MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 10: Implementing User-defined Functions

Contents

Microsoft

Project Lead: Rich Rose
Instructional Designers: Rich Rose, Cheryl Hoople, Marilyn McGill
Instructional Software Design Engineers: Karl Dehmer, Carl Raebler, Rick Byham
Technical Lead: Karl Dehmer
Subject Matter Experts: Karl Dehmer, Carl Raebler, Rick Byham
Graphic Artist: Kirsten Larson (Independent Contractor)
Editing Manager: Lynette Skinner
Editor: Wendy Cleary
Copy Editor: Edward McKillop (S&T Consulting)
Production Manager: Miracle Davis
Production Coordinator: Jenny Boe
Production Support: Lori Walker (S&T Consulting)
Test Manager: Sid Benavente
Courseware Testing: TestingTesting123
Classroom Automation: Lorrin Smith-Bates
Creative Director, Media/Sim Services: David Mahlmann
Web Development Lead: Lisa Pease
CD Build Specialist: Julie Challenger
Online Support: David Myka (S&T Consulting)
Localization Manager: Rick Terek
Operations Coordinator: John Williams
Manufacturing Support: Laura King; Kathy Hershey
Lead Product Manager, Release Management: Bo Galford
Lead Product Manager, Data Base: Margo Crandall
Group Manager, Courseware Infrastructure: David Bramble
Group Product Manager, Content Development: Dean Murray
General Manager: Robert Stewart

Instructor Notes

Presentation:
30 Minutes

Lab:
30 Minutes

This module provides students with a discussion of the implementation of user-defined functions. It explains the three types of user-defined functions and the general syntax for creating and altering them, and provides an example of each type.

- After completing this module, you will be able to:
- Describe the three types of user-defined functions.
- Create and alter user-defined functions.
- Create each of the three types of user-defined functions.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2073A_10.ppt
- The C:\Moc\2073A\Demo\D10_Ex.sql example file, which contains all of the example scripts from the module, unless otherwise noted in the module.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.

Module Strategy

Use the following strategy to present this module:

- What Is a User-defined Function?

  Knowledge of built-in functions is a prerequisite for this course. Review a simple function such as COUNT or SUM to confirm that students understand this concept. Introduce the three types of user-defined functions, indicating that the SCALAR function is the most similar to built-in functions. Point out that the other two functions return a table.

- Defining User-defined Functions

  Review the general syntax for creating a function, specifically CREATE FUNCTION, the function name, the input parameters, the RETURNS clause, and the content area. Use the SCALAR function example on the slide, but keep the discussion focused on the portions of the syntax that are common to all user-defined functions. Discuss the meaning of the term nondeterministic by pointing out that GETDATE() returns a different value each time that it is used. Point out that schema binding, security, and altering and dropping of user-defined functions work similar to the way that they do with views.

- Examples of User-defined Functions

  Point out that the example provided—when discussing the creation of user-defined functions—was a SCALAR function. Review the second example, emphasizing the data type declaration and the placement of RETURN inside the BEGIN…END block. Show how the function is called, emphasizing the owner name in the two-part naming system.

  Review the bulleted items on the Using a Multi-Statement Table-valued Function slide, and emphasize that the result of table-valued functions is a table. Show and review the syntax for creating the function. Point out how the RETURNS clause defines the table name and structure. Discuss the BEGIN…END block and the location of the RETURN near the end. Demonstrate the use of the function, emphasizing how the function replaces a table or view. Remind students that a stored procedure cannot be used in place of a table or view, and point out that a multi-statement table-valued function is essentially a stored procedure that can be used as a table or view.

  Review the bulleted items on the Using an In-Line Table-valued Function slide. Emphasize the restriction to one SELECT statement. Note that this restriction makes the BEGIN…END block unnecessary. Point out that the restriction to a single SELECT statement makes an in-line table-valued function similar to a view. Advance to the syntax slide, and after reviewing the syntax, point out that an in-line table-valued function differs from a view in its ability to accept a parameter. Demonstrate the example, calling the function. Point out that an indexed view with aggregations can be slow to create, but provides a fast response to queries. An in-line table-valued function directed against the indexed view provides a convenient method for retrieving the desired aggregations.

Customization Information

This section identifies the lab setup requirements for a module and the configuration changes that occur on student computers during the labs. This information is provided to assist you in replicating or customizing Microsoft Official Curriculum (MOC) courseware.

Important The lab in this module is dependent on the classroom configuration that is specified in the Customization Information section at the end of the *Classroom Setup Guide* for course 2073A, *Programming a Microsoft SQL Server 2000 Database*.

Lab Setup

The following section describes the setup requirement for the lab in this module.

Setup Requirement

The lab in this module requires the **ClassNorthwind** database to be in a state required for this lab. To prepare student computers to meet this requirement, perform one of the following actions:

- Complete the prior lab
- Execute the C:\Moc\2073A\Batches\Restore10.cmd batch file.

Warning If this course has been customized, students must execute the C:\Moc\2073A\Batches\Restore10.cmd batch file to ensure that the lab will function properly.

Lab Results

There are no configuration changes on student computers that affect replication or customization.

Overview

Topic Objective
To provide an overview of the module topics and objectives.

Lead-in
In this module, you will learn about creating and using user-defined functions.

- **What Is a User-defined Function?**
- **Defining User-defined Functions**
- **Examples of User-defined Functions**

This module provides an overview of user-defined functions. It explains why and how you use them, as well as the syntax for creating them.

After completing this module, you will be able to:

- Describe the three types of user-defined functions.
- Create and alter user-defined functions.
- Create each of the three types of user-defined functions.

What Is a User-defined Function?

Topic Objective
To introduce the concept of a user-defined function and to state the advantages of using one.

Lead-in
There are three types of user-defined functions.

- **Scalar Functions**
  - Similar to a built-in function
- **Multi-Statement Table-valued Functions**
  - Content like a stored procedure
  - Referenced like a view
- **In-Line Table-valued Functions**
  - Similar to a view with parameters
  - Returns a table as the result of single SELECT statement

With Microsoft® SQL Server™ 2000, you can design your own functions to supplement and extend the system-supplied (built-in) functions.

A user-defined function takes zero, or more, input parameters and returns either a *scalar* value or a table. Input parameters can be any data type except **timestamp**, **cursor**, or **table**. User-defined functions do not support output parameters.

Delivery Tip
This page introduces the three types of user-defined functions. Advise students that subsequent topics will discuss the differences between these types of functions.

SQL Server 2000 supports three types of user-defined functions:

Scalar Functions

A scalar function is similar to a built-in function.

Multi-Statement Table-valued Functions

A multi-statement table-valued function returns a table built by one or more Transact-SQL statements and is similar to a stored procedure. Unlike a stored procedure, a multi-statement table-valued function can be referenced in the FROM clause of a SELECT statement as if it were a view.

In-Line Table-valued Functions

An in-line table-valued function returns a table that is the result of a single SELECT statement. It is similar to a view but offers more flexibility than views in the use of parameters, and extends the features of indexed views.

◆ Defining User-defined Functions

Topic Objective
To introduce the topics in this section.

Lead-in
This section covers creating, altering, and dropping a user-defined function. It also covers permissions.

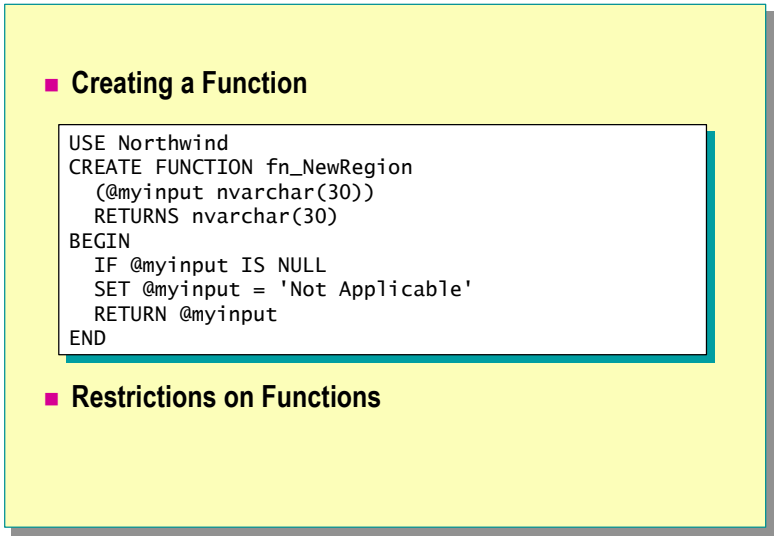
- Creating a User-defined Function
- Creating a Function with Schema Binding
- Setting Permissions for User-defined Functions
- Altering and Dropping User-defined Functions

This section covers creating, altering, and dropping a user-defined function. It also covers permissions.

Creating a User-defined Function

Topic Objective
To describe the general CREATE FUNCTION statement.

Lead-in
You create a user-defined function in nearly the same way that you create a view or stored procedure.



Delivery Tip
Ask students what type of user-defined function is used in the example.

This example is a scalar user-defined function, but keep the discussion generic and applicable to all user-defined functions.

You create a user-defined function in nearly the same way that you create a view or stored procedure.

Creating a Function

You create user-defined functions by using the CREATE FUNCTION statement. Each fully qualified user-defined function name (database_name.owner_name.function_name) must be unique. The statement specifies the input parameters with their data types, the processing instructions, and the value returned with each data type.

Syntax

```
CREATE FUNCTION [ owner_name. ] function_name
  ( [ { @parameter_name scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
  RETURNS scalar_return_data_type
  [ WITH < function_option> [,...n] ]
  [ AS ]
  BEGIN
  function_body
  RETURN scalar_expression
  END
```

Example

This example creates a user-defined function to replace a null value with the words Not Applicable.

```
USE Northwind
GO

CREATE FUNCTION fn_NewRegion
(@myinput nvarchar(30))
RETURNS nvarchar(30)
BEGIN
  IF @myinput IS NULL
  SET @myinput = 'Not Applicable'
  RETURN @myinput
END
```

When referencing a scalar user-defined function, specify the function owner and the function name in two-part syntax.

```
SELECT LastName, City, dbo.fn_NewRegion(Region) AS Region,
   Country
FROM dbo.Employees
```

Result

LastName	City	Region	Country
Davolio	Seattle	WA	USA
Fuller	Tacoma	WA	USA
Leverling	Kirkland	WA	USA
Peacock	Redmond	WA	USA
Buchanan	London	Not Applicable	UK
Suyama	London	Not Applicable	UK
King	London	Not Applicable	UK
Callahan	Seattle	WA	USA
Dodsworth	London	Not Applicable	UK

Restrictions on Functions

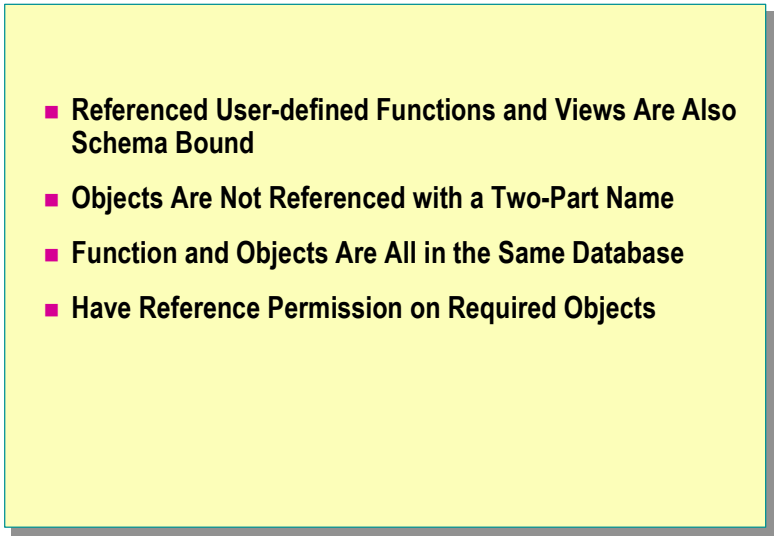
Nondeterministic functions are functions, such as GETDATE(), that could return different result values each time that they are called with the same set of input values. Built-in nondeterministic functions are not allowed in the body of user-defined functions. The following built-in functions are nondeterministic.

@@ERROR	FORMATMESSAGE	IDENTITY	USER_NAME
@@IDENTITY	GETANSINULL	NEWID	@@ERROR
@@ROWCOUNT	GETDATE	PERMISSIONS	@@IDENTITY
@@TRANCOUNT	GetUTCDate	SESSION_USER	@@ROWCOUNT
APP_NAME	HOST_ID	STATS_DATE	@@TRANCOUNT
CURRENT_TIMESTAMP	HOST_NAME	SYSTEM_USER	
CURRENT_USER	IDENT_INCR	TEXTPTR	
DATENAME	IDENT_SEED	TEXTVALID	

Creating a Function with Schema Binding

Topic Objective
To discuss the purpose and restrictions of schema binding.

Lead-in
You can use schema binding to bind the function to the database objects that it references.



You can use schema binding to bind the function to the database objects that it references. If a function is created with the SCHEMABINDING option, then the database objects that the function references cannot be altered (by using the ALTER statement) or dropped (by using a DROP statement).

A function can be schema-bound only if the following conditions are true:

- Any user-defined functions and views referenced by the function are also schema-bound.
- The objects that the function references are not referenced with a two-part name in the **owner.objectname** format.
- The function and the objects that it references belong to the same database.
- The user who executed the CREATE FUNCTION statement has REFERENCE permission on all of the database objects that the function references.

Setting Permissions for User-defined Functions

Topic Objective
To discuss the importance of setting permissions to use user-defined functions.

Lead-in
The security model for user-defined functions is similar to that of views.

- **Need CREATE FUNCTION Permission**
- **Need EXECUTE Permission**
- **Need REFERENCE Permission on Cited Tables, Views, or Functions**
- **Must Own the Function to Use in CREATE or ALTER TABLE Statement**

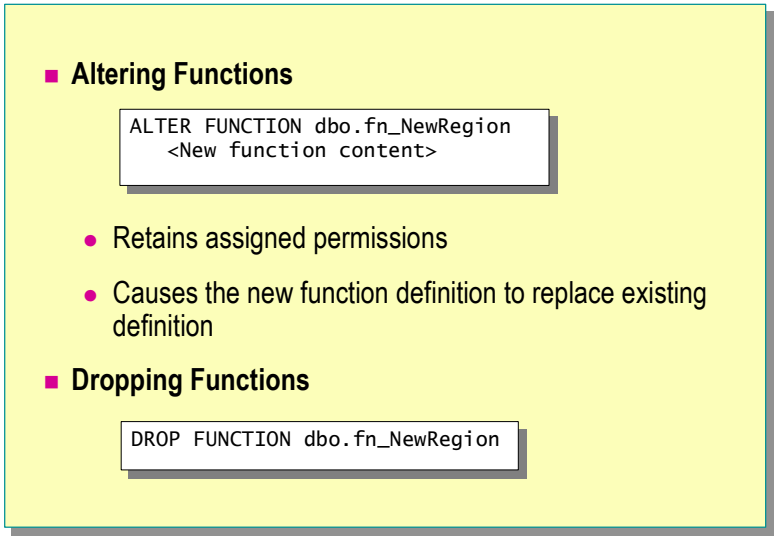
The permission requirements for user-defined functions are similar to that of other database objects.

- You must have CREATE FUNCTION permission to create, alter, or drop user-defined functions.
- Users other than the owner must be granted EXECUTE permission on a function before they can use it in a Transact-SQL statement.
- If the function is being schema-bound, you must have REFERENCE permission on tables, views, and functions referenced by the function. REFERENCE permissions can be granted through the GRANT statement to views and user-defined functions, as well as tables.
- If a CREATE TABLE or ALTER TABLE statement references a user-defined function in a CHECK constraint, DEFAULT clause, or computed column, the table owner must also own the function.

Altering and Dropping User-defined Functions

Topic Objective
To describe the ALTER FUNCTION and the DROP FUNCTION statements.

Lead-in
You modify a user-defined function by using the ALTER FUNCTION statement.



You can alter and drop user-defined functions by using the ALTER FUNCTION statement.

The benefit of altering a function instead of dropping and recreating it is the same as it is for views and procedures. The permissions on the function remain and immediately apply to the revised function.

Altering Functions

You modify a user-defined function by using the ALTER FUNCTION statement.

Example

This example shows how to alter a function.

```
ALTER FUNCTION dbo.fn_NewRegion
   <New function content>
```

Dropping Functions

You drop a user-defined function by using the DROP FUNCTION statement.

Example

This example shows how to drop a function.

```
DROP FUNCTION dbo.fn_NewRegion
```

◆ Examples of User-defined Functions

Topic Objective
To introduce the topics in this section.

Lead-in
This section describes the three types of user -defined functions.

- Using a Scalar User-defined Function
- Example of a Scalar User-defined Function
- Using a Multi-Statement Table-valued Function
- Example of a Multi-Statement Table-valued Function
- Using an In-Line Table-valued Function
- Example of an In-Line Table-valued Function

This section describes the three types of user-defined functions. It describes their purpose and provides examples of the syntax that you can use to create and call them.

Using a Scalar User-defined Function

Topic Objective
To describe how a scalar function works.

Lead-in
A scalar user-defined function is similar to built-in functions.

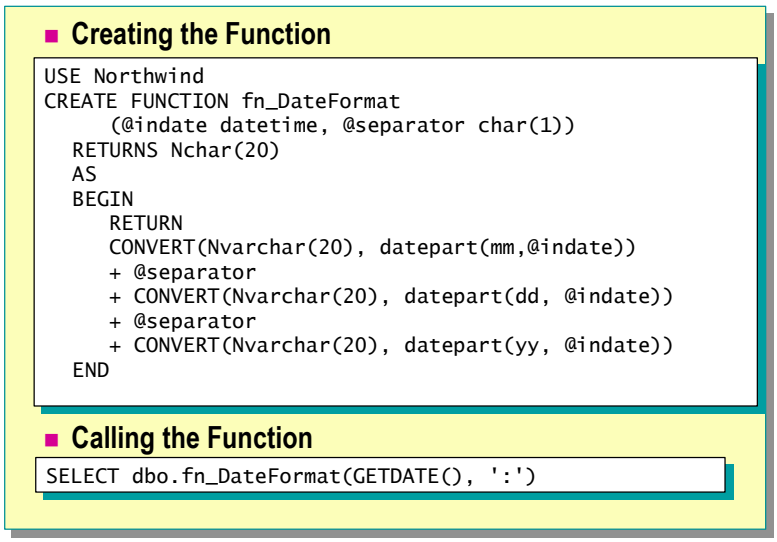
- **RETURNS Clause Specifies Data Type**
- **Function Is Defined Within a BEGIN and END Block**
- **Return Type Is Any Data Type Except text, ntext, image, cursor, or timestamp**

A scalar function returns a single data value of the type defined in a RETURNS clause. The body of the function, defined in a BEGIN…END block, contains the series of Transact-SQL statements that return the value. The return type can be any data type except **text**, **ntext**, **image**, **cursor**, or **timestamp**.

Example of a Scalar User-defined Function

Topic Objective
To provide an example of a scalar user-defined function for class discussion.

Lead-in
Here is an example of a scalar user-defined function.



A scalar user-defined function is similar to a built-in function. After you create it, you can reuse it.

Example

This example creates a user-defined function that receives date and column separators as variables and reformats the date as a character string.

```
USE Northwind
GO
CREATE FUNCTION fn_DateFormat
   (@indate datetime, @separator char(1))
RETURNS Nchar(20)
AS
BEGIN
   RETURN
   CONVERT(Nvarchar(20), datepart(mm,@indate))
   + @separator
   + CONVERT(Nvarchar(20), datepart(dd, @indate))
   + @separator
   + CONVERT(Nvarchar(20), datepart(yy, @indate))
END
```

You can call a scalar user-defined function in the same way that you do a built-in function.

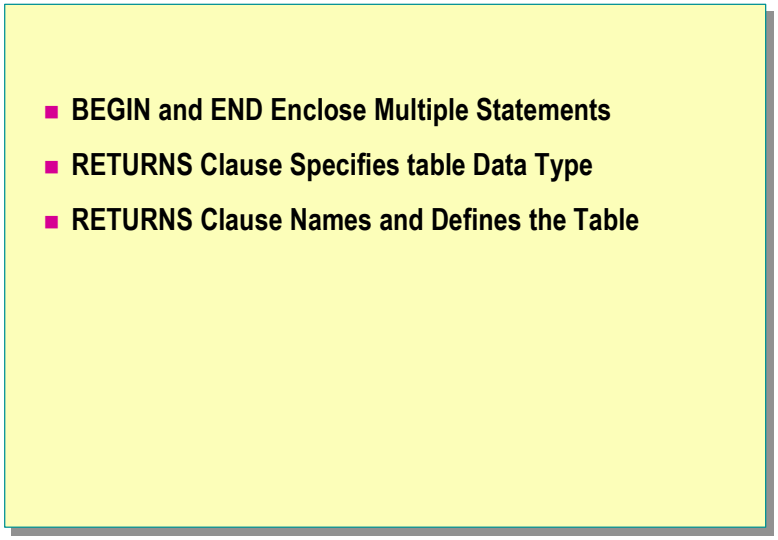
```
SELECT dbo.fn_DateFormat(GETDATE(), ':')
```

Delivery Tip
This example shows how a nondeterministic function such as GETDATE() can be used when calling a user-defined function, even though it cannot be used inside a user-defined function.

Using a Multi-Statement Table-valued Function

Topic Objective
To describe how a multi-statement table-valued function works.

Lead-in
A multi-statement table-valued function is a combination of a view and a stored procedure.



A multi-statement table-valued function is a combination of a view and a stored procedure. You can use user-defined functions that return a table to replace stored procedures or views.

A table-valued function (like a stored procedure) can use complex logic and multiple Transact-SQL statements to build a table. In the same way that you use a view, you can use a table-valued function in the FROM clause of a Transact-SQL statement.

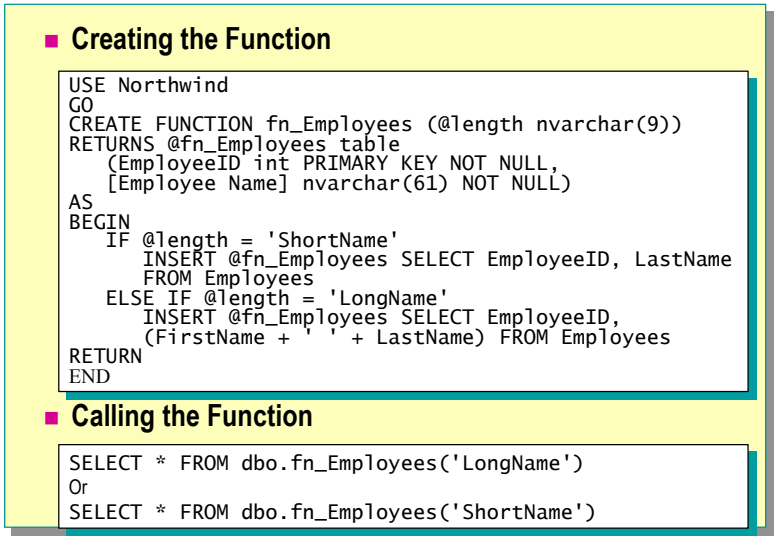
When using a multi-statement table-valued function, consider the following facts:

- The BEGIN and END delimit the body of the function.
- The RETURNS clause specifies **table** as the data type returned.
- The RETURNS clause defines a name for the table and defines the format of the table. The scope of the return variable name is local to the function.

Example of a Multi-Statement Table-valued Function

Topic Objective
To provide an example of a multi-statement table-valued function for class discussion.

Lead-in
Here is an example of a multi-statement table-valued function.



**********ILLEGAL FOR NON-TRAINER USE**********

You can create functions by using many statements that perform complex operations.

Example

This example creates a multi-statement table-valued function that returns the last name or both the first and last names of an employee, depending on the parameter provided.

```
USE Northwind
GO

CREATE FUNCTION fn_Employees
   (@length nvarchar(9))
RETURNS @fn_Employees TABLE
   (EmployeeID int PRIMARY KEY NOT NULL,
   [Employee Name] Nvarchar(61) NOT NULL)
AS
BEGIN
   IF @length = 'ShortName'
      INSERT @fn_Employees SELECT EmployeeID, LastName
      FROM Employees
   ELSE IF @length = 'LongName'
      INSERT @fn_Employees SELECT EmployeeID,
      (FirstName + ' ' + LastName) FROM Employees
RETURN
END
```

You can call the function instead of a table or view.

```
SELECT * FROM dbo.fn_Employees('LongName')
```

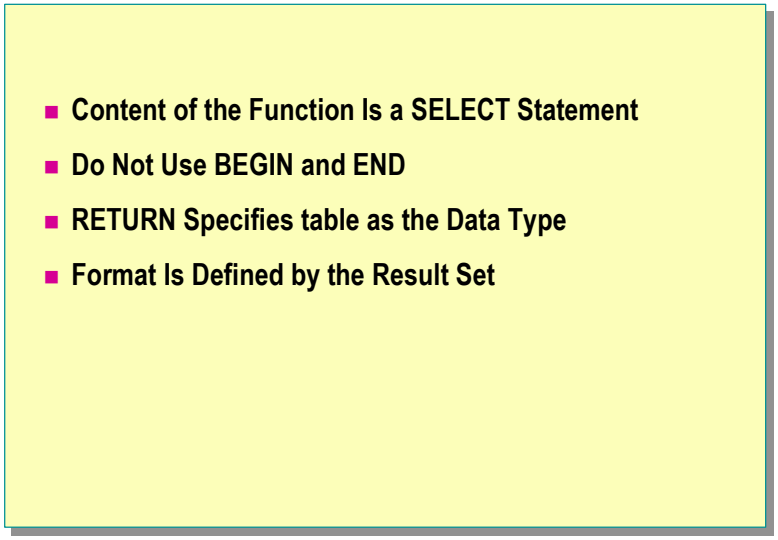
or

```
SELECT * FROM dbo.fn_Employees('ShortName')
```

Using an In-Line Table-valued Function

Topic Objective
To describe how an in-line table-valued function works.

Lead-in
An in-line table-valued function can contain only a single SELECT statement.



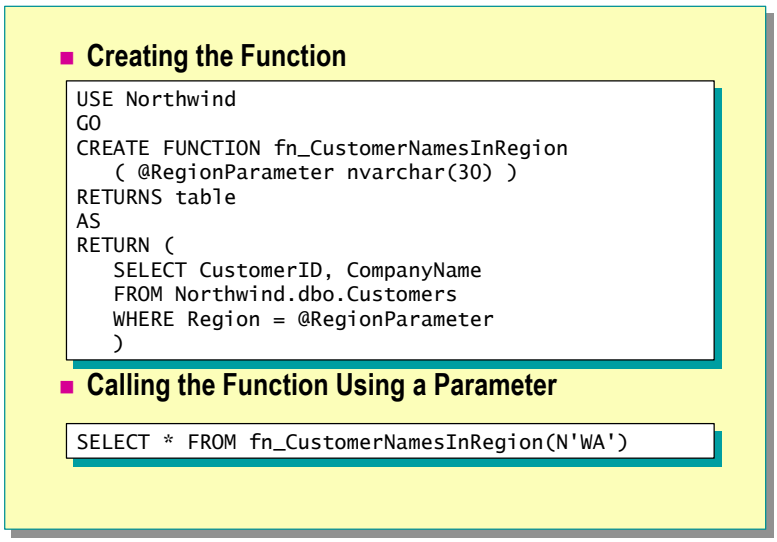
In-line user-defined functions return a table and are referenced in the FROM clause, just like a view. When using in-line user-defined functions, consider the following facts and guidelines:

- The RETURN clause contains a single SELECT statement in parentheses. The result set of the SELECT statement forms the table that the function returns. The SELECT statement used in an in-line function is subject to the same restrictions as SELECT statements used in views.
- BEGIN and END do not delimit the body of the function.
- RETURN specifies **table** as the data type returned.
- You do not have to define the format of a return variable, because it is set by the format of the result set of the SELECT statement in the RETURN clause.

Example of an In–Line Table-valued Function

Topic Objective
To provide an example of an in-line table-valued function for class discussion.

Lead-in
Here is an example of an in-line table-valued function.



Delivery Tip
Point out that you cannot create a view like:
CREATE VIEW CustView AS
SELECT <fields> FROM Customers WHERE Region = @RegionParameter

You can use in-line functions to achieve the functionality of parameterized views.

You are not allowed to include a user-provided parameter within the view when you create it. You can usually resolve this by providing a WHERE clause when calling the view. However, this may require building a string for dynamic execution, which can increase the complexity of the application. You can achieve the functionality of a parameterized view by using an in-line table-valued function.

Example

This example creates an in-line table-valued function that takes a region value as a parameter.

```
USE Northwind
GO

CREATE FUNCTION fn_CustomerNamesInRegion
   ( @RegionParameter nvarchar(30) )
RETURNS table
AS
RETURN (
   SELECT CustomerID, CompanyName
   FROM Northwind.dbo.Customers
   WHERE Region = @RegionParameter
   )
```

To call the function, provide the function name as the FROM clause and provide a region value as a parameter.

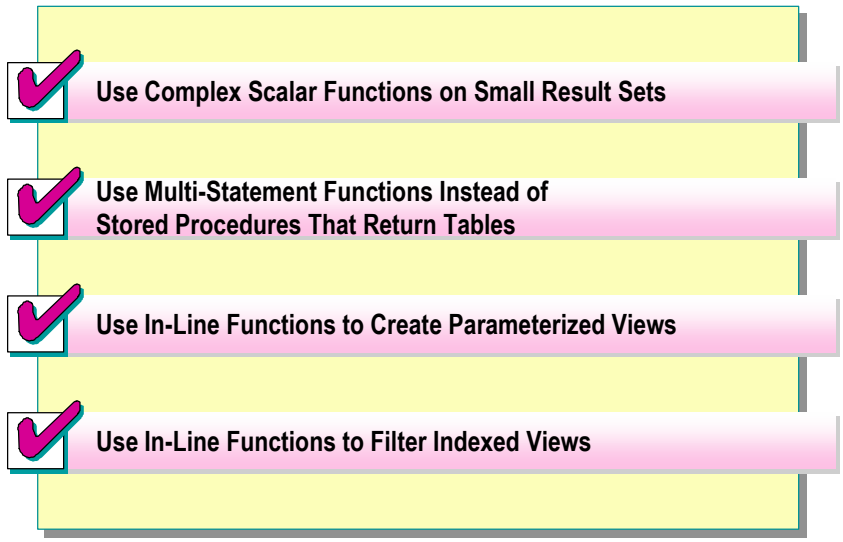
```
SELECT * FROM fn_CustomerNamesInRegion(N'WA')
```

Tip In-line functions can greatly increase performance when used with indexed views. SQL Server performs complex aggregation and join operations when the index is created. Subsequent queries can use an in-line function with a parameter to filter rows from the simplified, stored result set.

Recommended Practices

Topic Objective
To present recommended practices for using user-defined functions.

Lead-in
The following suggestions are recommended practices for using user-defined functions.



The following recommended practices should help you implement user-defined functions:

- Use complex scalar functions on small result sets. User-defined functions provide a way for you to encapsulate complex reasoning into a simple query, but if all users of the function do not appreciate the complexity of the underlying calculation, the function could result in time-consuming calculations that the user does not see. Do not apply a complex aggregation on each member of a large result set.
- Use multi-statement functions instead of stored procedures that return tables. Writing stored procedures that return tables as multi-statement user-defined functions can improve efficiency.
- Use in-line functions to create views by using parameters. Using parameters with in-line functions can simplify references to tables and views.
- Use in-line functions to filter views. Using in-line functions with indexed views can greatly increase performance.

Additional information on the following topics is available in SQL Server Books Online.

Topic	Search on
Creating user-defined functions	"CREATE FUNCTION"
Describing the purpose and types of user-defined functions	"user-defined functions"
Definition and list of deterministic and nondeterministic functions.	"nondeterministic functions"

Lab A: Creating User-defined Functions

Topic Objective
To introduce the lab.

Lead-in
In this lab, you will create user-defined functions.

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Create scalar user-defined functions.
- Create multi-statement table-valued user-defined functions.
- Create in-line table-valued user-defined functions.

Prerequisites

Before working on this lab, you must have:

- Script files for this lab, which are located in C:\Moc\2073A\Labfiles\L10.
- Answer files for this lab, which are located in C:\Moc\2073A\Labfiles\L10\Answers.

Lab Setup

To complete this lab, you must have either:

- Completed the prior lab, or
- Executed the C:\Moc\2073A\Batches\Restore10.cmd batch file.

  This command file restores the **ClassNorthwind** database to a state required for this lab.

For More Information

If you require help with executing files, search SQL Query Analyzer Help for "Execute a query".

Other resources that you can use include:

- The **Northwind** database schema.
- Microsoft SQL Server Books Online.

Scenario

The organization of the classroom is meant to simulate that of a worldwide trading firm named Northwind Traders. Its fictitious domain name is nwtraders.msft. The primary DNS server for nwtraders.msft is the instructor computer, which has an Internet Protocol (IP) address of 192.168.*x*.200 (where *x* is the assigned classroom number). The name of the instructor computer is London.

The following table provides the user name, computer name, and IP address for each student computer in the fictitious **nwtraders.msft** domain. Find the user name for your computer, and make a note of it.

User name	Computer name	IP address
SQLAdmin1	Vancouver	192.168.*x*.1
SQLAdmin2	Denver	192.168.*x*.2
SQLAdmin3	Perth	192.168.*x*.3
SQLAdmin4	Brisbane	192.168.*x*.4
SQLAdmin5	Lisbon	192.168.*x*.5
SQLAdmin6	Bonn	192.168.*x*.6
SQLAdmin7	Lima	192.168.*x*.7
SQLAdmin8	Santiago	192.168.*x*.8
SQLAdmin9	Bangalore	192.168.*x*.9
SQLAdmin10	Singapore	192.168.*x*.10
SQLAdmin11	Casablanca	192.168.*x*.11
SQLAdmin12	Tunis	192.168.*x*.12
SQLAdmin13	Acapulco	192.168.*x*.13
SQLAdmin14	Miami	192.168.*x*.14
SQLAdmin15	Auckland	192.168.*x*.15
SQLAdmin16	Suva	192.168.*x*.16
SQLAdmin17	Stockholm	192.168.*x*.17
SQLAdmin18	Moscow	192.168.*x*.18
SQLAdmin19	Caracas	192.168.*x*.19
SQLAdmin20	Montevideo	192.168.*x*.20
SQLAdmin21	Manila	192.168.*x*.21
SQLAdmin22	Tokyo	192.168.*x*.22
SQLAdmin23	Khartoum	192.168.*x*.23
SQLAdmin24	Nairobi	192.168.*x*.24

Estimated time to complete this lab: 30 minutes

Exercise 1
Creating a Scalar User-defined Function

The products that your company sells are subject to varying tax rates that are based on the product categories. Some products such as beverages are heavily taxed, and some products such as condiments have no tax. You decide to create a user-defined function to encapsulate the tax logic. This user-defined function gives you a central point for administering the tax rates and reducing the number of locations in which you must repeat and maintain the logic.

In this exercise, you will create and execute a script that creates a scalar user-defined function. After you create the function, you will test it to verify that it works.

► To create a scalar user-defined function

In this procedure, you will create a trigger by executing a script file.

1. Log on to the **NWTraders** classroom domain by using the information in the following table.

Option	Value
User name	**SQLAdmin*x*** (where *x* corresponds to your computer name as designated in the nwtr**aders.msft** classroom domain)
Password	**password**

2. Open SQL Query Analyzer and, if requested, log in to the (local) server with Microsoft Windows® authentication.

   You have permission to log in to and administer SQL Server because you are logged as **SQLAdmin*x***, which is a member of the Microsoft Windows 2000 local group, Administrators. All members of this group are automatically mapped to the SQL Server **sysadmin** role.

3. In the **DB** list, click **ClassNorthwind**.

4. Open, examine, and execute TaxRate.sql in the C:\Moc\2073A\Labfiles\L10 folder.

   This script demonstrates a CASE statement that calculates a tax rate for each product, based on its product category.

5. Create a scalar user-defined function called **fn_TaxRate** to encapsulate the CASE statement. Accept a parameter as **@ProdID int** and a return a **numeric(5,4)** data type.

   C:\Moc\2073A\Labfiles\L10\Answers\fn_TaxRate.sql is a completed script containing this function.

► To test the function

In this procedure, you will test the function that you just created by selecting columns from the **Products** table.

- Execute the following SELECT statement to select **ProductName**, **UnitPrice**, **CategoryID**, **TaxRate**, and a calculated **PriceWithTax** value from the **Products** table:

```
SELECT ProductName, UnitPrice, CategoryID,
   ClassNorthwind.dbo.fn_TaxRate(ProductID) AS TaxRate,
   UnitPrice * ClassNorthwind.dbo. fn_TaxRate(ProductID)
      AS PriceWithTax
FROM Products
```

  The **TaxRate** column should contain values of 1.00, 1.05, or 1.10 for each product.

  The **PriceWithTax** column should contain the **UnitPrice** multiplied by the **TaxRate**.

Exercise 2
Creating a Multi-Statement Table-valued User-defined Function

In this exercise, you will create a multi-statement table-valued user-defined function that queries the **Employees** table in the **ClassNorthwind** database and that shows all direct and non-direct reports.

► To create a multi-statement table-valued user-defined function

The **Employees** table in the **ClassNorthwind** database contains a column called **ReportsTo** that contains the employee ID number of the manager to whom each employee reports.

In this exercise, you will create a user-defined function that takes the **EmployeeID** number of a manager as a parameter and iterates through the **Employee** table, gathering employees that report to that designated manager at any level.

- Open, review, and execute fn_FindReports.sql in the C:\Moc\2073A\Labfiles\L10 folder.

  This script creates a function called **fn_FindReports** that returns a table of reporting employees.

► To test the function

In this procedure, you will test the function that you just created by selecting columns from the **fn_FindReports** function.

1. Execute the following SELECT statement to select **EmployeeID**, **Name**, **Title**, and **MgrEmployeeID** columns from the **fn_FindReports** function.

   ```
   SELECT EmployeeID, [Name], Title, MgrEmployeeID
   FROM dbo.fn_FindReports(5)
   ```

   This SELECT statement returns the employees who report to Steven Buchanan (**EmployeeID** 5).

2. Execute the following SELECT statement to select the **EmployeeID**, **Name**, **Title**, and **MgrEmployeeID** from the **fn_FindReports** function.

   ```
   SELECT EmployeeID, Name, Title, MgrEmployeeID
   FROM dbo.fn_FindReports(2)
   ```

   This statement returns a table containing the names of employees who report to Andrew Fuller (**EmployeeID** 2).

   C:\Moc\2073A\Labfiles\L10\Answers\Call_fn_FindReports.sql is a completed script containing this statement.

Exercise 3
Creating an In-Line Table-valued User-defined Function

In this exercise, you will create an in-line table-valued user-defined function as an alternative to a view. This function, called **fn_LargeFreight**, will accept a dollar amount as a parameter and return freight orders with amounts that exceed the dollar amount.

► To create an in-line table-valued user-defined function

The **Orders** table in the **ClassNorthwind** database contains a column called **Freight** that contains the dollar amount charged for freight for each order. You want to join that table with the **Shippers** table to return both order and shipping information. You also want to filter the result set to show only those orders that have expensive freight charges.

In this procedure, you will create an in-line table-valued user-defined function that serves as a parameterized view.

- Create an in-line table-valued user-defined function called **fn_LargeFreight** that accepts a parameter called **@FreightAmt** of data type **money** and returns the output from the following SELECT statement:

```
SELECT S.ShipperID, S.CompanyName,
   O.OrderID, O.ShippedDate, O.Freight
FROM Shippers AS S JOIN Orders AS O
   ON S.ShipperID = O.ShipVia
WHERE O.Freight > @FreightAmt
```

  C:\Moc\2073A\Labfiles\L10\Answers\fn_LargeFreight.sql is a completed script containing this function.

► To test the function

In this procedure, you will test the function that you just created by selecting columns from the **fn_LargeFreight** function.

- Execute the following SELECT statement to select rows from the **fn_LargeFreight** function that have dollar sales greater than $600.

```
SELECT * FROM fn_LargeFreight(600)
```

  C:\Moc\2073A\Labfiles\L10\Answers\Call_fn_LargeFreight.sql is a completed script containing this statement.

Review

Topic Objective
To reinforce module objectives by reviewing key points.

Lead-in
The review questions cover some of the key concepts taught in the module.

- What Is a User-defined Function?
- Defining User-defined Functions
- Examples of User-defined Functions

Delivery Tip
Use these questions to review module topics.

Ask students whether they have any questions.

1. Describe the three types of user-defined functions.

   Scalar functions are similar to built-in functions.

   Multi-statement table-valued functions are similar to stored procedures.

   In-line table-valued functions are similar to views.

2. What built-in functions are not permitted in the body of a user-defined function?

   You cannot use non-deterministic functions such as GETDATE().

3. What types of user-defined functions require you to specify the names and data types of the output columns?

   Scalar and multi-statement user-defined functions require complete column descriptions, including column name and data type definitions. In-line functions use the column names and implicit data types of the output columns.