SQL Server 2008 Books Online (December 2008)

**Stored Procedures (Database Engine)**

When you create an application with Microsoft SQL Server, the Transact-SQL programming language is the primary programming interface between your applications and the Microsoft SQL Server database. When you use Transact-SQL programs, two methods are available for storing and executing the programs.

* You can store the programs locally and create applications that send the commands to SQL Server and process the results.
* You can store the programs as stored procedures in SQL Server and create applications that execute the stored procedures and process the results.
* http://i.msdn.microsoft.com/Global/Images/clear.gif In This Section

This section provides the information necessary to understand, design, and implement stored procedures.

[Understanding Stored Procedures](http://msdn.microsoft.com/en-us/library/ms191428.aspx)

Explains stored procedure concepts and includes descriptions and examples of each stored procedure type available in SQL Server.

[Designing Stored Procedures (Database Engine)](http://msdn.microsoft.com/en-us/library/ms191132.aspx)

Provides specific guidelines, rules, and restrictions you should know when designing a stored procedure.

[Implementing Stored Procedures](http://msdn.microsoft.com/en-us/library/ms187451.aspx)

Describes how to create, modify, or delete a stored procedure.

**Understanding Stored Procedures**

**Stored Procedure Basics**

Stored procedures in Microsoft SQL Server are similar to procedures in other programming languages in that they can:

* Accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch.
* Contain programming statements that perform operations in the database, including calling other procedures.
* Return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure).

You can use the Transact-SQL EXECUTE statement to run a stored procedure. Stored procedures are different from functions in that they do not return values in place of their names and they cannot be used directly in an expression.

The benefits of using stored procedures in SQL Server rather than Transact-SQL programs stored locally on client computers are:

* They are registered at the server.
* They can have security attributes (such as permissions) and ownership chaining, and certificates can be attached to them.  
  Users can be granted permission to execute a stored procedure without having to have direct permissions on the objects referenced in the procedure.
* They can enhance the security of your application.  
  Parameterized stored procedures can help protect your application from SQL Injection attacks. For more information see [SQL Injection](http://msdn.microsoft.com/en-us/library/ms161953.aspx).
* They allow modular programming.   
  You can create the procedure once, and call it any number of times in your program. This can improve the maintainability of your application and allow applications to access the database in a uniform manner.
* They are named code allowing for delayed binding.  
  This provides a level of indirection for easy code evolution.
* They can reduce network traffic.   
  An operation requiring hundreds of lines of Transact-SQL code can be performed through a single statement that executes the code in a procedure, rather than by sending hundreds of lines of code over the network.

**Types of Stored Procedures**

There are many types of stored procedures available in Microsoft SQL Server. This topic briefly describes each stored procedure type and includes an example of each.

http://i.msdn.microsoft.com/Global/Images/clear.gif User-defined Stored Procedures

Stored procedures are modules or routines that encapsulate code for reuse. A stored procedure can take input parameters, return tabular or scalar results and messages to the client, invoke data definition language (DDL) and data manipulation language (DML) statements, and return output parameters. In SQL Server 2008, a stored procedure can be of two types: Transact-SQL or CLR.

### Transact-SQL

A Transact-SQL stored procedure is a saved collection of Transact-SQL statements that can take and return user-supplied parameters. For example, a stored procedure might contain the statements needed to insert a new row into one or more tables based on information supplied by the client application. Or, the stored procedure might return data from the database to the client application. For example, an e-commerce Web application might use a stored procedure to return information about specific products based on search criteria specified by the online user.

### CLR

A CLR stored procedure is a reference to a Microsoft .NET Framework common language runtime (CLR) method that can take and return user-supplied parameters. They are implemented as public, static methods on a class in a .NET Framework assembly. For more information, see [CLR Stored Procedures](http://msdn.microsoft.com/en-us/library/ms131094.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Extended Stored Procedures

|  |
| --- |
| **ms187644.note(en-us,SQL.100).gifImportant:** |
| This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [CLR Integration](http://msdn.microsoft.com/en-us/library/ms131089.aspx) instead. |

Extended stored procedures let you create your own external routines in a programming language such as C. Extended stored procedures are DLLs that an instance of Microsoft SQL Server can dynamically load and run. Extended stored procedures run directly in the address space of an instance of SQL Server and are programmed by using the SQL Server Extended Stored Procedure API.

|  |
| --- |
| **ms187644.note(en-us,SQL.100).gifNote:** |
| CLR Integration provides a more robust and secure alternative to writing extended stored procedures. |

http://i.msdn.microsoft.com/Global/Images/clear.gif **System Stored Procedures**

Many administrative activities in SQL Server are performed through a special kind of procedure known as a system stored procedure. For example, **sys.sp\_changedbowner** is a system stored procedure. System stored procedures are physically stored in the [Resource database](http://msdn.microsoft.com/en-us/library/ms190940.aspx) and have the **sp\_** prefix. System stored procedures logically appear in the **sys** schema of every system- and user-defined database. In SQL Server 2008, GRANT, DENY, and REVOKE permissions can be applied to system stored procedures. For a complete list of system stored procedures, see [System Stored Procedures (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187961.aspx).

SQL Server supports the system stored procedures that provide an interface from SQL Server to external programs for various maintenance activities. These extended stored procedures use the **xp\_** prefix. For a complete list of extended stored procedures, see [General Extended Stored Procedures (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187389.aspx).

**Designing Stored Procedures (Database Engine)**

Almost any Transact-SQL code that can be written as a batch can be used to create a stored procedure.

http://i.msdn.microsoft.com/Global/Images/clear.gif Rules for Designing Stored Procedures

Rules for designing stored procedures include the following:

* The CREATE PROCEDURE definition itself can include any number and type of SQL statements, except for the following statements. These cannot be used anywhere within a stored procedure.

|  |  |
| --- | --- |
| CREATE AGGREGATE | CREATE RULE |
| CREATE DEFAULT | CREATE SCHEMA |
| CREATE or ALTER FUNCTION | CREATE or ALTER TRIGGER |
| CREATE or ALTER PROCEDURE | CREATE or ALTER VIEW |
| SET PARSEONLY | SET SHOWPLAN\_ALL |
| SET SHOWPLAN\_TEXT | SET SHOWPLAN\_XML |
| USE *database\_name* |  |

* Other database objects can be created within a stored procedure. You can reference an object created in the same stored procedure as long as it is created before it is referenced.
* You can reference temporary tables within a stored procedure.
* If you create a local temporary table inside a stored procedure, the temporary table exists only for the purposes of the stored procedure; it disappears when you exit the stored procedure.
* If you execute a stored procedure that calls another stored procedure, the called stored procedure can access all objects created by the first stored procedure, including temporary tables.
* If you execute a remote stored procedure that makes changes on a remote instance of Microsoft SQL Server, those changes cannot be rolled back. Remote stored procedures do not take part in transactions.
* The maximum number of parameters in a stored procedure is 2100.
* The maximum number of local variables in a stored procedure is limited only by available memory.
* Depending on available memory, the maximum size of a stored procedure is 128 megabytes (MB).

http://i.msdn.microsoft.com/Global/Images/clear.gif Qualifying Names Inside Stored Procedures

Inside a stored procedure, object names used with statements (for example, SELECT or INSERT) that are not schema-qualified default to the schema of the stored procedure. If a user who creates a stored procedure does not qualify the name of the tables or views referenced in SELECT, INSERT, UPDATE, or DELETE statements within the stored procedure, access to those tables through the stored procedure is restricted by default to the creator of the procedure.

Object names used with all Data Definition Language (DDL) statements such as CREATE, ALTER, or DROP statements, DBCC statements, EXECUTE and dynamic SQL statements should be qualified with the name of the object schema if other users are to use the stored procedure. Specifying the schema name for these objects ensures the name resolves to the same object regardless who the caller of the stored procedure is. If a schema name is not specified, SQL Server will attempt to resolve the object name first using the default schema of the caller or the user specified in the EXECUTE AS clause and then the **dbo** schema.

http://i.msdn.microsoft.com/Global/Images/clear.gif Obfuscating Procedure Definitions

To convert the original text of the CREATE PROCEDURE statement to an obfuscated format, use the WITH ENCRYPTION option. The output of the obfuscation is not directly visible in any of the system tables or views in SQL Server 2008: users without access to system tables, system views, or database files cannot retrieve the obfuscated text. However, the text is available to privileged users with direct access to database files. These users may be able to reverse engineer the obfuscation to retrieve the original text of the stored procedure definition.

http://i.msdn.microsoft.com/Global/Images/clear.gif SET Statement Options

The Database Engine saves the settings of both SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS when a Transact-SQL stored procedure is created or altered. These original settings are used when the stored procedure is executed. Therefore, any client session settings for SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS are ignored during stored procedure execution. SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS statements that occur within the stored procedure do not affect the functionality of the stored procedure.

Other SET options, such as SET ARITHABORT, SET ANSI\_WARNINGS, or SET ANSI\_PADDINGS are not saved when a stored procedure is created or altered. If the logic of the stored procedure is dependent on a particular setting, include a SET statement at the start of the procedure to ensure the proper setting. When a SET statement is executed from a stored procedure, the setting remains in effect only until the stored procedure completes. The setting is then restored to the value it had when the stored procedure was called. This allows individual clients to set the options wanted without affecting the logic of the stored procedure.

|  |
| --- |
| **ms191132.note(en-us,SQL.100).gifNote:** |
| ANSI\_WARNINGS is not honored when passing parameters in a stored procedure, user-defined function, or when declaring and setting variables in a batch statement. For example, if a variable is defined as char(3), and then set to a value larger than three characters, the data is truncated to the defined size and the INSERT or UPDATE statement succeeds. |

**Implementing Stored Procedures**

This section provides the information necessary to create, modify, and remove stored procedures.

http://i.msdn.microsoft.com/Global/Images/clear.gif In This Section

[Creating Stored Procedures (Database Engine)](http://msdn.microsoft.com/en-us/library/ms190669.aspx)

Provides guidelines for creating stored procedures.

[Creating CLR Stored Procedures](http://msdn.microsoft.com/en-us/library/ms190790.aspx)

Describes how to create CLR stored procedures.

[Deferred Name Resolution and Compilation](http://msdn.microsoft.com/en-us/library/ms190686.aspx)

Describes when and how stored procedures are parsed and compiled. Explains deferred name resolution of referenced objects when creating a stored procedure.

[Executing Stored Procedures (Database Engine)](http://msdn.microsoft.com/en-us/library/ms189915.aspx)

Provides guidelines for executing stored procedures.

[Extended Stored Procedures](http://msdn.microsoft.com/en-us/library/ms175200.aspx)

Provides guidelines for creating extended stored procedures.

[Modifying Stored Procedures](http://msdn.microsoft.com/en-us/library/ms187001.aspx)

Provides guidelines for modifying and renamed stored procedures

[Recompiling Stored Procedures](http://msdn.microsoft.com/en-us/library/ms190439.aspx)

Provides guidelines for recompiling stored procedures.

[Viewing Stored Procedures](http://msdn.microsoft.com/en-us/library/ms190464.aspx)

Describes how to view stored procedure definitions and properties.

[Deleting Stored Procedures](http://msdn.microsoft.com/en-us/library/ms189043.aspx)

Provides guidelines for deleting stored procedures.

**Creating Stored Procedures (Database Engine)**

You can create stored procedures using the CREATE PROCEDURE Transact-SQL statement.

Before creating a stored procedure, consider that:

* CREATE PROCEDURE statements cannot be combined with other SQL statements in a single batch.
* To create procedures, you must have CREATE PROCEDURE permission in the database and ALTER permission on the schema in which the procedure is being created. For CLR stored procedures, you must either own the assembly referenced in <method\_specifier>, or have REFERENCES permission on that assembly.
* Stored procedures are schema-scoped objects, and their names must follow the rules for identifiers.
* You can create a stored procedure only in the current database.

When creating a stored procedure, you should specify:

* Any input parameters and output parameters to the calling procedure or batch.
* The programming statements that perform operations in the database, including calling other procedures.
* The status value returned to the calling procedure or batch to indicate success or failure (and the reason for failure).
* Any error handling statements needed to catch and handle potential errors.   
  Error handing functions such as ERROR\_LINE and ERROR\_PROCEDURE can be specified in the stored procedure. For more information, see [Using TRY...CATCH in Transact-SQL](http://msdn.microsoft.com/en-us/library/ms179296.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Naming Stored Procedures

We recommend that you do not create any stored procedures using **sp\_** as a prefix. SQL Server uses the **sp\_** prefix to designate system stored procedures. The name you choose may conflict with some future system procedure. If your application uses nonschema qualified name references and your own procedure name conflicts with a system procedure name, your application will break because the name binds to the system procedure, not your own.

A user-defined stored procedure that has the same name as a system stored procedure and is either nonqualified or is in the **dbo** schema will never be executed; the system stored procedure will always execute instead. The following example demonstrates this behavior.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl06other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl06other');)

USE AdventureWorks;

GO

CREATE PROCEDURE dbo.sp\_who

AS

SELECT FirstName, LastName FROM Person.Contact;

GO

EXEC sp\_who;

EXEC dbo.sp\_who;

GO

DROP PROCEDURE dbo.sp\_who;

GO

Using an explicit schema qualifier also provides a slight performance advantage. Name resolution is slightly faster if the Database Engine does not have to search multiple schemas to find the procedure. For more information, see [Executing a Stored Procedure](http://msdn.microsoft.com/en-us/library/ms189915.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Temporary Stored Procedures

Private and global temporary stored procedures, analogous to temporary tables, can be created with the # and ## prefixes added to the procedure name. # denotes a local temporary stored procedure; ## denotes a global temporary stored procedure. These procedures do not exist after SQL Server is shut down.

Temporary stored procedures are useful when connecting to earlier versions of SQL Server that do not support the reuse of execution plans for Transact-SQL statements or batches. Applications connecting to SQL Server 2000 and higher should use the [sp\_executesql](http://msdn.microsoft.com/en-us/library/ms188001.aspx) system stored procedure instead of temporary stored procedures. Only the connection that created a local temporary procedure can execute it, and the procedure is automatically deleted when the connection is closed.

Any connection can execute a global temporary stored procedure. A global temporary stored procedure exists until the connection used by the user who created the procedure is closed and any currently executing versions of the procedure by any other connections are completed. Once the connection that was used to create the procedure is closed, no further execution of the global temporary stored procedure is allowed. Only those connections that have already started executing the stored procedure are allowed to complete.

If a stored procedure not prefixed with # or ## is created directly in the **tempdb** database, the stored procedure is automatically deleted when SQL Server is shut down because **tempdb** is re-created every time SQL Server is started. Procedures created directly in **tempdb** exist even after the creating connection is terminated.

|  |
| --- |
| **ms190669.note(en-us,SQL.100).gifNote:** |
| Heavy use of temporary stored procedures can create contention on the system tables in **tempdb** and adversely affect performance. It is recommended that **sp\_executesql** be used instead. **sp\_executesql** does not store data in the system tables and therefore avoids the problem. |

CLR stored procedures cannot be created as temporary stored procedures.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

### A. Using a simple procedure with a complex SELECT

The following stored procedure returns all employees (first and last names supplied), their titles, and their department names from a view. This stored procedure does not use any parameters.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl26other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl26other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ( 'HumanResources.uspGetAllEmployees', 'P' ) IS NOT NULL

DROP PROCEDURE HumanResources.uspGetAllEmployees;

GO

CREATE PROCEDURE HumanResources.uspGetAllEmployees

AS

SET NOCOUNT ON;

SELECT LastName, FirstName, JobTitle, Department

FROM HumanResources.vEmployeeDepartment;

GO

The uspGetEmployees stored procedure can be executed in these ways:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl27other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl27other');)

EXECUTE HumanResources.uspGetAllEmployees;

GO

-- Or

EXEC HumanResources.uspGetAllEmployees;

GO

-- Or, if this procedure is the first statement within a batch:

HumanResources.uspGetAllEmployees;

### B. Using a simple procedure with parameters

The following stored procedure returns only the specified employee (first and last name supplied), her title, and her department name from a view. This stored procedure accepts exact matches for the parameters passed.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl28other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl28other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ( 'HumanResources.uspGetEmployees', 'P' ) IS NOT NULL

DROP PROCEDURE HumanResources.uspGetEmployees;

GO

CREATE PROCEDURE HumanResources.uspGetEmployees

@LastName nvarchar(50),

@FirstName nvarchar(50)

AS

SET NOCOUNT ON;

SELECT FirstName, LastName, JobTitle, Department

FROM HumanResources.vEmployeeDepartment

WHERE FirstName = @FirstName AND LastName = @LastName;

GO

The uspGetEmployees stored procedure can be executed in the following ways:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl29other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl29other');)

EXECUTE HumanResources.uspGetEmployees N'Ackerman', N'Pilar';

-- Or

EXEC HumanResources.uspGetEmployees @LastName = N'Ackerman', @FirstName = N'Pilar';

GO

-- Or

EXECUTE HumanResources.uspGetEmployees @FirstName = N'Pilar', @LastName = N'Ackerman';

GO

-- Or, if this procedure is the first statement within a batch:

HumanResources.uspGetEmployees N'Ackerman', N'Pilar';

### C. Using a simple procedure with wildcard parameters

The following stored procedure returns only the specified employees (first and last names supplied), their titles, and their departments from a view. This stored procedure pattern matches the parameters passed or, if not supplied, uses the preset default (last names that start with the letter D).

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl30other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl30other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ( 'HumanResources.uspGetEmployees2', 'P' ) IS NOT NULL

DROP PROCEDURE HumanResources.uspGetEmployees2;

GO

CREATE PROCEDURE HumanResources.uspGetEmployees2

@LastName nvarchar(50) = N'D%',

@FirstName nvarchar(50) = N'%'

AS

SET NOCOUNT ON;

SELECT FirstName, LastName, JobTitle, Department

FROM HumanResources.vEmployeeDepartment

WHERE FirstName LIKE @FirstName AND LastName LIKE @LastName;

GO

The uspGetEmployees2 stored procedure can be executed in many combinations. Only a few combinations are shown here:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl31other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl31other');)

EXECUTE HumanResources.uspGetEmployees2;

-- Or

EXECUTE HumanResources.uspGetEmployees2 N'Wi%';

-- Or

EXECUTE HumanResources.uspGetEmployees2 @FirstName = N'%';

-- Or

EXECUTE HumanResources.uspGetEmployees2 N'[CK]ars[OE]n';

-- Or

EXECUTE HumanResources.uspGetEmployees2 N'Hesse', N'Stefen';

-- Or

EXECUTE HumanResources.uspGetEmployees2 N'H%', N'S%';

### D. Using OUTPUT parameters

The following example creates the uspGetList stored procedure, which returns a list of products that have prices that do not exceed a specified amount. The example shows using multiple SELECT statements and multiple OUTPUT parameters. OUTPUT parameters allow an external procedure, a batch, or more than one Transact-SQL statement to access a value set during the procedure execution.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl32other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl32other');)

USE AdventureWorks;

GO

IF OBJECT\_ID ( 'Production.uspGetList', 'P' ) IS NOT NULL

DROP PROCEDURE Production.uspGetList;

GO

CREATE PROCEDURE Production.uspGetList @Product varchar(40)

, @MaxPrice money

, @ComparePrice money OUTPUT

, @ListPrice money OUT

AS

SET NOCOUNT ON;

SELECT p.[Name] AS Product, p.ListPrice AS 'List Price'

FROM Production.Product AS p

JOIN Production.ProductSubcategory AS s

ON p.ProductSubcategoryID = s.ProductSubcategoryID

WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;

-- Populate the output variable @ListPprice.

SET @ListPrice = (SELECT MAX(p.ListPrice)

FROM Production.Product AS p

JOIN Production.ProductSubcategory AS s

ON p.ProductSubcategoryID = s.ProductSubcategoryID

WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice);

-- Populate the output variable @compareprice.

SET @ComparePrice = @MaxPrice;

GO

Execute uspGetList to return a list of Adventure Works products (Bikes) that cost less than $700. The OUTPUT parameters **@Cost** and **@ComparePrices** are used with control-of-flow language to return a message in the **Messages** window.

|  |
| --- |
| **ms190669.note(en-us,SQL.100).gifNote:** |
| The OUTPUT variable must be defined during the procedure creation as well as during the use of the variable. The parameter name and variable name do not have to match; however, the data type and parameter positioning must match (unless **@ListPrice** **=** variable is used). |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl34other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl34other');)

DECLARE @ComparePrice money, @Cost money

EXECUTE Production.uspGetList '%Bikes%', 700,

@ComparePrice OUT,

@Cost OUTPUT

IF @Cost <= @ComparePrice

BEGIN

PRINT 'These products can be purchased for less than

$'+RTRIM(CAST(@ComparePrice AS varchar(20)))+'.'

END

ELSE

PRINT 'The prices for all products in this category exceed

$'+ RTRIM(CAST(@ComparePrice AS varchar(20)))+'.'

Here is the partial result set:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl35other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl35other');)

Product List Price

-------------------------------------------------- ------------------

Road-750 Black, 58 539.99

Mountain-500 Silver, 40 564.99

Mountain-500 Silver, 42 564.99

...

Road-750 Black, 48 539.99

Road-750 Black, 52 539.99

(14 row(s) affected)

These items can be purchased for less than $700.00.

**How to: Create a Stored Procedure (SQL Server Management Studio)**

This topic describes how to create a Transact-SQL stored procedure by using Object Explorer in SQL Server Management Studio and provides an example that creates a simple stored procedure in the **AdventureWorks** database.

http://i.msdn.microsoft.com/Global/Images/clear.gif To create a stored procedure

1. In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.
2. Expand **Databases**, expand the database in which the stored procedure belongs, and then expand **Programmability**.
3. Right-click **Stored Procedures**, and then click **New Stored Procedure**.
4. On the **Query** menu, click **Specify Values for Template Parameters**.
5. In the **Specify Values for Template Parameters** dialog box, the **Value** column contains suggested values for the parameters. Accept the values or replace them with new values, and then click **OK**.
6. In the query editor, replace the SELECT statement with the statements for your procedure.
7. To test the syntax, on the **Query** menu, click **Parse**.
8. To create the stored procedure, on the **Query** menu, click **Execute**.
9. To save the script, on the **File** menu, click **Save**. Accept the file name or replace it with a new name, and then click **Save**.

|  |
| --- |
| **ms345415.security(en-us,SQL.100).gifSecurity Note:** |
| Validate all user input. Do not concatenate user input before you validate it. Never execute a command constructed from unvalidated user input. For more information, see [SQL Injection](http://msdn.microsoft.com/en-us/library/ms161953.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif To create a stored procedure example

1. In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.
2. Expand **Databases**, expand the **AdventureWorks** database, and then expand **Programmability**.
3. Right-click **Stored Procedures**, and then click **New Stored Procedure**.
4. On the **Query** menu, click **Specify Values for Template Parameters**.
5. In the **Specify Values for Template Parameters** dialog box, enter the following values for the parameters shown.

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Author | *Your name* |
| Create Date | *Today's date* |
| Description | Returns employee data. |
| Procedure\_name | HumanResources.uspGetEmployees |
| @Param1 | @LastName |
| @Datatype\_For\_Param1 | **nvarchar**(50) |
| Default\_Value\_For\_Param1 | NULL |
| @Param2 | @FirstName |
| @Datatype\_For\_Param2 | **nvarchar**(50) |
| Default\_Value\_For\_Param2 | NULL |

1. Click **OK**.
2. In the query editor, replace the SELECT statement with the following statement:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl11other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl11other');)

SELECT FirstName, LastName, JobTitle, Department

FROM HumanResources.vEmployeeDepartment

WHERE FirstName = @FirstName AND LastName = @LastName;

1. To test the syntax, on the **Query** menu, click **Parse**. If an error message is returned, compare the statements with the information above and correct as needed.
2. To create the stored procedure, on the **Query** menu, click **Execute**.
3. To save the script, on the **File** menu, click **Save**. Enter a new file name, and then click **Save**.
4. To run the stored procedure, on the toolbar, click **New Query**.
5. In the query window, enter the following statements:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl12other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl12other');)

USE AdventureWorks;

GO

EXECUTE HumanResources.uspGetEmployees @FirstName = N'Diane', @LastName = N'Margheim';

GO

1. On the **Query** menu, click **Execute**.

**Creating CLR Stored Procedures**

In SQL Server 2008, you can create a database object inside an instance of SQL Server that is programmed in an assembly created in the Microsoft .NET Framework common language runtime (CLR). Database objects that can leverage the rich programming model provided by the CLR include triggers, stored procedures, functions, aggregate functions, and types.

Creating a CLR stored procedure in SQL Server involves the following steps:

* Define the stored procedure as a static method of a class in a language supported by the .NET Framework. For more information about how to program CLR stored procedures, see [CLR Stored Procedures](http://msdn.microsoft.com/en-us/library/ms131094.aspx). Then, compile the class to build an assembly in the .NET Framework by using the appropriate language compiler.
* Register the assembly in SQL Server by using the CREATE ASSEMBLY statement. For more information about how to work with assemblies in SQL Server, see [Assemblies](http://msdn.microsoft.com/en-us/library/ms186221.aspx).
* Create the stored procedure that references the registered assembly by using the CREATE PROCEDURE statement.

|  |
| --- |
| **ms190790.note(en-us,SQL.100).gifNote:** |
| Deploying a SQL Server Project in Microsoft Visual Studio registers an assembly in the database that was specified for the project. Deploying the project also creates CLR stored procedures in the database for all methods that are annotated with the **SqlProcedure** attribute. For more information, see [Deploying CLR Database Objects](http://msdn.microsoft.com/en-us/library/ms345099.aspx). |
| **ms190790.note(en-us,SQL.100).gifNote:** |
| The ability of SQL Server to execute CLR code is off by default. You can create, alter, and drop database objects that reference managed code modules, but these references will not execute in SQL Server unless the [clr enabled Option](http://msdn.microsoft.com/en-us/library/ms175193.aspx) is enabled by using [sp\_configure (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188787.aspx). |

**To create, modify, or drop an assembly**

* [CREATE ASSEMBLY (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189524.aspx)
* [ALTER ASSEMBLY (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms186711.aspx)
* [DROP ASSEMBLY (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms177514.aspx)

**To create a CLR Stored Procedure**

* [CREATE PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187926.aspx)

**Deferred Name Resolution and Compilation**

When a stored procedure is created, the statements in the procedure are parsed for syntactical accuracy. If a syntactical error is encountered in the procedure definition, an error is returned and the stored procedure is not created. If the statements are syntactically correct, the text of the stored procedure is stored in the [sys.sql\_modules](http://msdn.microsoft.com/en-us/library/ms175081.aspx) catalog view.

When a stored procedure is executed for the first time, the query processor reads the text of the stored procedure from the **sys.sql\_modules** catalog view and checks that the names of the objects used by the procedure are present. This process is called deferred name resolution because table objects referenced by the stored procedure need not exist when the stored procedure is created, but only when it is executed.

|  |
| --- |
| **ms190686.note(en-us,SQL.100).gifNote:** |
| Deferred name resolution can only be used when you reference nonexistent table objects. All other objects must exist at the time the stored procedure is created. For example, when you reference an existing table in a stored procedure you cannot list nonexistent columns for that table. |

In the resolution stage, Microsoft SQL Server also performs other validation activities (for example, checking the compatibility of a column data type with variables). If the objects referenced by the stored procedure are missing when the stored procedure is executed, the stored procedure stops executing when it gets to the statement that references the missing object. In this case, or if other errors are found in the resolution stage, an error is returned.

|  |
| --- |
| **ms190686.note(en-us,SQL.100).gifNote:** |
| If an object referenced by a stored procedure is deleted or renamed, then an error is returned when the stored procedure is executed. However, if an object referenced in a stored procedure is replaced with an object of the same name, the stored procedure executes without having to be re-created. For example, if stored procedure **Proc1** references table **Test1**, and **Test1** is deleted and a different table called **Test1** is created, **Proc1** references the new table. The stored procedure does not have to be recreated. |

If procedure execution successfully passes the resolution stage, the Microsoft SQL Server query optimizer analyzes the Transact-SQL statements in the stored procedure and creates an execution plan. The execution plan describes the fastest method of executing the stored procedure, based on information such as:

* The amount of data in the tables.
* The nature and presence of indexes on the tables and the distribution of data in the indexed columns.
* The comparison operators and comparison values used in WHERE clause conditions.
* The presence of joins and UNION, GROUP BY, and ORDER BY keywords.

After the query optimizer has analyzed these factors in the stored procedure, it places the execution plan in memory. The process of analyzing the stored procedure and creating an execution plan is called compilation. The optimized in-memory execution plan is used to execute the query. The execution plan stays in memory until SQL Server is restarted, or until space is needed for storage of another object.

When the stored procedure is subsequently executed, SQL Server reuses the existing execution plan if it is still in memory. If the execution plan is no longer in memory, a new execution plan is created.

**Executing Stored Procedures (Database Engine)**

To execute a stored procedure, use the Transact-SQL EXECUTE statement. Alternatively, you can execute a stored procedure without using the EXECUTE keyword if the stored procedure is the first statement in the batch.

http://i.msdn.microsoft.com/Global/Images/clear.gif Executing System Stored Procedures

System stored procedures begin with the characters **sp\_**. They are physically stored in the [Resource database](http://msdn.microsoft.com/en-us/library/ms190940.aspx), but logically appear in the **sys** schema of every system- and user-defined database in the instance of SQL Server. System stored procedures can be executed from any database without having to fully quality the stored procedure name. A nonschema-qualified name is either a one-part name such as **sp\_someproc** or a three-part name such as **somedb..sp\_someproc** with the second part, the schema name, unspecified.

We recommend that you schema-qualify all system stored procedure names with the **sys** schema name to prevent name conflicts. The following example demonstrates the recommended method of executing a system stored procedure.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl11other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl11other');)

EXEC sys.sp\_who;

The following examples demonstrate backward compatible methods of executing system stored procedures.

|  |
| --- |
| **ms189915.note(en-us,SQL.100).gifNote:** |
| The following methods of executing system stored procedures will be removed in a future version of SQL Server. Avoid using these methods in new development work, and plan to modify applications that currently use them. |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl13other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl13other');)

EXEC sp\_who;

EXEC master.dbo.sp\_who;

EXEC mydatabase..sp\_who;

EXEC dbo.sp\_who;

EXEC mydatabase.dbo.sp\_who;

### Database Collation Matching

SQL Server 2008 uses the calling database collation when matching system procedure names. Therefore, you should always use the exact case of system procedure names in your application. For example, this code will fail if executed in the context of a database that has a case-sensitive collation:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl14other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl14other');)

exec SP\_heLP; -- Will fail to resolve because SP\_heLP does not equal sp\_help

Use the [sys.system\_objects](http://msdn.microsoft.com/en-us/library/ms173551.aspx) and [sys.system\_parameters](http://msdn.microsoft.com/en-us/library/ms174367.aspx) catalog views to display the exact system stored procedure names.

http://i.msdn.microsoft.com/Global/Images/clear.gif Executing System Extended Stored Procedures

System extended stored procedures begin with the characters **xp\_**. They are physically stored in the [Resource database](http://msdn.microsoft.com/en-us/library/ms190940.aspx), but logically appear in the **sys** schema of every system- and user-defined database in the instance of SQL Server. The following example demonstrates the recommended method for executing a system extended stored procedure.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl22other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl22other');)

EXEC sys.xp\_subdirs 'c:\';

http://i.msdn.microsoft.com/Global/Images/clear.gif Executing User-defined Stored Procedures

When executing a user-defined stored procedure (either in a batch or inside a module such as a user-defined stored procedure or function), we strongly recommend qualifying the stored procedure name with at least the schema name.

The following example demonstrates the recommended method for executing a user-defined stored procedure.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl30other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl30other');)

USE AdventureWorks;

GO

EXEC dbo.uspGetEmployeeManagers 50;

-Or-

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl31other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl31other');)

EXEC AdventureWorks.dbo.uspGetEmployeeManagers 50;

GO

If a nonqualified user-defined stored procedure is specified, the Database Engine searches for the procedure in the following order:

* The **sys** schema of the current database.
* The caller's default schema if executed in a batch or in dynamic SQL. Or, if the nonqualified procedure name appears inside the body of another procedure definition, the schema containing this other procedure is searched next. For more information about default schemas, see [User-Schema Separation](http://msdn.microsoft.com/en-us/library/ms190387.aspx).
* The **dbo** schema in the current database.

|  |
| --- |
| **ms189915.note(en-us,SQL.100).gifImportant:** |
| If any user-created stored procedure has the same name as a system stored procedure, the user-created stored procedure will never be executed if you use a nonschema-qualified name reference. For more information, see [Creating Stored Procedures (Database Engine)](http://msdn.microsoft.com/en-us/library/ms190669.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif Specifying Parameters

Parameter values can be supplied if a stored procedure is written to accept them.

The supplied value must be a constant or a variable; you cannot specify a function name as a parameter value. Variables can be user-defined or system variables such as @@spid.

The following examples demonstrate passing parameter values to the stored procedure uspGetWhereUsedProductID. The procedure expects values for two input parameters: a product ID and a date. The examples show how to pass parameters as constants and variables and also how to use a variable to pass the value of a function.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl40other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl40other');)

USE AdventureWorks;

GO

-- Passing values as constants.

EXEC dbo.uspGetWhereUsedProductID 819, '20050225';

GO

-- Passing values as variables.

DECLARE @ProductID int, @CheckDate datetime;

SET @ProductID = 819;

SET @CheckDate = '20050225';

EXEC dbo.uspGetWhereUsedProductID @ProductID, @CheckDate;

GO

-- Try to use a function as a parameter value.

-- This produces an error message.

EXEC dbo.uspGetWhereUsedProductID 819, GETDATE();

GO

-- Passing the function value as a variable.

DECLARE @CheckDate datetime;

SET @CheckDate = GETDATE();

EXEC dbo.uspGetWhereUsedProductID 819, @CheckDate;

GO

If you want to specify the parameters in a different order than the order they are defined in the stored procedure, you must name them. For more information, see [Specifying a Parameter Name](http://msdn.microsoft.com/en-us/library/ms177436.aspx).

To specify that a parameter should return a value to the calling program, use the OUTPUT keyword. For more information, see [Specifying the Direction of a Parameter](http://msdn.microsoft.com/en-us/library/ms191422.aspx).

### Specifying the Order of Parameters

If you supply parameters in the form **@parameter =** value, you can supply them in any order. You can also omit parameters for which defaults have been supplied. If you supply one parameter in the form **@parameter =** value, you must supply all subsequent parameters this way. If you do not supply parameters in the form **@parameter =** value, you must supply them in the order given in the CREATE PROCEDURE statement.

When executing a stored procedure, the server rejects any parameters that were not included with the parameter list during procedure creation. Any parameter passed by reference (explicitly passing the parameter name) is not accepted if the parameter name does not match.

### Using Default Values in Parameters

Although you can omit parameters for which defaults have been supplied, you can only truncate the list of parameters. For example, if a stored procedure has five parameters, you can omit both the fourth and the fifth parameters, but you cannot skip the fourth and still include the fifth unless you supply parameters in the form **@parameter =** value.

The default value of a parameter, if defined for the parameter in the stored procedure, is used when:

* No value for the parameter is specified when the stored procedure is executed.
* The DEFAULT keyword is specified as the value for the parameter.

**Extended Stored Procedures**

|  |
| --- |
| **ms175200.note(en-us,SQL.100).gifImportant:** |
| This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [CLR Integration](http://msdn.microsoft.com/en-us/library/ms131089.aspx) instead. |

Extended stored procedures let you create your own external routines in a programming language such as C. The extended stored procedures appear to users as regular stored procedures and are executed in the same way. Parameters can be passed to extended stored procedures, and extended stored procedures can return results and return status.

Extended stored procedures are DLLs that an instance of SQL Server can dynamically load and run. Extended stored procedures run directly in the address space of an instance of SQL Server and are programmed by using the SQL Server Extended Stored Procedure API.

|  |
| --- |
| **ms175200.note(en-us,SQL.100).gifNote:** |
| Extended stored procedures should not be used to instantiate the Microsoft .NET Framework common language runtime and execute managed code. This scenario will not be supported in future versions of SQL Server. CLR Integration provides a more robust and secure alternative to writing extended stored procedures. |

After an extended stored procedure has been written, members of the **sysadmin** fixed server role can register the extended stored procedure with the instance of SQL Server, and then grant permission to other users to execute the procedure. Extended stored procedures can be added only to the **master** database.

|  |
| --- |
| **ms175200.note(en-us,SQL.100).gifNote:** |
| Extended stored procedures may produce memory leaks or other problems that reduce the performance and reliability of the server. You should consider storing extended stored procedures in an instance of SQL Server that is separate from the instance that contains the referenced data. You should also consider using distributed queries to access the database. For more information, see [Distributed Queries](http://msdn.microsoft.com/en-us/library/ms188721.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif Security

Before adding extended stored procedures to the server and granting execute permissions to other users, the system administrator should thoroughly review each extended stored procedure to make sure that it does not contain harmful or malicious code.

Additional security guidelines include the following:

* Validate all user input.
* Do not concatenate user input before validating it.
* Never execute a command constructed from unvalidated user input. For more information, see [SQL Injection](http://msdn.microsoft.com/en-us/library/ms161953.aspx).

**Modifying Stored Procedures**

If you need to change the statements or parameters in a stored procedure, you can either delete and re-create the stored procedure or alter the stored procedure in a single step. When you delete and re-create a stored procedure, all permissions associated with the stored procedure are lost. When you alter the stored procedure, the procedure or parameter definition is changed but the permissions defined for the stored procedure are retained and any dependent stored procedures or triggers are not affected.

A stored procedure can also be modified to encrypt the definition or cause the procedure to be recompiled each time it is executed.

|  |
| --- |
| **ms187001.note(en-us,SQL.100).gifNote:** |
| Changing the name or definition of a stored procedure can cause any dependent objects to fail when executed if those dependent objects are not also updated to reflect the changes made to the stored procedure. |

**To modify a stored procedure**

* [ALTER PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189762.aspx)
* [How to: Modify a Stored Procedure (SQL Server Management Studio)](http://msdn.microsoft.com/en-us/library/ms345356.aspx)

**ALTER PROCEDURE (Transact-SQL)**

Modifies a previously created procedure that was created by executing the CREATE PROCEDURE statement. ALTER PROCEDURE does not change permissions and does not affect any dependent stored procedures or triggers. However, the current session settings for QUOTED\_IDENTIFIER and ANSI\_NULLS are included in the stored procedure when it is modified. If the settings are different from those in effect when stored procedure was originally created, the behavior of the stored procedure may change.

 Syntax

ALTER { PROC | PROCEDURE } [schema\_name.] procedure\_name [ ; number ]

    [ { @parameter [ type\_schema\_name. ] data\_type }

    [ VARYING ] [ = default ] [ [ OUT [ PUT ]

    ] [ ,...n ]

[ WITH <procedure\_option> [ ,...n ] ]

[ FOR REPLICATION ]

AS

     { <sql\_statement> [ ...n ] | <method\_specifier> }

<procedure\_option> ::=

    [ ENCRYPTION ]

    [ RECOMPILE ]

    [ EXECUTE\_AS\_Clause ]

<sql\_statement> ::=

{ [ BEGIN ] statements [ END ] }

<method\_specifier> ::=

EXTERNAL NAME

assembly\_name.class\_name.method\_name

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

schema\_name

Is the name of the schema to which the procedure belongs.

procedure\_name

Is the name of the procedure to change. Procedure names must comply with the rules for [identifiers](http://msdn.microsoft.com/en-us/library/ms175874.aspx).

**;** number

Is an existing optional integer that is used to group procedures of the same name so that they can be dropped together by using one DROP PROCEDURE statement.

|  |
| --- |
| **ms189762.note(en-us,SQL.100).gifNote:** |
| This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. |

**@** parameter

Is a parameter in the procedure. Up to 2,100 parameters can be specified.

[ type\_schema\_name**.** ] data\_type

Is the data type of the parameter and the schema it belongs to.

For information about data type restrictions, see [CREATE PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187926.aspx).

VARYING

Specifies the result set supported as an output parameter. This parameter is constructed dynamically by the stored procedure and its contents can vary. Applies only to cursor parameters.

default

Is a default value for the parameter.

OUTPUT

Indicates that the parameter is a return parameter.

RECOMPILE

Indicates that the SQL Server 2005 Database Engine does not cache a plan for this procedure and the procedure is recompiled at run time.

ENCRYPTION

Indicates that the Database Engine will convert the original text of the ALTER PROCEDURE statement to an obfuscated format. The output of the obfuscation is not directly visible in any of the catalog views in SQL Server 2005. Users that have no access to system tables or database files cannot retrieve the obfuscated text. However, the text will be available to privileged users that can either access system tables over the [DAC port](http://msdn.microsoft.com/en-us/library/ms189595.aspx) or directly access database files. Also, users that can attach a debugger to the server process can retrieve the original procedure from memory at runtime. For more information about accessing system metadata, see [Metadata Visibility Configuration](http://msdn.microsoft.com/en-us/library/ms187113.aspx).

Procedures created with this option cannot be published as part of SQL Server replication.

This option cannot be specified for common language runtime (CLR) stored procedures.

|  |
| --- |
| **ms189762.note(en-us,SQL.100).gifNote:** |
| During an upgrade, the Database Engine uses the obfuscated comments stored in **sys.sql\_modules** to re-create procedures. |

EXECUTE AS

Specifies the security context under which to execute the stored procedure after it is accessed.

For more information, see [EXECUTE AS Clause (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188354.aspx).

FOR REPLICATION

Specifies that stored procedures that are created for replication cannot be executed on the Subscriber. A stored procedure created with the FOR REPLICATION option is used as a stored procedure filter and only executed during replication. Parameters cannot be declared if FOR REPLICATION is specified. The RECOMPILE option is ignored for procedures created with FOR REPLICATION.

AS

Are the actions the procedure is to take.

<sql\_statement>

Is any number and type of Transact-SQL statements to be included in the procedure. Some limitations do apply. For more information, see "<sql\_statement> Limitations" in [CREATE PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187926.aspx).

EXTERNAL NAME assembly\_name**.**class\_name**.**method\_name

Specifies the method of a Microsoft .NET Framework assembly for a CLR stored procedure to reference. class\_name must be a valid SQL Server identifier and must exist as a class in the assembly. If the class has a namespace-qualified name uses a period (**.**) to separate namespace parts, the class name must be delimited by using brackets (**[** **]**) or quotation marks (**"** **"**). The specified method must be a static method of the class.

|  |
| --- |
| **ms189762.note(en-us,SQL.100).gifNote:** |
| By default, SQL Server cannot execute CLR code. You can create, modify, and drop database objects that reference common language runtime modules; however, you cannot execute these references in SQL Server until you enable the [clr enabled option](http://msdn.microsoft.com/en-us/library/ms175193.aspx). To enable the option, use [sp\_configure](http://msdn.microsoft.com/en-us/library/ms188787.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

Transact-SQL stored procedures cannot be modified to be CLR stored procedures and vice versa.

For more information, see the Remarks section in [CREATE PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187926.aspx).

|  |
| --- |
| **ms189762.note(en-us,SQL.100).gifNote:** |
| If a previous procedure definition was created using WITH ENCRYPTION or WITH RECOMPILE, these options are enabled only if they are included in ALTER PROCEDURE. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

Requires ALTER permission on the procedure.

### Examples

The following example creates the uspVendorAllInfo stored procedure. This procedure returns the names of all the vendors that supply Adventure Works Cycles, the products they supply, their credit ratings, and their availability. After this procedure is created, it is then modified to return a different result set.

USE AdventureWorks;

GO

IF OBJECT\_ID ( 'Purchasing.uspVendorAllInfo', 'P' ) IS NOT NULL

DROP PROCEDURE Purchasing.uspVendorAllInfo;

GO

CREATE PROCEDURE Purchasing.uspVendorAllInfo

WITH EXECUTE AS CALLER

AS

SET NOCOUNT ON;

SELECT v.Name AS Vendor, p.Name AS 'Product name',

v.CreditRating AS 'Credit Rating',

v.ActiveFlag AS Availability

FROM Purchasing.Vendor v

INNER JOIN Purchasing.ProductVendor pv

ON v.VendorID = pv.VendorID

INNER JOIN Production.Product p

ON pv.ProductID = p.ProductID

ORDER BY v.Name ASC;

GO

The following example alters the uspVendorAllInfo stored procedure (without the EXECUTE AS option) to return only those vendors that supply the specified product. The LEFT and CASE functions customize the appearance of the result set.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl43other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl43other');)

ALTER PROCEDURE Purchasing.uspVendorAllInfo

@Product varchar(25)

AS

    SELECT LEFT(v.Name, 25) AS Vendor, LEFT(p.Name, 25) AS 'Product name',

    'Credit rating' = CASE v.CreditRating

        WHEN 1 THEN 'Superior'

        WHEN 2 THEN 'Excellent'

        WHEN 3 THEN 'Above average'

        WHEN 4 THEN 'Average'

        WHEN 5 THEN 'Below average'

        ELSE 'No rating'

        END

    , Availability = CASE v.ActiveFlag

        WHEN 1 THEN 'Yes'

        ELSE 'No'

        END

    FROM Purchasing.Vendor AS v

    INNER JOIN Purchasing.ProductVendor AS pv

      ON v.VendorID = pv.VendorID

    INNER JOIN Production.Product AS p

      ON pv.ProductID = p.ProductID

    WHERE p.Name LIKE @Product

    ORDER BY v.Name ASC;

GO

EXEC Purchasing.uspVendorAllInfo N'LL Crankarm';

GO

Here is the result set.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl44other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl44other');)

Vendor Product name Credit rating Availability

-------------------- ------------------- ------------- ------------

Proseware, Inc. LL Crankarm Average No

Vision Cycles, Inc. LL Crankarm Superior Yes

(2 row(s) affected)

**How to: Modify a Stored Procedure (SQL Server Management Studio)**

This topic describes how to modify a Transact-SQL stored procedure by using Object Explorer in SQL Server Management Studio. When you change the statements or parameters in a stored procedure by using this method, the permissions that have been defined for the stored procedure are retained, and any dependent stored procedures or triggers are not affected.

|  |
| --- |
| **ms345356.note(en-us,SQL.100).gifImportant:** |
| Changing the name or definition of a stored procedure can cause dependent objects to fail if the dependent objects are not updated to reflect the changes that have been made to the stored procedure. For more information, see [How to: View the Dependencies of a Stored Procedure (SQL Server Management Studio)](http://msdn.microsoft.com/en-us/library/ms345404.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif To modify a stored procedure

1. In Object Explorer, connect to an instance of Database Engine and then expand that instance.
2. Expand **Databases**, expand the database in which the stored procedure belongs, and then expand **Programmability**.
3. Expand **Stored Procedures**, right-click the procedure to modify, and then click **Modify**.
4. Modify the text of the stored procedure.
5. To test the syntax, on the **Query** menu, click **Parse**.
6. To modify the stored procedure, on the **Query** menu, click **Execute**.
7. To save the script, on the **File** menu, click **Save As**. Accept the file name or replace it with a new name, and then click **Save**.

**Recompiling Stored Procedures**

As a database is changed by such actions as adding indexes or changing data in indexed columns, the original query plans used to access its tables should be optimized again by recompiling them. This optimization happens automatically the first time a stored procedure is run after Microsoft SQL Server is restarted. It also occurs if an underlying table used by the stored procedure changes. But if a new index is added from which the stored procedure might benefit, optimization does not happen until the next time the stored procedure is run after Microsoft SQL Server is restarted. In this situation, it can be useful to force the stored procedure to recompile the next time it executes

Another reason to force a stored procedure to recompile is to counteract, when necessary, the "parameter sniffing" behavior of stored procedure compilation. When SQL Server executes stored procedures, any parameter values used by the procedure when it compiles are included as part of generating the query plan. If these values represent the typical ones with which the procedure is called subsequently, then the stored procedure benefits from the query plan each time it compiles and executes. If not, performance may suffer.

|  |
| --- |
| **ms190439.note(en-us,SQL.100).gifNote:** |
| SQL Server 2008 features statement-level recompilation of stored procedures. When SQL Server 2008 recompiles stored procedures, only the statement that caused the recompilation is compiled, rather than the entire procedure. As a result, SQL Server uses the parameter values as they exist in the recompiled statement when regenerating a query plan. These values may differ from those that were originally passed into the procedure. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Forcing a Stored Procedure to Recompile

SQL Server provides three ways to force a stored procedure to recompile:

* The **sp\_recompile** system stored procedure forces a recompile of a stored procedure the next time it is run.
* Creating a stored procedure that specifies the WITH RECOMPILE option in its definition indicates that SQL Server does not cache a plan for this stored procedure; the stored procedure is recompiled each time it is executed. Use the WITH RECOMPILE option when stored procedures take parameters whose values differ widely between executions of the stored procedure, resulting in different execution plans to be created each time. Use of this option is uncommon and causes the stored procedure to execute more slowly, because the stored procedure must be recompiled each time it is executed.  
  If you only want individual queries inside the stored procedure to be recompiled, rather than the entire stored procedure, specify the RECOMPILE query hint inside each query you want recompiled. This behavior mimics SQL Server's statement-level recompilation behavior noted above, but in addition to using the stored procedure's current parameter values, the RECOMPILE query hint also uses the values of any local variables inside the stored procedure when compiling the statement. Use this option when atypical or temporary values are used in only a subset of queries belonging to the stored procedure. For more information, see [Query Hint (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms181714.aspx).
* You can force the stored procedure to be recompiled by specifying the WITH RECOMPILE option when you execute the stored procedure. Use this option only if the parameter you are supplying is atypical or if the data has significantly changed since the stored procedure was created.

|  |
| --- |
| **ms190439.note(en-us,SQL.100).gifNote:** |
| If an object referenced by a stored procedure is deleted or renamed, an error is returned when the stored procedure is executed. If, however, an object referenced in a stored procedure is replaced with an object of the same name, the stored procedure executes without having to be re-created. |

**To recompile a stored procedure next time it is run**

* [sp\_recompile (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms181647.aspx)

**sp\_recompile (Transact-SQL)**

Causes stored procedures and triggers to be recompiled the next time they are run.

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

sp\_recompile [ @objname = ] 'object'

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

[ **@objname =** ] **'***object***'**

Is the qualified or unqualified name of a stored procedure, trigger, table, or view in the current database. *object* is **nvarchar(776)**, with no default. If *object* is the name of a stored procedure or trigger, the stored procedure or trigger will be recompiled the next time that it is run. If *object* is the name of a table or view, all the stored procedures that reference the table or view will be recompiled the next time they are run.

http://i.msdn.microsoft.com/Global/Images/clear.gif Return Code Values

0 (success) or a nonzero number (failure)

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

**sp\_recompile** looks for an object in the current database only.

The queries used by stored procedures and triggers are optimized only when they are compiled. As indexes or other changes that affect statistics are made to the database, compiled stored procedures and triggers may lose efficiency. By recompiling stored procedures and triggers that act on a table, you can reoptimize the queries.

|  |
| --- |
| **ms181647.note(en-us,SQL.100).gifNote:** |
| SQL Server automatically recompiles stored procedures and triggers when it is advantageous to do this. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

Requires ALTER permission on the specified object.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

The following example causes stored procedures that act on the Customer table to be recompiled the next time they are run.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl20other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl20other');)

USE AdventureWorks;

GO

EXEC sp\_recompile N'Sales.Customer';

GO

**Stored Procedures**

Several system stored procedures and catalog views provide information about stored procedures. Using these, you can:

* See the definition of the stored procedure. That is, the Transact-SQL statements used to create a stored procedure. This can be useful if you do not have the Transact-SQL script files used to create the stored procedure.
* Get information about a stored procedure such as its schema, when it was created, and its parameters.
* List the objects used by the specified stored procedure, and the procedures that use the specified stored procedure. This information can be used to identify the procedures affected by the changing or removal of an object in the database.

**To view the definition of a stored procedure**

* [sys.sql\_modules (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms175081.aspx)
* [OBJECT\_DEFINITION (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms176090.aspx)
* [sp\_helptext (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms176112.aspx)

**To view information about a stored procedure**

* [sys.objects (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms190324.aspx)
* [sys.procedures (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188737.aspx)
* [sys.parameters (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms176074.aspx)
* [sys.numbered\_procedures (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms179865.aspx)
* [sys.numbered\_procedure\_parameters (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187805.aspx)
* [sp\_help (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms187335.aspx)

**To view the dependencies of a stored procedure**

* [sys.sql\_expression\_dependencies (Transact-SQL)](http://msdn.microsoft.com/en-us/library/bb677315.aspx)
* [sys.dm\_sql\_referenced\_entities (Transact-SQL)](http://msdn.microsoft.com/en-us/library/bb677185.aspx)
* [sys.dm\_sql\_referencing\_entities (Transact-SQL)](http://msdn.microsoft.com/en-us/library/bb630351.aspx)

**To view information about an extended stored procedure**

* [sp\_helpextendedproc (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms186854.aspx)

**sp\_helpextendedproc (Transact-SQL)**

Reports the currently defined extended stored procedures and the name of the dynamic-link library (DLL) to which the procedure (function) belongs.

|  |
| --- |
| **ms186854.note(en-us,SQL.100).gifNote:** |
| This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [CLR Integration](http://msdn.microsoft.com/en-us/library/ms131089.aspx) instead. |

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

sp\_helpextendedproc [ [@funcname = ] 'procedure' ]

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

[ **@funcname =**] **'**procedure**'**

Is the name of the extended stored procedure for which information is reported. procedure is **sysname**, with a default of NULL.

http://i.msdn.microsoft.com/Global/Images/clear.gif Return Code Values

0 (success) or 1 (failure)

http://i.msdn.microsoft.com/Global/Images/clear.gif Result Sets

|  |  |  |
| --- | --- | --- |
| **Column name** | **Data Type** | **Description** |
| **name** | **sysname** | Name of the extended stored procedure. |
| **dll** | **nvarchar(255)** | Name of the DLL. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

When procedure is specified, **sp\_helpextendedproc** reports on the specified extended stored procedure. When this parameter is not supplied, **sp\_helpextendedproc** returns all extended stored procedure names and the DLL names to which each extended stored procedure belongs.

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

Permission to execute **sp\_helpextendedproc** is granted to **public**.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

### A. Reporting help on all extended stored procedures

The following example reports on all extended stored procedures.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl23other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl23other');)

USE master;

GO

EXEC sp\_helpextendedproc;

GO

### B. Reporting help on a single extended stored procedure

The following example reports on the xp\_cmdshell extended stored procedure.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl24other');)

USE master;

GO

EXEC sp\_helpextendedproc xp\_cmdshell;

GO

**Deleting Stored Procedures**

You can delete a stored procedure when you no longer need it. If a stored procedure that has been deleted is called by another stored procedure, Microsoft SQL Server displays an error message when the calling procedure is executed. However, if a new stored procedure of the same name and the same parameters is defined to replace the one that was deleted, other procedures that reference it will still execute successfully. For example, if stored procedure **proc1** references stored procedure **proc2**, and **proc2** is deleted and a different stored procedure called **proc2** is created, **proc1** now references the new stored procedure. **proc1** does not have to be recreated.

**To delete a stored procedure**

* [DROP PROCEDURE (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms174969.aspx)
* [How to: Delete a Stored Procedure (SQL Server Management Studio)](http://msdn.microsoft.com/en-us/library/ms345358.aspx)

**To delete an extended stored procedure**

* [sp\_dropextendedproc (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms189506.aspx)

**sp\_dropextendedproc (Transact-SQL)**

Drops an extended stored procedure.

|  |
| --- |
| **ms189506.note(en-us,SQL.100).gifNote:** |
| This feature will be removed in a future version of Microsoft SQL Server. Avoid using this feature in new development work, and plan to modify applications that currently use this feature. Use [CLR Integration](http://msdn.microsoft.com/en-us/library/ms131089.aspx) instead. |

Topic link icon[Transact-SQL Syntax Conventions](http://msdn.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

sp\_dropextendedproc [ @functname = ] 'procedure'

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

[ **@functname =**] **'***procedure***'**

Is the name of the extended stored procedure to drop. *procedure* is **nvarchar(517)**, with no default.

http://i.msdn.microsoft.com/Global/Images/clear.gif Return Code Values

0 (success) or 1 (failure)

http://i.msdn.microsoft.com/Global/Images/clear.gif Result Sets

None

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

Executing **sp\_dropextendedproc** drops the user-defined extended stored procedure name from the [sys.objects](http://msdn.microsoft.com/en-us/library/ms190324.aspx) catalog view and removes the entry from the [sys.extended\_procedures](http://msdn.microsoft.com/en-us/library/ms176079.aspx) catalog view. This stored procedure can be run only in the **master** database.

In SQL Server 2008 and SQL Server 2005, **sp\_dropextendedproc** does not drop system extended stored procedures. Instead, the system administrator should deny EXECUTE permission on the extended stored procedure to the **public** role. In SQL Server 2000, **sp\_dropextendedproc** could be used to drop any extended stored procedure.

**sp\_dropextendedproc** cannot be executed inside a transaction.

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

Only members of the **sysadmin** fixed server role can execute **sp\_dropextendedproc**.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

The following example drops the xp\_hello extended stored procedure.

|  |
| --- |
| **ms189506.note(en-us,SQL.100).gifNote:** |
| This extended stored procedure must already exist, or the example will return an error message. |

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl28other');)](javascript:CopyCode('ctl00_LibFrame_MainContent_ctl28other');)

USE master;

GO

EXEC sp\_dropextendedproc 'xp\_hello';