User Defined Functions

In addition to SQL Server function that we have been using so far, you can also create your own functions. To do that, you use code that is similar to the code you use to create a stored procedure, however there are some distinct differences.

There are three types of user defined functions:

- **Scalar-valued functions**: Returns a single value
- **Table-valued functions**: Returns an entire table
  - **Simple table-valued function**: Based on a single SELECT statement
  - **Multi-statement table-valued function**: Based on multiple SQL statements (we won't be covering this since you will be able to achieve what you want using the simple table-valued functions)

Like a stored procedure, a function can accept one or more input parameters. However a function can't be defined with output parameters. Instead, the RETURN statement must be used to pass a value back to the calling program. The value that is returned must be compatible with the data type that's specified in the RETURNS clause.

How to Create Scalar-Valued Functions

```
CREATE FUNCTION function_name
[(pameter declaration]
RETURNS data_type
WITH ENCRYPTION
[AS]
BEGIN
      [SQL_Statement]
      RETURN scalar_expression
END
```

The following example creates a scalar function that returns the total invoice amount due. This function doesn't accept any input parameters and returns a value with the money data type. In this case the code for the function consists of a single SELECT statement coded within the RETURN statement.

```
CREATE FUNCTION fnBalanceDue()
      RETURNS money
BEGIN
      RETURN (SELECT SUM(InvoiceTotal-PaymentTotal-CreditTotal)
                  FROM Invoices
                  WHERE InvoiceTotal-PaymentTotal-CreditTotal>0);
END
```

If you find yourself repeatedly coding the same expression, you may want to create a scalar function for the expression. Most SQL programmers create a set of useful functions each time they work on a new database.

- To invoke a function, list the parameters within parentheses after the name of the function. To use the default value of a parameter, code the DEFAULT keyword in place of the parameter value.
- The ENCRYPTION option prevents users from viewing the code in the function.

To invoke the above function:

```
PRINT dbo.fnBalanceDue()
```

The following function returns the users age given a DOB.

```
CREATE FUNCTION GetAge (@DOB datetime, @TODAY datetime)
      RETURNS int
BEGIN
RETURN DateDiff (year,@DOB, @TODAY)- (CASE WHEN Month(@DOB) >
Month(@TODAY) Or (Month(@DOB) = Month(@TODAY) And Day(@DOB) >
Day(@TODAY)) THEN 1 ELSE 0 END)
END
```

The following shows how to use the function to update the Employees table's Age column with the age of each person.

```
UPDATE Employees SET Age= dbo.GetAge(DOB,GetDate())
```

How to Create a Simple Table-Valued Function

```
CREATE FUNCTION function_name
[(parameter declaration)]
RETURNS TABLE
WITH ENCRYPTION
[AS]
RETURN
      [(SELECT Statement)]
```

The above syntax is used if the result set can be returned from a single SELECT statement. To declare a function as table-valued, code table data type in the RETURNS clause. Then code the SELECT statement that defines the table in parentheses in the RETURN statement. Note that because a table can't have any unnamed columns, you must assign a name to every calculated column in the result set.

The following example shows a function that returns a table that contains the vendor name and total balance due for each vendor with a balance due. The one input parameter, @CutOff, is an optional parameter because it's assigned a default value of 0. This parameter is used in the

HAVING clause to return only those vendors with total invoices that are greater than or equal to the specified amount.

```
CREATE FUNCTION fntopVendorsDue
      (@CutOff money = 0)
      RETURNS TABLE
RETURN
      (SELECT VendorName, SUM(InvoiceTotal) AS TotalDue
      FROM Invoices INNER JOIN Vendors ON
Invoices.VendorID=Vendors.VendorID
      GROUP BY VendorName

      HAVING SUM(InvoiceTotal) >= @CutOff);
```

- To use a simple table-valued function, code the function name in place of a table name. If you want to use a table-valued function in a join operation, give it an alias name.

The shows calling the function:

```
SELECT * FROM dbo.fntopVendorsDue(5000);
```

And the below shows using the function in a JOIN statement.

```
SELECT Vendors.VendorName, VendorCity, TotalDue
FROM Vendors INNER JOIN dbo.fntopVendorsDue(DEFAULT) AS TopVendors ON
Vendors.VendorName = TopVendors.VendorName;
```

- Use DROP FUNCTION to permanently delete a function.
- To modify the function use ALTER FUNCTION statement

SQL Exercise:

1. Create a scalar-valued function named fnUnpaidInvoiceID that returns the InvoiceID of the earliest invoice with an unpaid balance.

2. Test the function from #1 in a SELECT statement that returns the VendorName, InvoiceNumber, InvoiceDueDate and balance of the invoices returns from this function.

3. Create a table-valued function name fnDateRange, similar to the stored procedure of spDateRange (that you created in last lecture). The function requires two parameters of data type smalldatetime. Don't validate the parameters. Return a result set that includes that InvoiceNumber, InvoiceDate, InvoiceTotal and Balance for each invoice for which the InvoiceDate is within the data range.

4. Invoke the function from #3 from within a SELECT statement to return those invoices with InvoiceDate between December 10 and December 20, 2019.