**How to: Create Stored Procedures and User-Defined Functions**

**Visual Studio 2013**

[Other Versions](javascript:;)

http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2012](http://msdn.microsoft.com/en-us/library/yt786c0h(d=printer,v=vs.110).aspx)
* [Visual Studio 2010](http://msdn.microsoft.com/en-us/library/yt786c0h(d=printer,v=vs.100).aspx)
* [Visual Studio .NET 2003](http://msdn.microsoft.com/en-us/library/yt786c0h(d=printer,v=vs.71).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/yt786c0h(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/yt786c0h(d=printer,v=vs.80).aspx)

You can use [Server Explorer](http://msdn.microsoft.com/en-us/library/x603htbk.aspx) to create stored procedures. Stored procedures can define complex business rules, control data modification, limit access through security permissions, provide transaction integrity, and generally do the database work your application requires.

With Microsoft SQL Server 2000 and later, you can also use Server Explorer to create user-defined functions. A user-defined function is a routine that encapsulates useful logic for use in other queries. While views are limited to a single SELECT statement, user-defined functions can have multiple SELECT statements and provide more powerful logic than is possible with views.

User-defined functions always return a value. Depending on the type of value it returns, each user-defined function falls into one of three categories:

* Scalar-valued function   A user-defined function can return a scalar value such as an integer or a timestamp. If a function returns a scalar value, you can use it in a query anywhere you would use a column name.
* Inline function   If a user-defined function contains a single SELECT statement and that statement is updatable, then the tabular result returned by the function is also updatable. Such functions are called inline functions. When an inline function returns a table, you can use that function in the FROM clause of another query. For more information, see [How to: Create Queries using Something Besides a Table](http://msdn.microsoft.com/en-us/library/77h3c21y.aspx).
* Table-valued function   If a user-defined function contains more than one SELECT statement, or contains a SELECT statement that is not updatable, then the tabular result returned by that function is not updatable. When a table-valued function returns a table, you can use that function in the FROM clause of another query.

|  |
| --- |
| **NoteNote** |
| In the following examples, dbo is an acronym for database owner, and is used to qualify the stored procedure and user-defined function name. The dbo is a user that has implied permissions to perform all activities in the database. Any object created by any member of the sysadmin role belongs to dbo automatically. In the following examples, the dbo name qualifier is included. |

|  |
| --- |
| **NoteNote** |
| Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Customizing Development Settings](http://msdn.microsoft.com/en-us/library/zbhkx167.aspx). |

**To create a new stored procedure**

1. In Server Explorer, right-click the Stored Procedures folder or any stored procedure in that folder.
2. Choose Add New Stored Procedure on the shortcut menu.

A new stored procedure is created with the following skeleton SQL statements:

CREATE PROCEDURE dbo.StoredProcedure1

/\*

(

@parameter1 datatype = default value,

@parameter2 datatype OUTPUT

)

\*/

AS

/\* SET NOCOUNT ON \*/

RETURN

1. You can replace StoredProcedure1 in the first line with the name of the new procedure. For example, you might use MyProcedure as the name:

CREATE PROCEDURE dbo.MyProcedure

|  |
| --- |
| **NoteNote** |
| Stored procedures must have unique names. If you choose a name that is already assigned to another stored procedure, an error message warns you that a stored procedure with that name already exists. |

1. Write the remaining procedure text in SQL.

For more information and examples of stored procedures, see the documentation for your database server. If you are using Microsoft SQL Server, see "CREATE PROCEDURE" in SQL Server Books Online.

**To create a new user-defined function**

1. In Server Explorer, right-click the Functions folder or any function in that folder.
2. Point to Add New and then choose Inline Function, Table-valued Function, or Scalar-valued Function on the shortcut menu.

|  |
| --- |
| **NoteNote** |
| You cannot modify the skeleton SQL statements for a new function into the SQL statements for a different function type and save the results. For example, if you begin with an inline function, you cannot modify the SQL statements to a scalar-valued function. The save process will fail. |

1. A new user-defined function is created using skeleton SQL statements. For example, choosing Scalar-valued Function displays the following skeleton SQL statements:

CREATE FUNCTION dbo.Function

(

/\*

@parameter1 datatype = default value,

@parameter2 datatype

\*/

)

RETURNS /\* datatype \*/

AS

BEGIN

/\* sql statement ... \*/

RETURN /\* value \*/

END

1. You can replace Function1 in the first line with the name of the new function. For example, you might use MyFunction as the name:

CREATE FUNCTION dbo.MyFunction

|  |
| --- |
| **NoteNote** |
| User-defined functions must have unique names. If you choose a name that is already assigned to another function, an error message warns you that a function with that name already exists. |

1. Write the remaining function text in SQL.

|  |
| --- |
| **NoteNote** |
| If the shortcut menu does not offer the options you need, the version of Visual Studio you are using may not support this feature. For more information see [Visual Database Tools Editions](http://msdn.microsoft.com/en-us/library/8czxd086.aspx). |

**Rewriting Stored Procedures as Functions**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms187650(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms187650(d=printer,v=sql.90).aspx)

This topic describes how to determine whether to rewrite existing stored procedure logic as user-defined functions. For example, if you want to invoke a stored procedure directly from a query, repackage the code as a user-defined function.

In general, if the stored procedure returns a, single, result set, define a table-valued function. If the stored procedure computes a scalar value, define a scalar function.

[Criteria for Table-Valued Functions](javascript:void(0))

If a stored procedure meets the following criteria, it is a good candidate for being rewritten as a table-valued function:

* The logic is expressible in a single SELECT statement but is a stored procedure, rather than a view, only because of the need for parameters. This scenario can be handled with an inline table-valued function.
* The stored procedure does not perform update operations, except to table variables.
* There is no need for dynamic EXECUTE statements
* The stored procedure returns one result set.
* The primary purpose of the stored procedure is to build intermediate results that are to be loaded into a temporary table, which is then queried in a SELECT statement. INSERT...EXEC statements can be written using table-valued functions. For example, consider the following sequence:

INSERT #temp EXEC sp\_getresults

SELECT ...

FROM #temp, t1

WHERE ...

The **sp\_getresults** stored procedure can be rewritten as a table-valued function, for example **fn\_results()**, which means the preceding statements can be rewritten as:

INSERT #temp

SELECT ...

FROM fn\_results(), t1

WHERE ...

[Rewriting Extended Stored Procedures Using CLR](javascript:void(0))

CLR functions provide a more reliable and scalable alternative to extended stored procedures. Many extended stored procedures perform some computational task that is harder to express in Transact-SQL. Such stored procedures can be rewritten using the CLR given the benefits described above. Further, extended stored procedures that return result sets by accessing an external resource such as a file or Web Service can be rewritten using a CLR table-valued function. For more information, see [Creating CLR Functions](http://technet.microsoft.com/en-us/library/ms189876(v=sql.105).aspx).

**User Defined Functions and Stored Procedures**

**SQL Server 2012**

[Other Versions](javascript:;)



* [SQL Server 2008 R2](http://technet.microsoft.com/en-us/library/ms345075(d=printer,v=sql.105).aspx)
* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms345075(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms345075(d=printer,v=sql.90).aspx)

With ADOMD.NET server objects, you can create user defined function (UDF) or stored procedures for Microsoft SQL Server Analysis Services that interact with metadata and data from the server. These in-process methods are called through Multidimensional Expressions (MDX) or Data Mining Extensions (DMX) statements to provide added functionality without the latencies associated with network communications.

[UDF Examples](javascript:void(0))

A UDF is a method that can be called in the context of an MDX or DMX statement, can take any number of parameters, and can return any type of data.

A UDF created using MDX is similar to one created for DMX. The main difference is that certain properties of the [Context](http://technet.microsoft.com/en-us/library/microsoft.analysisservices.adomdserver.context.aspx) object, such as the [CurrentCube](http://technet.microsoft.com/en-us/library/microsoft.analysisservices.adomdserver.context.currentcube.aspx) and [CurrentMiningModel](http://technet.microsoft.com/en-us/library/microsoft.analysisservices.adomdserver.context.currentminingmodel.aspx) properties, are available only for one scripting language or the other.

The following examples show how to use a UDF to return a node description, filter tuples, and apply a filter to a tuple.

**Returning a Node Description**

The following example creates a UDF that returns the node description for a specified node. The UDF uses the current context in which it is being run, and uses a DMX FROM clause to retrieve the node from the current mining model.

public string GetNodeDescription(string nodeUniqueName)

{

return Context.CurrentMiningModel.GetNodeFromUniqueName(nodeUniqueName).Description;

}

Once deployed, the previous UDF example can be called by the following DMX expression, which retrieves the most-likely prediction node. The description contains information that describes the conditions that make up the prediction node.

select Cluster(), SampleAssembly.GetNodeDescription( PredictNodeId(Cluster()) ) FROM [Customer Clusters]

**Returning Tuples**

The following example takes a set and a return count, and randomly retrieves tuples from the set, returning a final subset:

public Set RandomSample(Set set, int returnCount)

{

//Return the original set if there are fewer tuples

//in the set than the number requested.

if (set.Tuples.Count <= returnCount)

return set;

System.Random r = new System.Random();

SetBuilder returnSet = new SetBuilder();

//Retrieve random tuples until the return set is filled.

int i = set.Tuples.Count;

foreach (Tuple t in set.Tuples)

{

if (r.Next(i) < returnCount)

{

returnCount--;

returnSet.Add(t);

}

i--;

//Stop the loop if we have enough tuples.

if (returnCount == 0)

break;

}

return returnSet.ToSet();

}

The previous example is called in the following MDX example. In this MDX example, five random states or provinces are retrieved from the Adventure Works database.

SELECT SampleAssembly.RandomSample([Geography].[State-Province].Members, 5) on ROWS,

[Date].[Calendar].[Calendar Year] on COLUMNS

FROM [Adventure Works]

WHERE [Measures].[Reseller Freight Cost]

**Applying a Filter to a Tuple**

In the following example, a UDF is defined that takes a set, and applies a filter to each tuple in the set, using the Expression object. Any tuples that conform to the filter will be added to a set that is returned.

C#

public static Set FilterSet(Set set, String filterExpression)

{

Expression expr = new Expression(filterExpression);

SetBuilder resultSetBuilder = new SetBuilder();

foreach (Tuple tuple in set)

{

if ((bool)(expr.Calculate(tuple)))

{

resultSetBuilder.Add(tuple);

}

}

return resultSetBuilder.ToSet();

}

The previous example is called in the following MDX example, which filters the set to cities with names beginning with 'A'.

Select Measures.Members on Rows,

SampleAssembly.FilterSet([Customer].[Customer Geography].[City], "[Customer].[Customer Geography].[City].CurrentMember.Name < 'B'") on Columns

From [Adventure Works]

[Stored Procedure Example](javascript:void(0))

In the following example, an MDX-based stored procedure uses AMO to create partitions, if needed, for Internet Sales.

C#

public static void CreateInternetSalesMeasureGroupPartitions()

{

//Check the current state of the data warehouse and

//create partitions with AMO if necessary

#region Retrieve order date of last sales transaction

// Open a connection to the data warehouse

// TODO: Change the first string parameter to reflect the right server\instance.

SqlConnection conn = new SqlConnection(string.Format("data source={0};initial catalog={1};Integrated Security=SSPI", "server\\instance", Context.CurrentDatabaseName));

conn.Open();

// Create a command

SqlCommand cmd = new SqlCommand();

cmd.Connection = conn;

// Get the order date key of the last internet sale

int lastInternetSaleDateKey = 0;

cmd.CommandText = "select max(OrderDateKey) from FactInternetSales";

lastInternetSaleDateKey = (int)cmd.ExecuteScalar();

// Get the order date key of the last reseller sale

int lastResellerSaleDateKey = 0;

cmd.CommandText = "select max(OrderDateKey) from FactResellerSales";

lastResellerSaleDateKey = (int)cmd.ExecuteScalar();

#endregion

#region Create partitions

// Connect to the calling session

Server svr = new Server();

svr.Connect("\*");

// Get the Adventure Works cube

Database db = svr.Databases.GetByName(Context.CurrentDatabaseName);

Cube cube = db.Cubes[0];

MeasureGroup mg;

int maxOrderDateKey;

mg = cube.MeasureGroups.GetByName("Internet Sales");

maxOrderDateKey = 0;

foreach (Partition part in mg.Partitions)

{

maxOrderDateKey = Math.Max(maxOrderDateKey, Convert.ToInt32(

part.Annotations.Find("LastOrderDateKey").Value.Value,

System.Globalization.CultureInfo.InvariantCulture));

}

if (maxOrderDateKey < lastInternetSaleDateKey)

{

Partition part = mg.Partitions.Add("Internet\_Sales\_"

+ lastInternetSaleDateKey);

part.StorageMode = StorageMode.Molap;

part.Source = new QueryBinding(db.DataSources[0].ID,

"SELECT \* FROM [dbo].[FactInternetSales] WHERE OrderDateKey > '"

+ maxOrderDateKey + "' and OrderDateKey <= '" + lastInternetSaleDateKey + "'");

part.Annotations.Add("LastOrderDateKey", Convert.ToString(lastInternetSaleDateKey,

System.Globalization.CultureInfo.InvariantCulture));

part.Update();

part.Process();

}

svr.Disconnect();

#endregion

}