

Module 15

Executing Stored Procedures

Module Overview

- Querying Data with Stored Procedures
- Passing Parameters to Stored Procedures
- Creating Simple Stored Procedures
- Working with Dynamic SQL

Lesson 1: Querying Data with Stored Procedures

- Examining Stored Procedures
- Executing Stored Procedures
- Demonstration: Querying Data with Stored Procedures

Examining Stored Procedures

- Stored procedures are collections of T-SQL statements stored in a database
- Procedures can return results, manipulate data, and perform administrative actions on the server
- With other objects, procedures can provide a trusted application programming interface to a database, insulating applications from database structure changes
 - Use views, functions, and procedures to return data
 - Use procedures to modify and add new data
 - Alter procedure definition in one place, rather than update application code

Executing Stored Procedures

- Invoke a stored procedure using EXECUTE or EXEC
- Call procedure with two-part name
- Pass parameters in @name=value form, using appropriate data type

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

```
EXEC Production.ProductsbySuppliers  
    @supplierid = 1, @numrows = 2;
```

Demonstration: Querying Data with Stored Procedures

In this demonstration, you will see how to:

- Use stored procedures
- Demonstration A

- **Answer**
- You must use the correct syntax to pass the two parameters.

Lesson 2: Passing Parameters to Stored Procedures

- Passing Input Parameters to Stored Procedures
- Working with OUTPUT Parameters
- Demonstration: Passing Parameters to Stored Procedures

Passing Input Parameters to Stored Procedures

- Parameters are defined in the header of the procedure code, including name, data type and direction (input is default)
- Parameters are discoverable using SQL Server Management Studio and the sys.parameters view
- To pass parameters in an EXEC statement, use names defined in procedure

```
CREATE PROCEDURE Production.ProductsbySuppliers  
(@supplierid AS INT)  
AS ...
```

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

Working with OUTPUT Parameters

- Output parameters allow you to return values from a stored procedure
 - Compare with returning a result set
- Parameter marked for output in procedure header and in calling query

```
CREATE PROCEDURE <proc_name>  
  (@<input_param> AS <type>,  
   @<output_param> AS <type> OUTPUT)  
AS ...
```

```
DECLARE @<output_param> AS <type>;  
EXEC <proc_name> <input_parameter_list>,  
@<output_param> OUTPUT;  
SELECT @output_param;
```

Demonstration: Passing Parameters to Stored Procedures

In this demonstration, you will see how to:

- Pass parameters to a stored procedure
- Demonstration B

- **Answer**

- (✓) Option 1: Pass the @empid parameter as an integer instead of an nvarchar.

Lesson 3: Creating Simple Stored Procedures

- Creating Procedures to Return Rows
- Creating Procedures That Accept Parameters
- Demonstration: Creating Simple Stored Procedures

Creating Procedures to Return Rows

- Stored procedures can be wrappers for simple or complex SELECT statements
- Procedures may include input and output parameters in addition to return values
- Use CREATE PROCEDURE statement:

```
CREATE PROCEDURE <schema_name.proc_name>  
(<parameter_list>  
AS  
SELECT <body of SELECT statement>;
```

- Modify design of procedure with ALTER PROCEDURE statement
 - No need to drop, recreate

Creating Procedures That Accept Parameters

- Input parameters passed to procedure logically behave like local variables within procedure code
- Assign name with @ prefix, data type in procedure header
- Refer to parameter in body of procedure

```
CREATE PROCEDURE Production.ProdsByCategory  
  (@numrows AS int, @catid AS int)  
AS  
SELECT TOP(@numrows) productid,  
                  productname, unitprice  
FROM Production.Products  
WHERE             categoryid = @catid;
```

Demonstration: Creating Simple Stored Procedures

In this demonstration, you will see how to:

- Create a stored procedure
- Demonstration C

Lesson 4: Working with Dynamic SQL - optional

- Constructing Dynamic SQL
- Writing Queries with Dynamic SQL
- Demonstration: Working with Dynamic SQL

Constructing Dynamic SQL

- Dynamic SQL is T-SQL code assembled into a character string, interpreted as a command, and executed
- Dynamic SQL provides flexibility for administrative and programming tasks
- Two methods for dynamically executing SQL statements:
 - EXEC command can accept a string as input in parentheses; no parameters may be passed in
 - System-stored procedure `sp_executesql` (preferred) supports parameters
- Beware of risks from unvalidated inputs in dynamic SQL

Writing Queries with Dynamic SQL

- Using `sp_executesql`
 - Accepts string as code to be run
 - Supports input, output parameters for query
 - Allows parameterized code while minimizing risk of SQL injection
 - Can perform better than EXEC due to query plan reuse

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

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

Demonstration: Working with Dynamic SQL

In this demonstration, you will see how to:

- Execute dynamic SQL queries
- Demonstration D

Module Review and Takeaways

- Review Question(s)
- **Question**
- What benefits do stored procedures provide for data retrieval that views do not?
- **Question**
- What form should parameter and value pairs take when passed to a stored procedure in the EXECUTE statement?