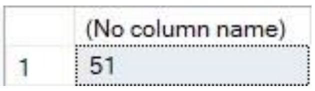
	(No column name)
1	51

- In the following code, the first parameter is a default parameter and the second is an input Because DEFAULT is specified as the first value, SQL Server will use the value assigned to that parameter, and the date value will be assigned to the input parameter.
-



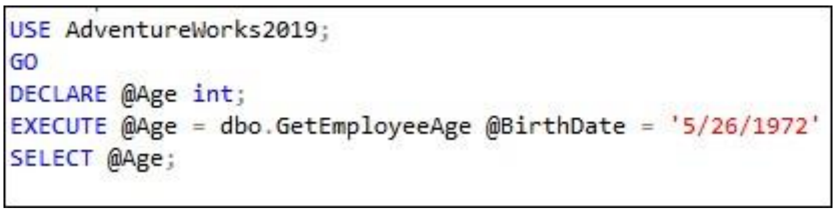
```
SELECT dbo.GetEmployeeAge(DEFAULT, '1/10/1990')
```

** Execute the query and view the result set.



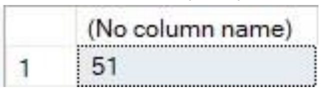
	(No column name)
1	51

Step 5 - A scalar function can also be called using the EXECUTE keyword. To obtain the output of a scalar function using the EXECUTE keyword, you must declare a variable that will hold the output:



```
USE AdventureWorks2019;
GO
DECLARE @Age int;
EXECUTE @Age = dbo.GetEmployeeAge @BirthDate = '5/26/1972'
SELECT @Age;
```

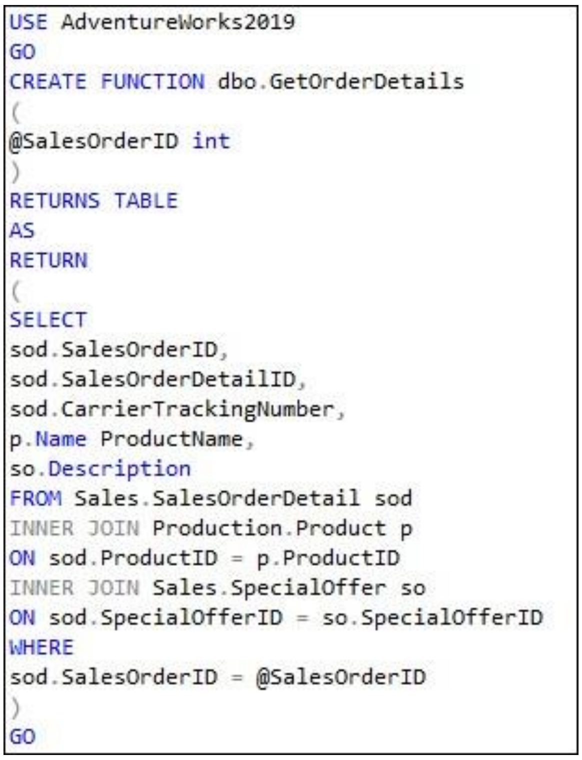
** Execute the query and view the result set.



	(No column name)
1	51

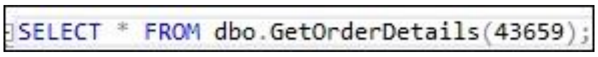
Step 6 – Table-valued functions come in two types: INLINE and MULTISTATEMENT. The inline function only returns the result of a SELECT as its set, while the multi-statement function uses a table variable that can be defined. The rows of data are added to the table as per the code and can be manipulated before the data is returned. This is unlike the inline table valued function, where any data manipulations or filters must be done in the actual query.

- In the following code, it is created the inline table-valued function. The inline table-valued function accepts one parameter, SalesOrderID. This parameter is used in the function to limit the result set to only rows that are associated with that value Inside the function is a single TSQL statement that is limited by the parameter.

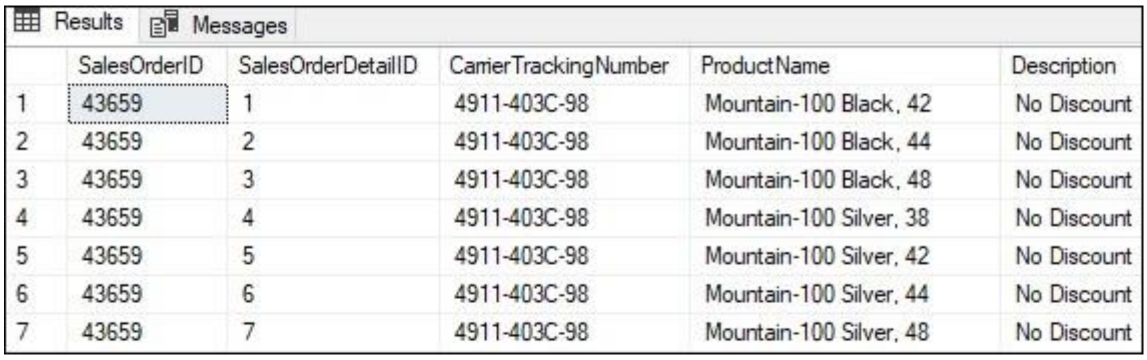


```
USE AdventureWorks2019
GO
CREATE FUNCTION dbo.GetOrderDetails
(
@SalesOrderID int
)
RETURNS TABLE
AS
RETURN
(
SELECT
sod.SalesOrderID,
sod.SalesOrderDetailID,
sod.CarrierTrackingNumber,
p.Name ProductName,
so.Description
FROM Sales.SalesOrderDetail sod
INNER JOIN Production.Product p
ON sod.ProductID = p.ProductID
INNER JOIN Sales.SpecialOffer so
ON sod.SpecialOfferID = so.SpecialOfferID
WHERE
sod.SalesOrderID = @SalesOrderID
)
GO
```

** Execute the query and view the result set.



```
SELECT * FROM dbo.GetOrderDetails(43659);
```



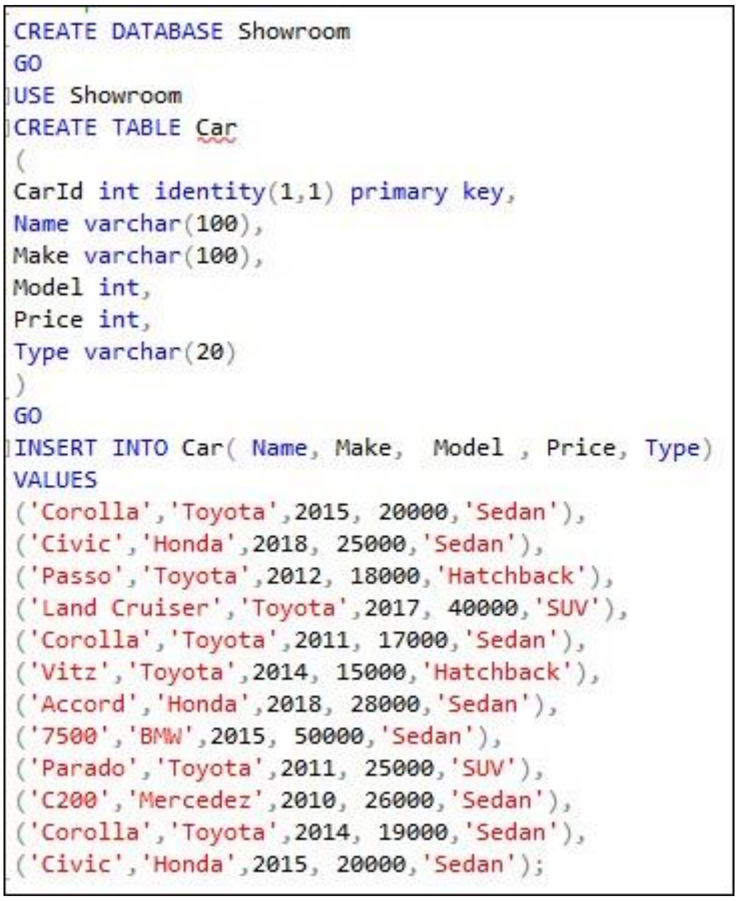
Results Messages

	SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	ProductName	Description
1	43659	1	4911-403C-98	Mountain-100 Black, 42	No Discount
2	43659	2	4911-403C-98	Mountain-100 Black, 44	No Discount
3	43659	3	4911-403C-98	Mountain-100 Black, 48	No Discount
4	43659	4	4911-403C-98	Mountain-100 Silver, 38	No Discount
5	43659	5	4911-403C-98	Mountain-100 Silver, 42	No Discount
6	43659	6	4911-403C-98	Mountain-100 Silver, 44	No Discount
7	43659	7	4911-403C-98	Mountain-100 Silver, 48	No Discount

PROCEDURE 2 – How to work with XML

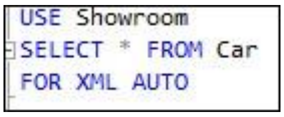
XML (eXtensible Markup Language) is one of the most common formats used to share information between different platforms.

Step 1 - In the following script, we will create a Showroom database with one table Car. The Car table has five attributes: CarId, Name, Make, Model, Price, and Type. Next, we added 12 records in the Car table.



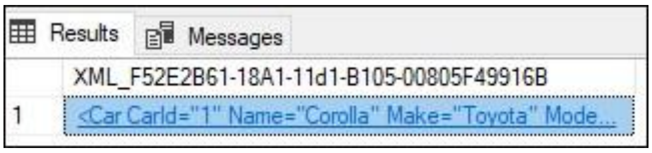
```
CREATE DATABASE Showroom
GO
USE Showroom
CREATE TABLE Car
(
CarId int identity(1,1) primary key,
Name varchar(100),
Make varchar(100),
Model int,
Price int,
Type varchar(20)
)
GO
INSERT INTO Car( Name, Make,  Model , Price, Type)
VALUES
('Corolla','Toyota',2015, 20000,'Sedan'),
('Civic','Honda',2018, 25000,'Sedan'),
('Passo','Toyota',2012, 18000,'Hatchback'),
('Land Cruiser','Toyota',2017, 40000,'SUV'),
('Corolla','Toyota',2011, 17000,'Sedan'),
('Vitz','Toyota',2014, 15000,'Hatchback'),
('Accord','Honda',2018, 28000,'Sedan'),
('7500','BMW',2015, 50000,'Sedan'),
('Parado','Toyota',2011, 25000,'SUV'),
('C200','Mercedez',2010, 26000,'Sedan'),
('Corolla','Toyota',2014, 19000,'Sedan'),
('Civic','Honda',2015, 20000,'Sedan');
```

Step 2 - The FOR XML AUTO clause converts each column in the SQL table into an attribute in the corresponding XML document.



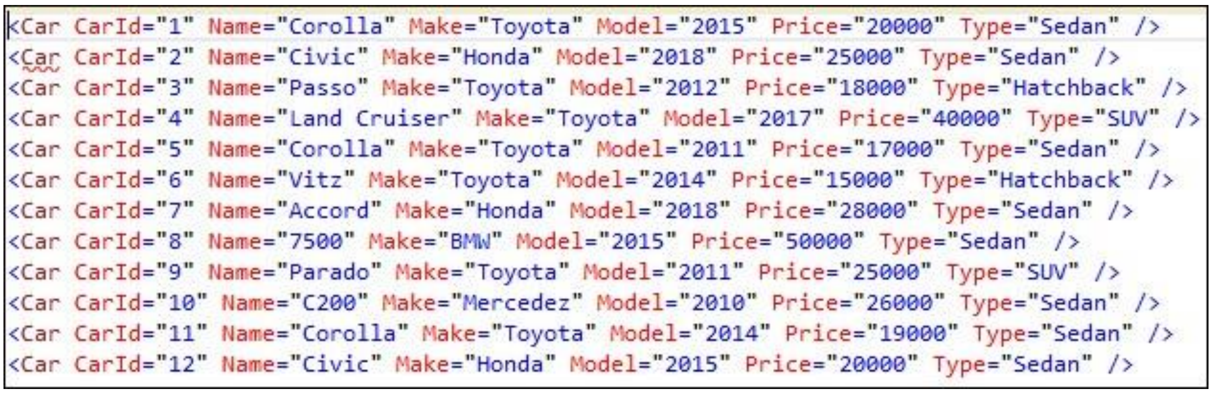
```
USE Showroom
SELECT * FROM Car
FOR XML AUTO
```

** Execute the query and view the result set.



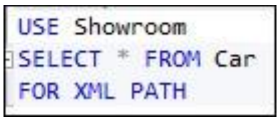
```
   XML_F52E2B61-18A1-11d1-B105-00805F49916B
1  <Car CarId="1" Name="Corolla" Make="Toyota" Mode...
```

** Click on the link and you will see the following document in a new query window of SQL Server management studio. Then, you can see that for each record an element Car has been created in the XML document, and for each column, an attribute with the same name has been added to each element in the XML document.



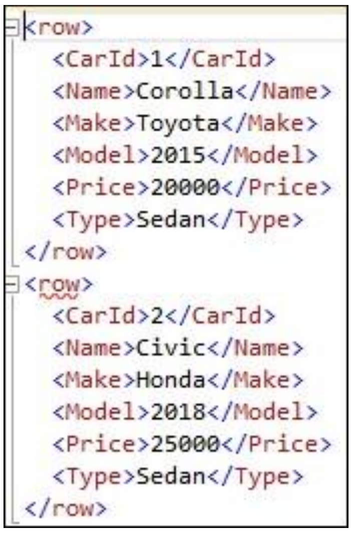
```
<Car CarId="1" Name="Corolla" Make="Toyota" Model="2015" Price="20000" Type="Sedan" />
<Car CarId="2" Name="Civic" Make="Honda" Model="2018" Price="25000" Type="Sedan" />
<Car CarId="3" Name="Passo" Make="Toyota" Model="2012" Price="18000" Type="Hatchback" />
<Car CarId="4" Name="Land Cruiser" Make="Toyota" Model="2017" Price="40000" Type="SUV" />
<Car CarId="5" Name="Corolla" Make="Toyota" Model="2011" Price="17000" Type="Sedan" />
<Car CarId="6" Name="Vitz" Make="Toyota" Model="2014" Price="15000" Type="Hatchback" />
<Car CarId="7" Name="Accord" Make="Honda" Model="2018" Price="28000" Type="Sedan" />
<Car CarId="8" Name="7500" Make="BMW" Model="2015" Price="50000" Type="Sedan" />
<Car CarId="9" Name="Parado" Make="Toyota" Model="2011" Price="25000" Type="SUV" />
<Car CarId="10" Name="C200" Make="Mercedez" Model="2010" Price="26000" Type="Sedan" />
<Car CarId="11" Name="Corolla" Make="Toyota" Model="2014" Price="19000" Type="Sedan" />
<Car CarId="12" Name="Civic" Make="Honda" Model="2015" Price="20000" Type="Sedan" />
```

Step 3 – The FOR XML PATH will create an XML document where each record is an element and each column is a nested element for a particular record. Execute the query and view the result set.



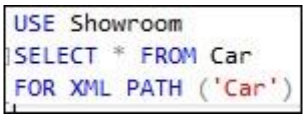
```
USE Showroom
SELECT * FROM Car
FOR XML PATH
```

** Click on the link and you will see the following document in a new query window of SQL Server management studio.



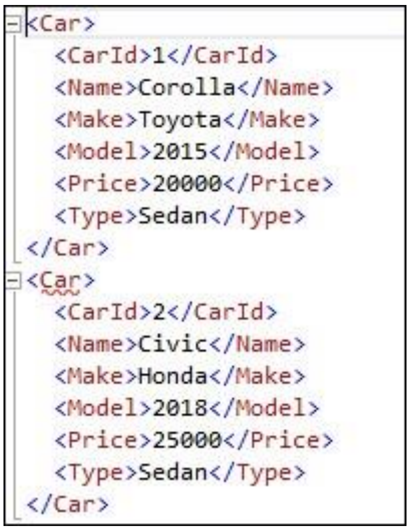
```
<row>
  <CarId>1</CarId>
  <Name>Corolla</Name>
  <Make>Toyota</Make>
  <Model>2015</Model>
  <Price>20000</Price>
  <Type>Sedan</Type>
</row>
<row>
  <CarId>2</CarId>
  <Name>Civic</Name>
  <Make>Honda</Make>
  <Model>2018</Model>
  <Price>25000</Price>
  <Type>Sedan</Type>
</row>
```

Step 4 - In the output of script above, you will see a total of 12 elements (the screenshot shows only the first 2). You can see that each column name has been converted to an element. However, there is one problem; by default, the parent element name is "row". We can change that using the following query:



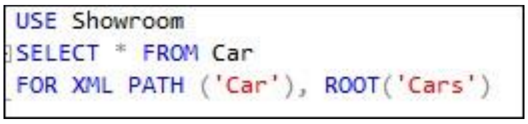
```
USE Showroom
SELECT * FROM Car
FOR XML PATH ('Car')
```

** Click on the link and you will see the following document in a new query window of SQL Server management studio.



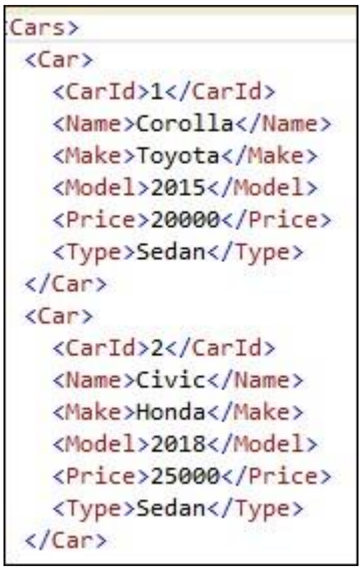
```
<Car>
  <CarId>1</CarId>
  <Name>Corolla</Name>
  <Make>Toyota</Make>
  <Model>2015</Model>
  <Price>20000</Price>
  <Type>Sedan</Type>
</Car>
<Car>
  <CarId>2</CarId>
  <Name>Civic</Name>
  <Make>Honda</Make>
  <Model>2018</Model>
  <Price>25000</Price>
  <Type>Sedan</Type>
</Car>
```

Step 5 - In the output of script above, you can see Car as the parent element for each sub-element. However, the document is not well-formed as there is no root element in the document. To add a root element, we need to execute the following script:



```
USE Showroom
SELECT * FROM Car
FOR XML PATH ('Car'), ROOT('Cars')
```

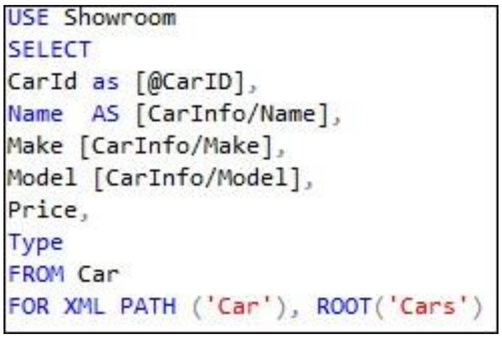
** Click on the link and you will see the following document in a new query window of SQL Server management studio.



```
Cars>
 <Car>
   <CarId>1</CarId>
   <Name>Corolla</Name>
   <Make>Toyota</Make>
   <Model>2015</Model>
   <Price>20000</Price>
   <Type>Sedan</Type>
 </Car>
 <Car>
   <CarId>2</CarId>
   <Name>Civic</Name>
   <Make>Honda</Make>
   <Model>2018</Model>
   <Price>25000</Price>
   <Type>Sedan</Type>
 </Car>
```

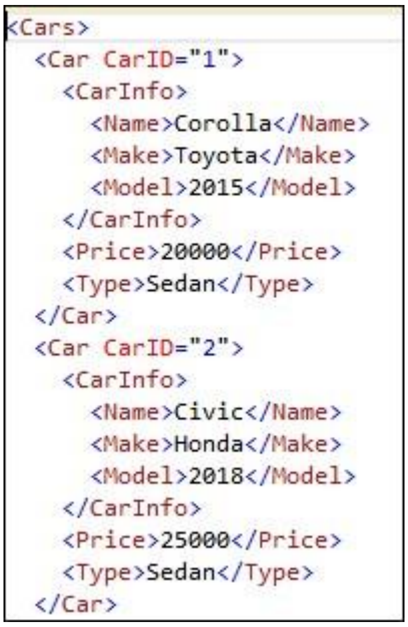
** In the output, you should see "Cars" as the root element.

Step 6 - Now suppose you want that CarId should be the attribute of the Car element rather than an element. Then, we can add further nesting levels to an XML document. For instance, if we want Name, Make and Model elements to be nested inside another element CarInfo we can do so with the following script:



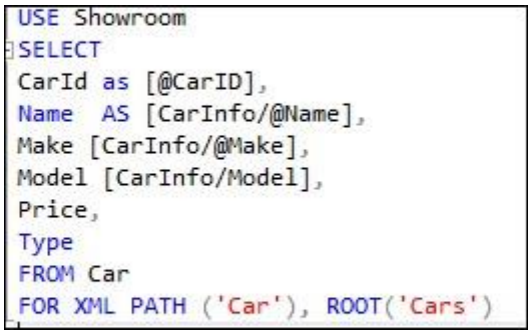
```
USE Showroom
SELECT
CarId as [@CarID],
Name  AS [CarInfo/Name],
Make [CarInfo/Make],
Model [CarInfo/Model],
Price,
Type
FROM Car
FOR XML PATH ('Car'), ROOT('Cars')
```

** Click on the link and you will see the following document in a new query window of SQL Server management studio. In the output, you will see a new element CarInfo that encloses the Name, Make and Model elements.



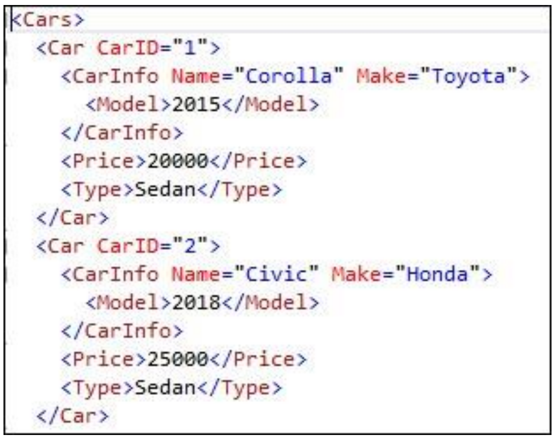
```
<Cars>
  <Car CarID="1">
    <CarInfo>
      <Name>Corolla</Name>
      <Make>Toyota</Make>
      <Model>2015</Model>
    </CarInfo>
    <Price>20000</Price>
    <Type>Sedan</Type>
  </Car>
  <Car CarID="2">
    <CarInfo>
      <Name>Civic</Name>
      <Make>Honda</Make>
      <Model>2018</Model>
    </CarInfo>
    <Price>25000</Price>
    <Type>Sedan</Type>
  </Car>
```

Step 7 - If you want to convert the elements Name and Make into an attribute of element CarInfo, you can do so with the following script:



```
USE Showroom
SELECT
CarId as [@CarID],
Name  AS [CarInfo/@Name],
Make [CarInfo/@Make],
Model [CarInfo/Model],
Price,
Type
FROM Car
FOR XML PATH ('Car'), ROOT('Cars')
```

** Click on the link and you will see the following document in a new query window of SQL Server management studio.



```
<Cars>
  <Car CarID="1">
    <CarInfo Name="Corolla" Make="Toyota">
      <Model>2015</Model>
    </CarInfo>
    <Price>20000</Price>
    <Type>Sedan</Type>
  </Car>
  <Car CarID="2">
    <CarInfo Name="Civic" Make="Honda">
      <Model>2018</Model>
    </CarInfo>
    <Price>25000</Price>
    <Type>Sedan</Type>
  </Car>
```

SUMMARY

In this LAB, you've learned the two types of user-defined functions available in Microsoft SQL Server 2019: scalar and table-valued. Scalar functions return a single value, while table-valued functions return a complete set. You've learned to how to create a document using XML from a SQL table. You also saw how to import into a table in SQL from an XML document.