Module 15: Executing Stored Procedures

Contents:

Module Overview

In addition to writing stand-alone SELECT statements to return data from Microsoft® SQL Server®, you may need to execute T-SQL procedures created by an administrator or developer and stored in a database. This module will show you how to execute stored procedures, including how to pass parameters into procedures written to accept them. This module will also show you how basic stored procedures are created, providing a better understanding of what happens on the server when you execute one. Finally, this module will show you how to generate dynamic SQL statements, which is often a requirement in development environments where stored procedures are not being used.

Objectives

After completing this module, you will be able to:

- Return results by executing stored procedures.
- Pass parameters to procedures.
- Create simple stored procedures that encapsulate a SELECT statement.
- Construct and execute dynamic SQL with EXEC and sp_executesql.

Lesson 1: Querying Data with Stored Procedures

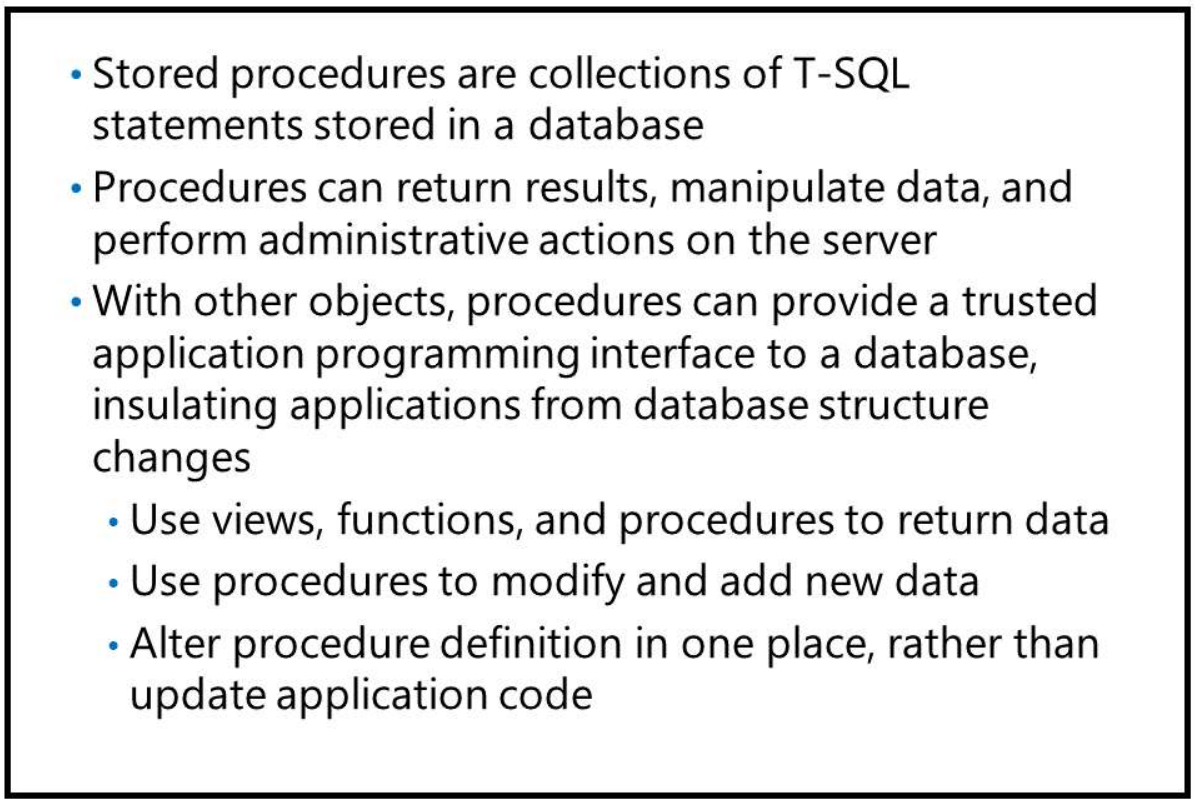
Many reporting and development tools offer the choice between writing and executing specific T-SQL SELECT statements, and choosing from queries saved as stored procedures in SQL Server. While stored procedures can encapsulate most T-SQL operations, including system administration tasks, this lesson will focus on using stored procedures to return result sets, as an alternative to writing your own SELECT statements.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe stored procedures and their use.
- Write T-SQL statements that execute stored procedures to return data.

Examining Stored Procedures



Stored procedures are named collections of T-SQL statements created with the CREATE PROCEDURE command. They encapsulate many server and database commands, and can provide a consistent application programming interface (API) to client applications using input parameters, output parameters, and return values.

Because this course focuses primarily on retrieving results from databases through SELECT statements, this lesson will only cover the use of stored procedures that encapsulate SELECT queries. However, it might be useful to note that stored procedures can also include INSERT, UPDATE, DELETE, and other valid T-SQL commands. They can also be used to provide an interface layer between a database and an application. Using such a layer, developers and administrators can ensure that all activity is performed by trusted code modules that validate input and handle errors appropriately. Elements of such an API would include:

- Views or table-valued functions as wrappers for simple retrieval.
- Stored procedures for retrieval when complex validation or manipulation is required.
- Stored procedures for inserting, updating, or deleting rows.

In addition to encapsulating code and making it easier to maintain, this approach provides a security layer. Users may be granted access to objects rather than the underlying tables themselves. This ensures that users might only

use the provided application to access data rather than other tools.

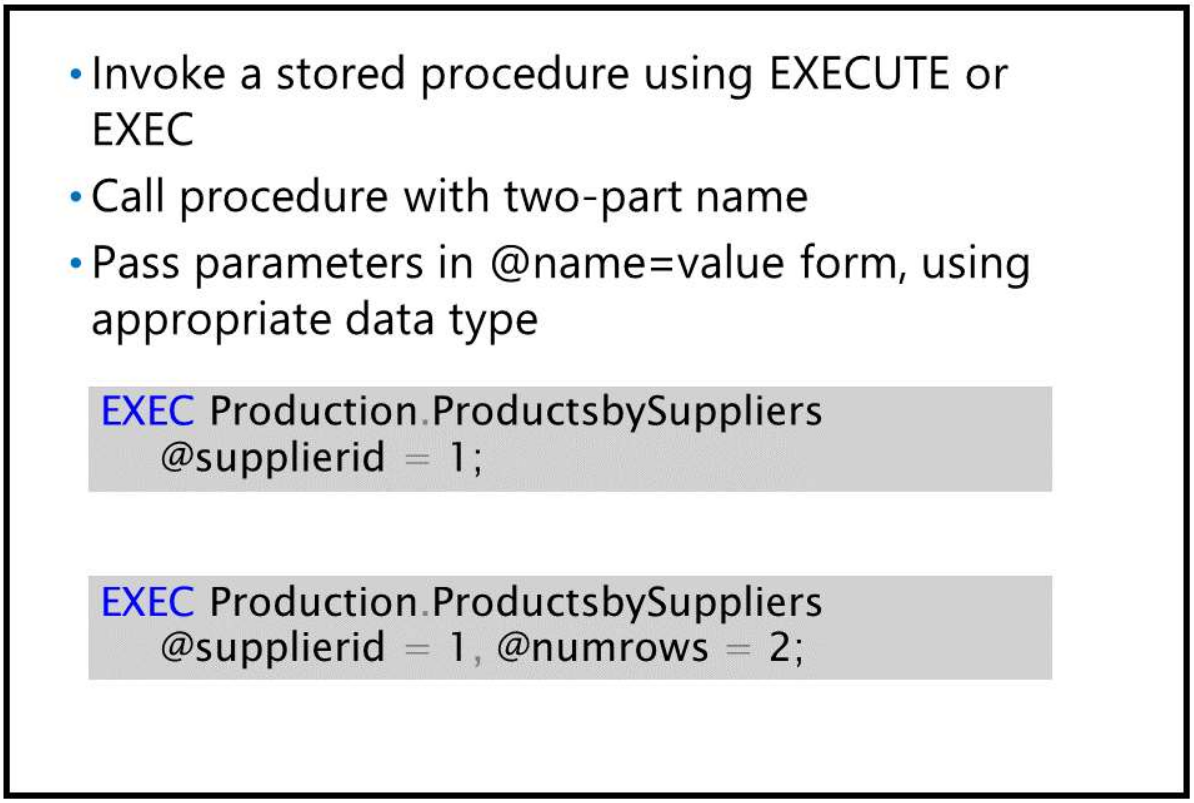
Stored procedures also offer other benefits, including network and database engine performance improvements. See the course 20762B: *Developing Microsoft SQL Server Databases* for additional information on these benefits and more details on creating and using stored procedures.

For more information, see Microsoft Docs:

Stored Procedures (Database Engine)

http://aka.ms/sshz88

Executing Stored Procedures



Earlier in this course, you learned how to execute system stored procedures. The same mechanism exists for executing user procedures. Therefore, some of the following guidelines are provided for review:

- To execute a stored procedure, use the EXECUTE command or its shortcut, EXEC, followed by the two-part name of the procedure. Your reporting tool may provide a graphical interface for selecting procedures by name, which will invoke the EXEC command for you.
- If the procedure accepts parameters, pass them as name-value pairs. For example, if the parameter is called custid and the value to pass is 5, use this form: @custid=5. Multiple parameters are separated with commas.
- Pass parameters of the appropriate data type to the stored procedure. For example, if a procedure accepts an NVARCHAR, pass in the Unicode character string format: N'string'.
- If the procedure encapsulates a simple SELECT statement, no additional elements are needed to execute it. If the procedure includes an OUTPUT parameter, additional steps will be required. See the lesson on OUTPUT

parameters later in this module.

> **Note:** You may see sample code that omits the use of the EXEC command before the name of a procedure. While this works on the first line of a batch (or in the only line of a one-line batch), this is not a best practice. Always use EXECUTE or EXEC to invoke stored procedures.

For more information, see Microsoft Docs:

Execute a Stored Procedure

http://aka.ms/Dyplvh

Demonstration: Querying Data with Stored Procedures

In this demonstration, you will see how to use stored procedures.

Demonstration Steps

Use Stored Procedures

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod15\Setup.cmd** as an administrator.
3. At the command prompt, type **y**, and press Enter.
4. Wait until the script completes, and then press Enter.
5. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
6. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod15\Demo** folder.
7. Open the **11 - Demonstration A.sql** script file.
8. Select the code under the comment **Step 1**, and then click **Execute**.
9. Select the code under the comment **Step 2**, and then click **Execute**.
10. Select the code under the comment **Step 3**, and then click **Execute**.
11. Select the code under the comment **Step 4**, and then click **Execute**.
12. Select the code under the comment **Step 5**, and then click **Execute**.
13. Select the code under the comment **Step 6**, and then click **Execute**.
14. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have the following query, which is intended to call a stored procedure called HumanResources.FilteredSkills:

```
EXEC HumanResources.FilteredSkills
  departmentid = @1, skilllevel = @4;
```

Your query returns an error. What should you do to resolve the error?

Show solution Reset

You must use the correct syntax to pass the two parameters. When passing parameters to stored procedures, place the "@" symbol before the name of the parameter, not before its value. The correct query is:

```
EXEC HumanResources.FilteredSkills
  @departmentid = 1, @skilllevel = 4;
```


Lesson 2: Passing Parameters to Stored Procedures

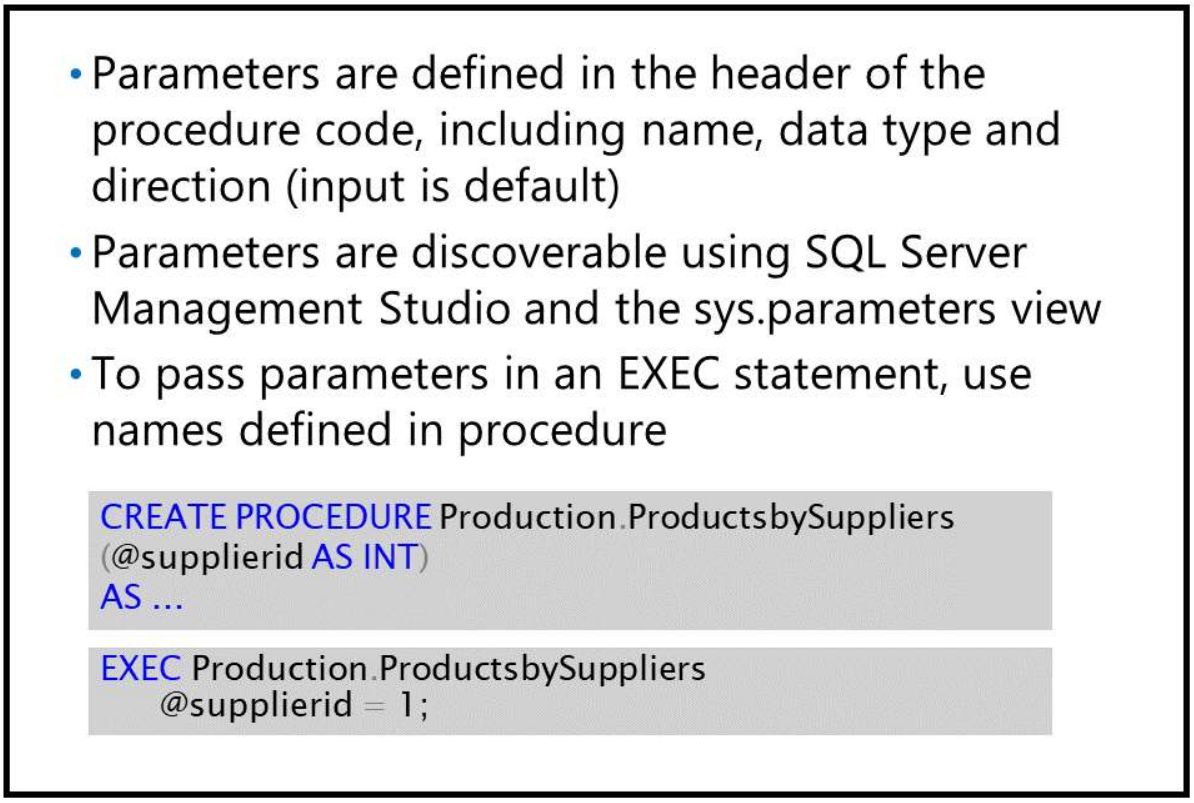
Procedures can be written to accept parameters to provide greater flexibility. Most parameters are written as input parameters, which accept values passed in the EXEC statement, and are used inside the procedure. Some procedures might also return values in the form of OUTPUT parameters, which require additional handling by the client when invoking the procedure. You will learn how to pass input and return output parameters in this lesson.

Lesson Objectives

After completing this lesson, you will be able to:

- Write EXECUTE statements that pass input parameters to stored procedures.
- Write T-SQL batches that prepare output parameters and execute stored procedures.

Passing Input Parameters to Stored Procedures



Stored procedures can be written to accept input parameters to provide greater flexibility. Procedures declare their parameters by name and data type in the header of the CREATE PROCEDURE statement, and then use the parameters as local variables in the body of the procedure. For example, an input parameter might be used in the predicate of a WHERE clause or as the value in a TOP operator.

Stored Procedure with Parameters Syntax

```
EXEC <schema_name>.<procedure_name> @<parameter_name> = <VALUE> [, ...]
```

Stored Procedure with Parameters Example

```
EXEC Production.ProductsBySuppliers @supplierid = 1;
```

Stored Procedure with Multiple Parameters Example

```
EXEC Sales.FindOrder @empid = 1, @custid=1;
```

> **Note:** The previous example refers to a procedure that does not exist in the sample database for the course. Other examples in the demonstration script for this lesson can be executed against procedures in the sample TSQL database.

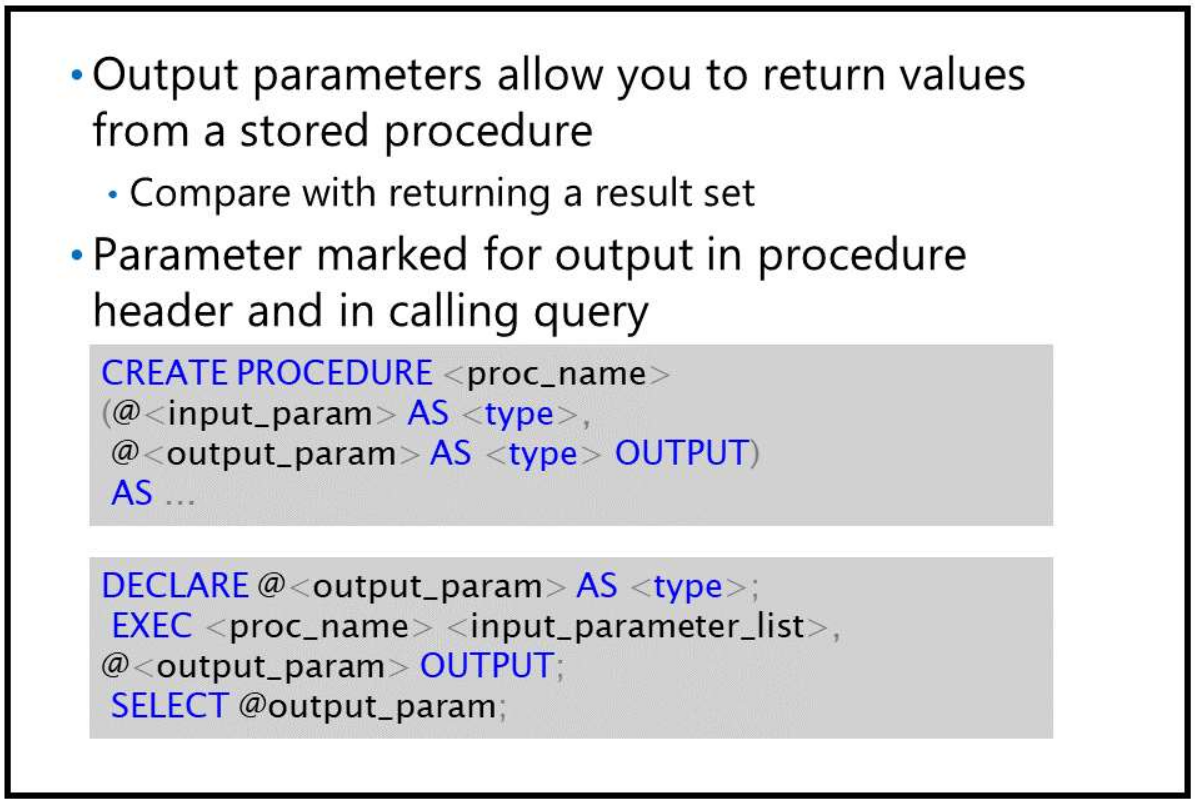
If you have not been provided with the names and data types of the parameters for the procedures you will be executing, you can typically discover them yourself, assuming you have permissions to do so. SQL Server Management Studio (SSMS) displays a parameters folder below each stored procedure that lists the names, types, and direction (input/output) of each defined parameter. Alternatively, you can query a system catalog view such as sys.parameters to retrieve parameter definitions. For an example, see the demonstration script provided for this lesson.

For more information about passing parameters to stored procedures, see Microsoft Docs:

Specify Parameters

http://aka.ms/R5zjjo

Working with OUTPUT Parameters



So far in this module, you have seen procedures that return results through an embedded SELECT statement. SQL Server also gives you the capability to return a scalar value through a parameter marked as an OUTPUT parameter. This has several benefits: a procedure can return a result set via a SELECT statement and provide an additional value, such as a row count, to the calling application. For some specific scenarios where only a single value is desired, a procedure that returns an OUTPUT parameter can perform faster than a procedure that returns the scalar value in a result set.

There are two aspects to working with stored procedures using output parameters:

1. The procedure itself must mark a parameter with the OUTPUT keyword in the parameter declaration.

   Creating a Stored Procedure with an OUTPUT Parameter Example

```
CREATE PROCEDURE Sales.GetCustPhone
(@custid AS INT, @phone AS nvarchar(24) OUTPUT)
AS ...
```

2. The T-SQL batch that calls the procedure must provide additional code to handle the output parameter. The code includes a local variable that acts as a container for the value that will be returned by the procedure when it executes. The parameter is added to the EXEC statement, marked with the OUTPUT keyword. After the stored procedure has completed, the variable will contain the value of the output parameter set inside the procedure.

Executing a Stored Procedure with OUTPUT Parameter Example

```
DECLARE @customerid INT =5, @phonenum NVARCHAR(24);
EXEC Sales.GetCustPhone @custid=@customerid, @phone=@phonenum OUTPUT;
SELECT @phonenum AS phone;
```

Demonstration: Passing Parameters to Stored Procedures

In this demonstration, you will see how to pass parameters to a stored procedure.

Demonstration Steps

Pass Parameters to a Stored Procedure

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. In Object Explorer, expand **Databases**, expand **AdventureWorks**, expand **Programmability**, and then expand **Stored Procedures**.
4. Expand any procedure, expand **Parameters**, and then point out list of parameters, data type and direction.
5. Select the code under the comment **Step 3**, and then click **Execute**.
6. Select the code under the comment **Step 4**, and then click **Execute**.
7. Select the code under the comment **Step 5**, and then click **Execute**.
8. Select the code under the comment **Step 6**, and then click **Execute**.
9. Select the code under the comment **Step 7**, and then click **Execute**.
10. Select the code under the comment **Step 8**, and then click **Execute**.
11. Select the code under the comment **Step 9**, and then click **Execute**.
12. Select the code under the comment **Step 10**, and then click **Execute**.

13. Select the code under the comment **Step 11**, and then click **Execute**.

14. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Select the best answer

A DBA has created a stored procedure in the HumanResources database by executing the following:

```
CREATE PROCEDURE HumanResources.SkillsForEmployee (@empid AS INT)
AS
  SELECT e.ID, e.FirstName, e.LastName, s.SkillName, s.Level
  FROM HumanResources.Employees AS e
  JOIN HumanResources.Skills AS s ON e.ID = s.EmployeeID
  WHERE e.ID = @empid
GO
```

You call the procedure with the following statement:

```
EXEC HumanResources.SkillsForEmployee @empid = N'24'
```

Your query returns an error. What should you do to fix your query?

Pass the @empid parameter as an integer instead of an nvarchar.

Move the position of the “@” symbol to the correct place.

Add a value for the e.ID parameter to your query.

Instead of using the stored procedure, execute your own SELECT query.

Add the OUTPUT keyword to the @empid parameter.

Check answer Show solution Reset

When you call stored procedures, you must ensure you pass parameters of the same datatypes that were defined in the stored procedure. The stored procedure expects an integer and not an NVARCHAR(). The correct query is:

```
EXEC HumanResources.SkillsForEmployee @empid = 24
```

Lesson 3: Creating Simple Stored Procedures

To better understand how to work with stored procedures written by developers and administrators, it is useful to learn how they are created. In this lesson, you will see how to write a stored procedure that returns a result set from an encapsulated SELECT statement.

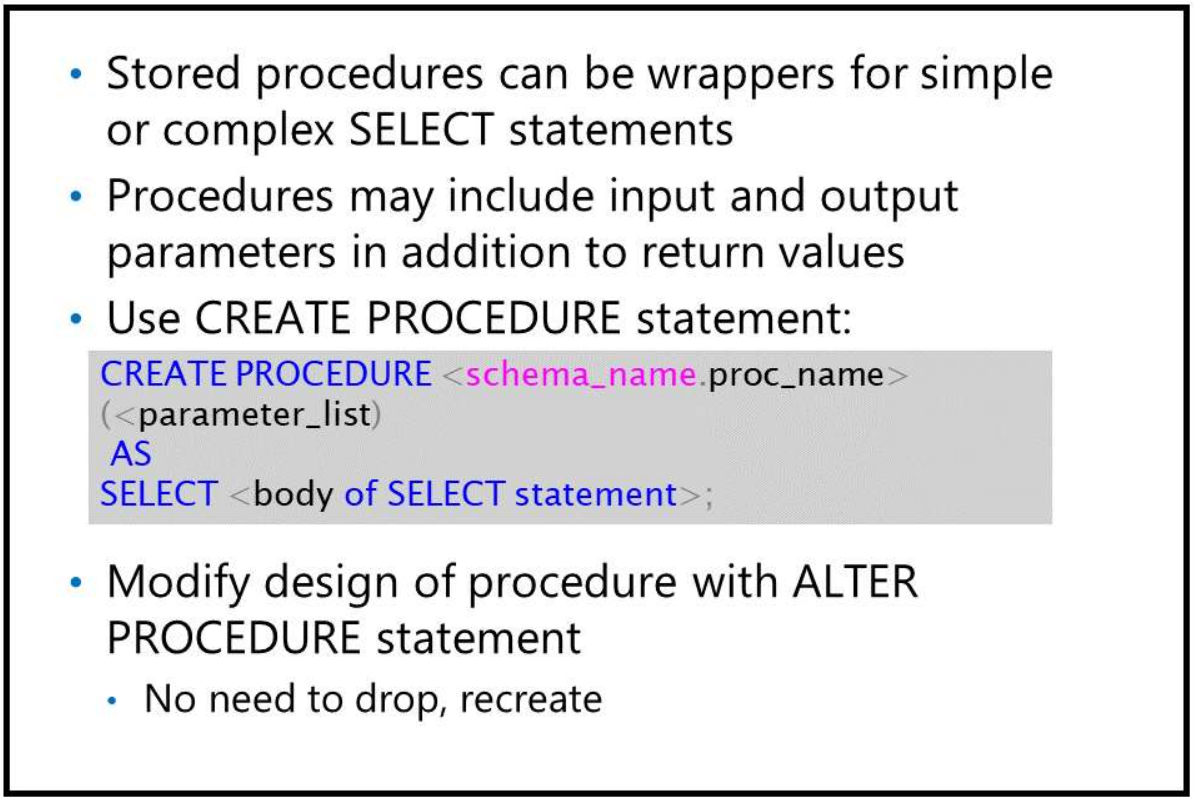
Lesson Objectives

After completing this lesson, you will be able to:

- Use the CREATE PROCEDURE statement to write a stored procedure.

- Create a stored procedure that accepts input parameters.

Creating Procedures to Return Rows



Stored procedures in SQL Server are used for many tasks, including system configuration and maintenance, in addition to data manipulation. As previously mentioned, there are advantages to creating procedures to standardize access to data. To do that, you can create a stored procedure that is a wrapper for a SELECT statement, which might include any of the data manipulations you have already learned in this course.

Example of a Procedure That Returns Rows

```
CREATE PROCEDURE Sales.OrderSummaries
AS
SELECT O.orderid, O.custid, O.empid, O.shipperid, CAST(O.orderdate AS date)AS orderdate,
  SUM(OD.qty) AS quantity,
  CAST(SUM(OD.qty * OD.unitprice * (1 - OD.discount))
       AS NUMERIC(12, 2)) AS ordervalue
FROM Sales.Orders AS O
  JOIN Sales.OrderDetails AS OD
    ON O.orderid = OD.orderid
GROUP BY O.orderid, O.custid, O.empid, O.shipperid, O.orderdate;
GO
```

Executing a Procedure That Returns Rows

```
EXEC [Sales].[OrderSummaries];
```

A partial result:

```
orderid custid empid shipperid orderdate  quantity ordervalue
------- ----- ------ --------- ---------- -------- ----------
10248       85     5         3 2006-07-04       27     440.00
10249       79     6         1 2006-07-05       49    1863.40
10250       34     4         2 2006-07-08       60    1552.60
```


To modify the design of the procedure, such as to change the columns in the SELECT list or add an ORDER BY clause, use the ALTER PROCEDURE (abbreviated ALTER PROC) statement and supply the full new code for the procedure.

Altering a Stored Procedure That Returns Rows

```
ALTER PROCEDURE Sales.OrderSummaries
AS
SELECT O.orderid, O.custid, O.empid, O.shipperid, CAST(O.orderdate AS date)AS orderdate,
  SUM(OD.qty) AS quantity,
  CAST(SUM(OD.qty * OD.unitprice * (1 - OD.discount))
       AS NUMERIC(12, 2)) AS ordervalue
FROM Sales.Orders AS O
  JOIN Sales.OrderDetails AS OD
    ON O.orderid = OD.orderid
GROUP BY O.orderid, O.custid, O.empid, O.shipperid, O.orderdate
ORDER BY orderid, orderdate;
```

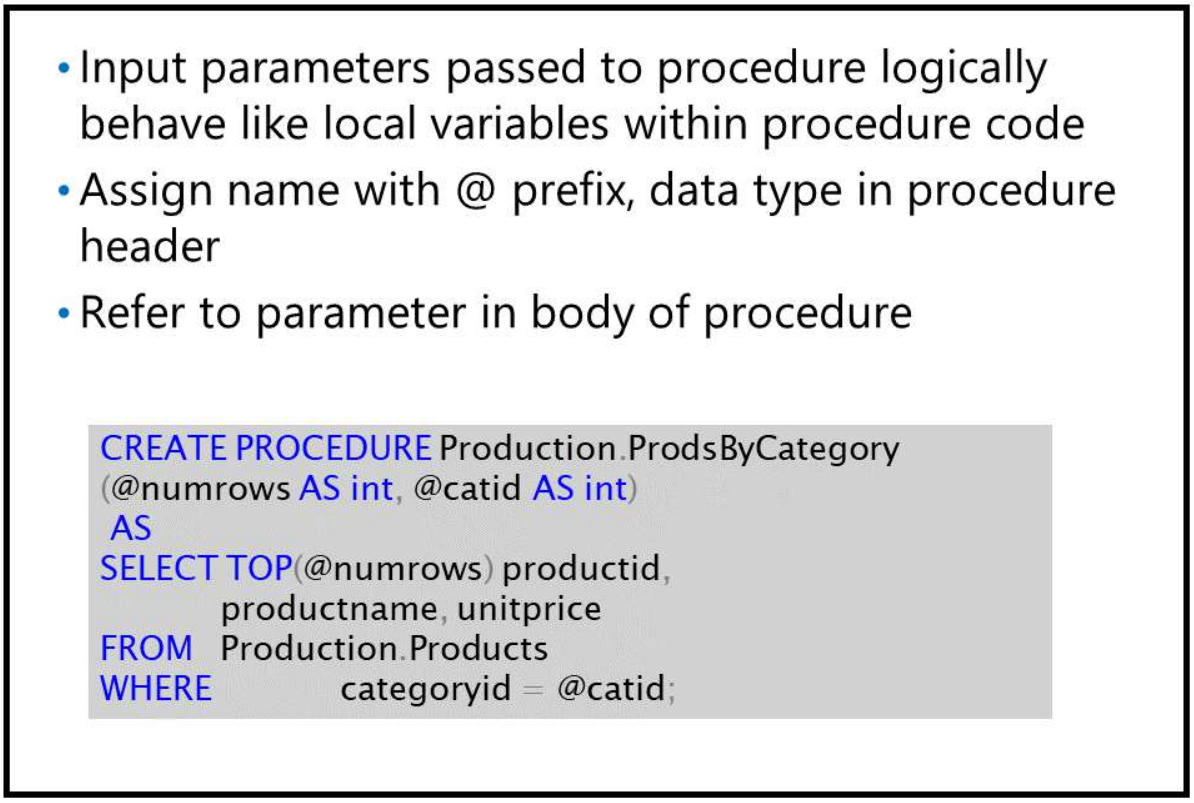
Changing the procedure with ALTER PROCEDURE is preferable to using DROP PROCEDURE to delete it, and then using CREATE PROCEDURE to rebuild it with a new definition. By altering it in place, security permissions do not need to be reassigned.

For more information on modifying stored procedures, see Microsoft Docs:

Modify a Stored Procedure

http://aka.ms/Bn33te

Creating Procedures That Accept Parameters



A stored procedure that accepts input parameters provides added flexibility to its use. To define input parameters in your own stored procedures, declare them in the header of the CREATE PROCEDURE statement, then refer to them in the body of the stored procedure. Define the parameters with an @ prefix in the name, then assign them a data type.

> **Note:** Parameters may also be assigned default values, including NULL.

Syntax of a Stored Procedure That Accepts Parameters

```
CREATE PROCEDURE <schema>.<procedure_name>
(@<parameter_name> AS <data_type>)
AS ...
```

Example of a Stored Procedure That Accepts Parameters

```
CREATE PROCEDURE Sales.OrderSummariesByEmployee
(@empid AS int)
AS
SELECT O.orderid, O.custid, O.empid, O.shipperid, CAST(O.orderdate AS date)AS orderdate,
  SUM(OD.qty) AS quantity,
  CAST(SUM(OD.qty * OD.unitprice * (1 - OD.discount))
       AS NUMERIC(12, 2)) AS ordervalue
FROM Sales.Orders AS O
```

```
  JOIN Sales.OrderDetails AS OD
    ON O.orderid = OD.orderid
WHERE empid = @empid
GROUP BY O.orderid, O.custid, O.empid, O.shipperid, O.orderdate
ORDER BY orderid, orderdate;
GO
```

Executing a Stored Procedure That Accepts Parameters

```
EXEC Sales.OrderSummariesByEmployee @empid = 5;
```

Demonstration: Creating Simple Stored Procedures

In this demonstration, you will see how to create a stored procedure.

Demonstration Steps

Create a Stored Procedure

1. In Solution Explorer, open the **31 - Demonstration C.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

The HumanResources.SkillLevelsForDepartment stored procedure is a popular procedure that ensures skills data can be examined in an anonymous form. You have been asked to add a new parameter to the stored procedure. Why should you use ALTER PROCEDURE instead of DROP PROCEDURE followed by CREATE PROCEDURE.

Show solution Reset

Because permissions assigned to the stored procedure will be maintained if you use ALTER PROCEDURE. In HumanResources databases, data is often security sensitive and you must be careful to ensure that only authorized personnel can see personal data. By using ALTER PROCEDURE, you ensure that permissions are maintained after the change. If you dropped and recreated the stored procedure, you would have to use GRANT and DENY statements to ensure security.

Lesson 4: Working with Dynamic SQL

In organizations where creating parameterized stored procedures is not supported, you might need to execute T-SQL code constructed in your application at runtime. Dynamic SQL provides a mechanism for constructing a character string that is passed to SQL Server, interpreted as a command, and executed.

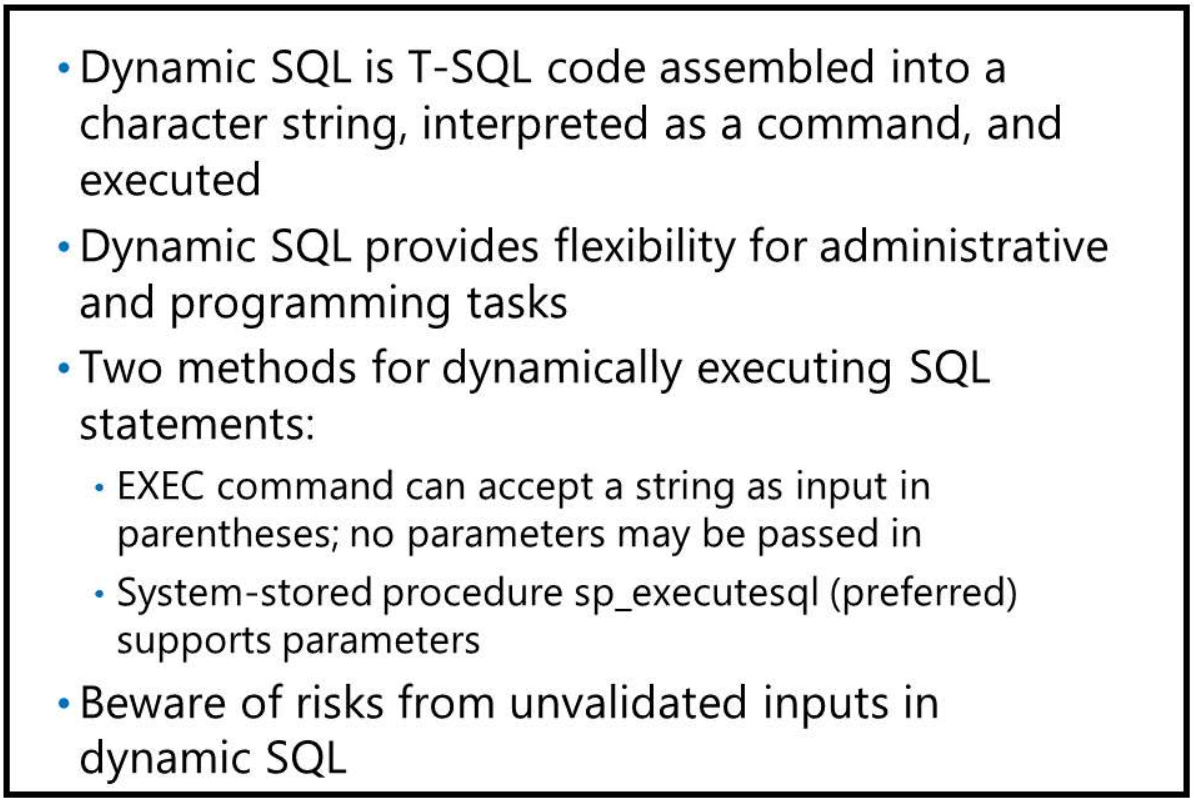
In this lesson, you will learn how to pass dynamic SQL queries to SQL Server, using the EXEC statement and the system procedure sp_executesql.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe how T-SQL can be dynamically constructed.
- Write queries that use dynamic SQL.

Constructing Dynamic SQL



Dynamic SQL provides a mechanism for constructing a character string that is passed to SQL Server, interpreted as a command, and executed. Why would you want to do this? You might not know all the values necessary for your query until execution time—such as taking the results of one query and using them as inputs to another (for example, a pivot query) or an administrative maintenance routine that accepts object names at runtime.

T-SQL supports two methods for building dynamic SQL expressions—using the EXECUTE command (or its shortcut EXEC) with a string or invoking the system-stored procedure sp_executesql:

1. The EXECUTE or EXEC command supports the use of a string as an input in the following form, but does not support parameters, which need to be combined in the input string:

   Dynamic SQL Example

   ```
   DECLARE @sqlstring AS VARCHAR(1000);
   SET @sqlstring='SELECT empid,' + ' lastname '+' FROM HR.employees;'
   EXEC(@sqlstring);
   GO
   ```

2. The system-stored procedure sp_executesql supports string input for the query, in addition to input parameters.

Passing Dynamic SQL with sp_executesql

```
DECLARE @sqlcode AS NVARCHAR(256) = N'SELECT GETDATE() AS dt';
EXEC sys.sp_executesql @statement = @sqlcode;
GO
```

It is important to know that EXEC cannot accept parameters and does not promote query plan reuse. Therefore, it is preferred that you use sp_executesql for passing dynamic SQL to SQL Server.

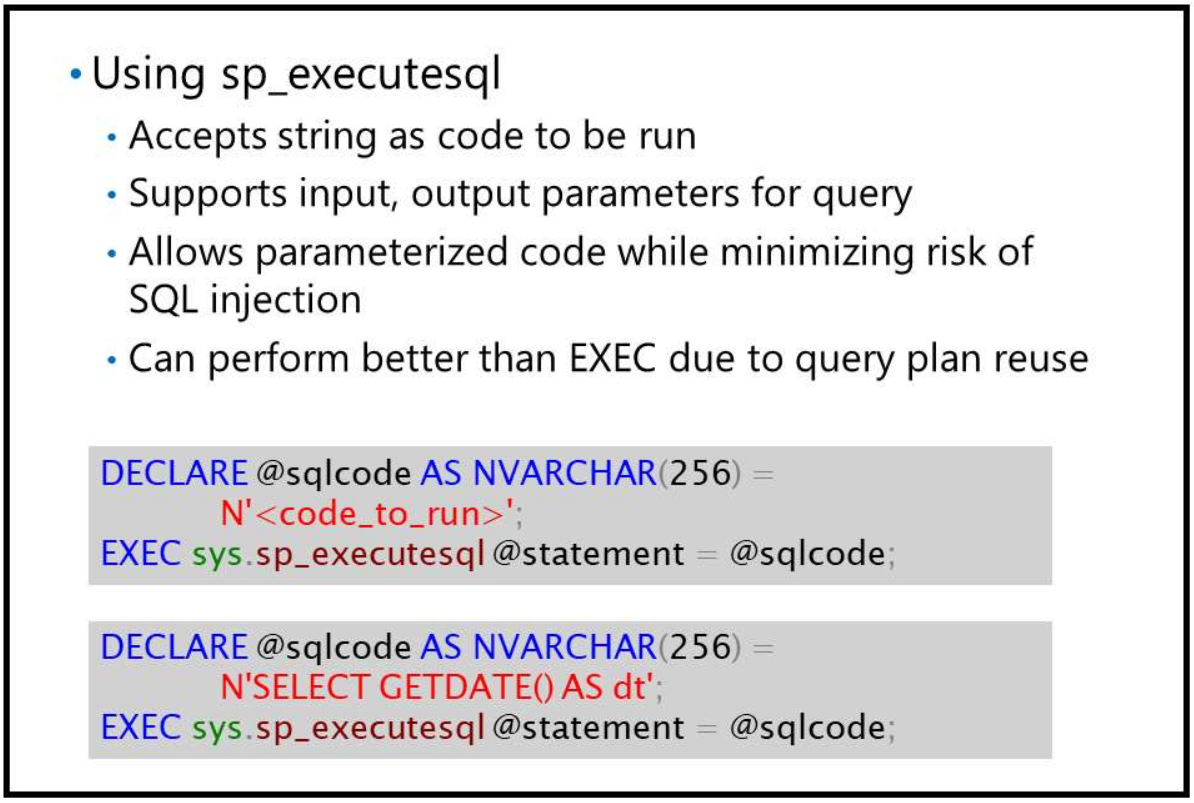
For more information, see the *Using EXECUTE with a Character String in the EXECUTE (Transact-SQL)* topic in Microsoft Docs:

EXECUTE (Transact-SQL)

http://aka.ms/Fors3z

For more information on using sp_executesql, see the next topic in this lesson.

Writing Queries with Dynamic SQL



In the previous topic, you learned that there were two methods for executing dynamic SQL. This topic focuses on the preferred method, calling sp_executesql.

Constructing and executing dynamic SQL with sp_executesql is preferred over using EXEC because EXEC cannot take parameters at runtime. In addition, sp_executesql generates execution plans that are more likely to be reused than EXEC. Perhaps most important, though, using sp_executesql can provide a line of defense against SQL injection attacks, by defining data types for parameters.

sp_executesql Syntax Example

```
DECLARE @sqlcode AS NVARCHAR(256) = N'<code_to_run>';
EXEC sys.sp_executesql @statement = @sqlcode;
GO
```

sp_executesql Example

```
DECLARE @sqlcode AS NVARCHAR(256) =
        N'SELECT GETDATE() AS dt';
EXEC sys.sp_executesql @statement = @sqlcode;
GO
```

To use sp_executesql with parameters, provide the query code, in addition to two additional parameters:

- @stmt, a Unicode string variable to hold the query text.

- @params, a Unicode string variable that holds a comma-separated list of parameter names and data types.

In addition to these two variables, you will declare and assign variables to hold the values for the parameters you wish to pass in to sp_executesql.

Using sp_executesql with Parameters

```
DECLARE @sqlstring AS NVARCHAR(1000);
DECLARE @empid AS INT;
SET @sqlstring=N'SELECT empid, lastname FROM HR.employees WHERE empid=@empid;'
EXEC sys.sp_executesql @statement = @sqlstring, @params=N'@empid AS INT',
        @empid = 5;
```

The result:

```
empid lastname
----- --------
5     Buck
```

> **Note:** sp_executesql can also use output parameters marked with the OUTPUT keyword, which you learned about earlier in this module.

For a discussion about query plan reuse and more coverage of sp_executesql, see the SQL Server Technical Documentation:

Using sp_executesql

http://go.microsoft.com/fwlink/?LinkID=402795

Demonstration: Working with Dynamic SQL

In this demonstration, you will see how to execute dynamic SQL queries.

Demonstration Steps

Execute Dynamic SQL Queries

1. In Solution Explorer, open the **41 - Demonstration D.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.

5. Select the code under the comment **Step 4**, and then click **Execute**.

6. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Discovery

You want to execute dynamic SQL with a single parameter named @skillname. In addition to the parameter itself, what other parameters should you send to sp_executesql?

Show solution Reset

@statement and @params. The sp_executesql procedure used the @statement parameter to accept the query you want to execute. Because this is a parameterized query, you must also use the @params parameter to pass the @skillname parameter.

Lab: Executing Stored Procedures

Scenario

As a business analyst for Adventure Works, you will be writing reports using corporate databases stored in SQL Server. You have been given a set of business requirements for data and will write T-SQL queries to retrieve the specified data from the databases. You have learned that some of the data can only be accessed via stored procedures instead of directly querying the tables. Additionally, some of the procedures require parameters in order to interact with them.

Objectives

After completing this lab, you will be able to:

- Use the EXECUTE statement to invoke stored procedures.
- Pass parameters to stored procedures.
- Execute system stored procedures.

Lab Setup

Estimated Time: 30 minutes

Virtual machine: **20761C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Using the EXECUTE Statement to Invoke Stored Procedures

Scenario

The IT department has supplied T-SQL code to create a stored procedure to retrieve the top 10 customers by the total sales amount. You will practice how to execute a stored procedure.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Create and Execute a Stored Procedure
3. Modify the Stored Procedure and Execute It

Detailed Steps ▲

Task 1: Prepare the Lab Environment

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab15\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Create and Execute a Stored Procedure

1. In SQL Server Management Studio, open the project file **D:\Labfiles\Lab15\Starter\Project\Project.ssmssln** and the T-SQL script **51 - Lab Exercise 1.sql**. Ensure that you are connected to the TSQL database.
2. Execute the provided T-SQL code to create the stored procedure Sales.GetTopCustomers:

```
CREATE PROCEDURE Sales.GetTopCustomers AS
SELECT TOP(10)
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC;
```

3. Write a T-SQL statement to execute the created procedure.

4. Execute the T-SQL statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab15\Solution\52 - Lab Exercise 1 - Task 1 Result.txt.

5. What is the difference between the previous T-SQL code and this one?

6. If some applications are using the stored procedure from task 1, would they still work properly after the changes you have applied in task 2?

Detailed Steps ▲

Task 3: Modify the Stored Procedure and Execute It

1. The IT department has changed the stored procedure from task 1 and supplied you with T-SQL code to apply the required changes. Execute the provided T-SQL code:

```
ALTER PROCEDURE Sales.GetTopCustomers AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

2. Write a T-SQL statement to execute the modified stored procedure.

3. Execute the T-SQL statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab15\Solution\53 - Lab Exercise 1 - Task 2 Result.txt.

Result: After this exercise, you should be able to invoke a stored procedure using the EXECUTE statement.

Exercise 2: Passing Parameters to Stored Procedures

Scenario

The IT department supplied you with additional modifications of the stored procedure in task 1. The modified stored procedure lets you pass parameters that specify the order year and number of customers to retrieve. You will practice how to execute the stored procedure with a parameter.

The main tasks for this exercise are as follows:

1. Execute a Stored Procedure with a Parameter for Order Year
2. Modify the Stored Procedure to Have a Default Value for the Parameter
3. Pass Multiple Parameters to the Stored Procedure
4. Return the Result from a Stored Procedure Using the OUTPUT Clause

Detailed Steps ▲

Task 1: Execute a Stored Procedure with a Parameter for Order Year

1. Open the SQL script **61 - Lab Exercise 2.sql**. Ensure that you are connected to the TSQL database.
2. Execute the provided T-SQL code to modify the Sales.GetTopCustomers stored procedure to include a parameter for order year (@orderyear):

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int
AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

3. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure for the year 2007.
4. Execute the T-SQL statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab15\Solution\62 - Lab Exercise 2 - Task 1_1 Result.txt.
5. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure for the year 2008.
6. Execute the T-SQL statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab15\Solution\63 - Lab Exercise 2 - Task 1_2 Result.txt.
7. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure without a parameter.
8. Execute the T-SQL statement. What happened? What is the error message?
9. If an application was designed to use the exercise 1 version of the stored procedure, would the modification made to the stored procedure in this exercise impact the usability of that application? Please explain.

Detailed Steps ▲

Task 2: Modify the Stored Procedure to Have a Default Value for the Parameter

1. Execute the provided T-SQL code to modify the Sales.GetTopCustomers stored procedure:

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int = NULL
AS
SELECT
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear OR @orderyear IS NULL
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY;
```

2. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure without a parameter.
3. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\64 - Lab Exercise 2 - Task 2 Result.txt.
4. If an application was designed to use the Exercise 1 version of the stored procedure, would the change made to the stored procedure in this task impact the usability of that application? How does this change influence the design of future applications?

Detailed Steps ▲

Task 3: Pass Multiple Parameters to the Stored Procedure

1. Execute the provided T-SQL code to add the parameter @n to the Sales.GetTopCustomers stored procedure. You use this parameter to specify how many customers you want retrieved. The default value is 10.

```
ALTER PROCEDURE Sales.GetTopCustomers
@orderyear int = NULL,
@n int = 10
AS
SELECT
```

```
c.custid,
c.contactname,
SUM(o.val) AS salesvalue
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
WHERE YEAR(o.orderdate) = @orderyear OR @orderyear IS NULL
GROUP BY c.custid, c.contactname
ORDER BY salesvalue DESC
OFFSET 0 ROWS FETCH NEXT @n ROWS ONLY;
```

2. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure without any parameters.
3. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\65 - Lab Exercise 2 - Task 3_1 Result.txt.
4. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure for order year 2008 and five customers.
5. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\66 - Lab Exercise 2 - Task 3_2 Result.txt.
6. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure for the order year 2007.
7. Execute the T-SQL statement and compare the results that you achieved with the recommended result shown in the file D:\Labfiles\Lab15\Solution\67 - Lab Exercise 2 - Task 3_3 Result.txt.
8. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure to retrieve 20 customers.
9. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\68 - Lab Exercise 2 - Task 3_4 Result.txt.
10. Do the applications using the stored procedure need to be changed because another parameter was added?

Detailed Steps ▲

Task 4: Return the Result from a Stored Procedure Using the OUTPUT Clause

1. Execute the provided T-SQL code to modify the Sales.GetTopCustomers stored procedure to return the customer contact name based on a specified position in a ranking of total sales, which is provided by the parameter @customerpos. The procedure also includes a new parameter named @customername, which has an OUTPUT option:

```
ALTER PROCEDURE Sales.GetTopCustomers
@customerpos int = 1,
```

```
@customername nvarchar(30) OUTPUT
AS
SET @customername = (
SELECT
c.contactname
FROM Sales.OrderValues AS o
INNER JOIN Sales.Customers AS c ON c.custid = o.custid
GROUP BY c.custid, c.contactname
ORDER BY SUM(o.val) DESC
OFFSET @customerpos - 1 ROWS FETCH NEXT 1 ROW ONLY
);
```

2. The IT department also supplied you with T-SQL code to declare the new variable @outcustomername. You will use this variable as an output parameter for the stored procedure.
3. DECLARE @outcustomername nvarchar(30);
4. Write an EXECUTE statement to invoke the Sales.GetTopCustomers stored procedure and retrieve the first customer.
5. Write a SELECT statement to retrieve the value of the output parameter @outcustomername.
6. Execute the batch of T-SQL code consisting of the provided DECLARE statement, the written EXECUTE statement, and the written SELECT statement.
7. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\69 - Lab Exercise 2 - Task 4 Result.txt.

Result: After this exercise, you should know how to invoke stored procedures that have parameters.

Exercise 3: Executing System Stored Procedures

Scenario

In the previous module, you learned how to query the system catalog. Now you will practice how to execute some of the most commonly used system-stored procedures to retrieve information about tables and columns.

The main tasks for this exercise are as follows:

1. Execute the Stored Procedure sys.sp_help
2. Execute the Stored Procedure sys.sp_helptext
3. Execute the Stored Procedure sys.sp_columns
4. Drop the Created Stored Procedure

Detailed Steps ▲

Task 1: Execute the Stored Procedure sys.sp_help

1. Open the SQL script **71 - Lab Exercise 3.sql**. Ensure that you are connected to the TSQL database.
2. Write an EXECUTE statement to invoke the sys.sp_help stored procedure without a parameter.
3. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\72 - Lab Exercise 3 - Task 1_1 Result.txt.
4. Write an EXECUTE statement to invoke the sys.sp_help stored procedure for a specific table by passing the parameter Sales.Customers.
5. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\73 - Lab Exercise 3 - Task 1_2 Result.txt.

Detailed Steps ▲

Task 2: Execute the Stored Procedure sys.sp_helptext

1. Write an EXECUTE statement to invoke the sys.sp_helptext stored procedure, passing the Sales.GetTopCustomers stored procedure as a parameter.
2. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\74 - Lab Exercise 3 - Task 2 Result.txt.

Detailed Steps ▲

Task 3: Execute the Stored Procedure sys.sp_columns

1. Write an EXECUTE statement to invoke the sys.sp_columns stored procedure for the table Sales.Customers. You will have to pass two parameters: @table_name and @table_owner.
2. Execute the T-SQL statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab15\Solution\75 - Lab Exercise 3 - Task 3 Result.txt.

Detailed Steps ▲

Task 4: Drop the Created Stored Procedure

- Execute the provided T-SQL statement to remove the Sales.GetTopCustomers stored procedure:

```
DROP PROCEDURE Sales.GetTopCustomers;
```

Result: After this exercise, you should have a basic knowledge of invoking different system-stored procedures.

Module Review and Takeaways

In this module, you have learned how to:

- Return results by executing stored procedures.
- Pass parameters to procedures.
- Create simple stored procedures that encapsulate a SELECT statement.
- Construct and execute dynamic SQL with EXEC and sp_executesql.

Review Question(s)

Check Your Knowledge

Discovery

What benefits do stored procedures provide for data retrieval that views do not?

Show solution Reset

Answers may vary, but ability to accept parameters is a correct answer.

Check Your Knowledge

Discovery

What form should parameter and value pairs take when passed to a stored procedure in the EXECUTE statement?

Show solution Reset

@NAME = VALUE.

Check Your Knowledge

Discovery

Which method for constructing dynamic SQL allows parameters to be passed at runtime?

Show solution Reset

Using sp_executesql.