Lab Answer Key: Module 15: Executing Stored Procedures

Lab: Executing Stored Procedures

Exercise 1: Using the EXECUTE Statement to Invoke Stored Procedures

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. In the **D:\Labfiles\Lab15\Starter** folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.

Task 2: Create and Execute a Stored Procedure

1. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine using Windows authentication.
2. On the **File** menu, click **Open** and click **Project/Solution**.
3. In the **Open Project** window, open the project **D:\Labfiles\Lab15\Starter\Project\Project.ssmssln**.
4. In Solution Explorer, expand **Queries**, and then double-click **51 - Lab Exercise 1.sql**.
5. In the query window, highlight the statement **USE TSQL;** and click **Execute** on the toolbar.
6. Highlight the following T-SQL code under the **Task 1** description:

```
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;
```

7. Click **Execute**. You have created a stored procedure named Sales.GetTopCustomers.
8. In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXECUTE Sales.GetTopCustomers;
```

9. Highlight the written T-SQL code and click **Execute**. You have executed the stored procedure.

Task 3: Modify the Stored Procedure and Execute It

1. Highlight the following T-SQL code after the **Task 2** description:

```
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. Click **Execute**. You have modified the Sales.GetTopCustomers stored procedure.
3. In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXECUTE Sales.GetTopCustomers;
```

4. Highlight the written T-SQL code and click **Execute**. You have executed the modified stored procedure.
5. Compare both the code and the result of the two versions of the stored procedure. What is the difference between them? In the modified version, the TOP option has been replaced with the OFFSET-FETCH option. Despite this change, the result is the same.

If some applications had been using the stored procedure in task 1, would they still work properly after the change you applied in task 2? Yes, since the result from the stored procedure is still the same. This demonstrates the huge benefit of using stored procedures as an additional layer between the database and the application/middle tier. Even if you change the underlying T-SQL code, the application would work properly without any changes. There are also other benefits of using stored procedures in terms of performance (for example, caching and reuse of plans) and security (for example, preventing SQL injections).

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

Exercise 2: Passing Parameters to Stored Procedures

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

1. In Solution Explorer, double-click the query **61 - Lab Exercise 2.sql**.
2. When the query window opens, highlight the statement **USE TSQL;** and click **Execute**.
3. Highlight the following T-SQL code under the **Task 1** description:

```
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;
```

4. Click **Execute**. You have modified the Sales.GetTopCustomers stored procedure to accept the parameter @orderyear. Notice that the modified stored procedure uses a predicate in the WHERE clause that isn't a search argument. This predicate was used to keep things simple. The best practice is to avoid such filtering because it does not allow efficient use of indexing. A better approach would be to use the DATETIMEFROMPARTS function to provide a search argument for orderdate:

```
WHERE o.orderdate >= DATETIMEFROMPARTS(@orderyear, 1, 1, 0, 0, 0, 0)
AND o.orderdate < DATETIMEFROMPARTS(@orderyear + 1, 1, 1, 0, 0, 0, 0)
```

5. In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2007;
```

Notice that you are passing the parameter by name as this is considered the best practice. There is also support for passing parameters by position. For example, the following EXECUTE statement would retrieve the same result as the T-SQL code you just typed:

```
EXECUTE Sales.GetTopCustomers 2007;
```

6. Highlight the written T-SQL code and click **Execute**.

7. After the previous T-SQL code, type the following T-SQL code to execute the stored procedure for the order year 2008:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2008;
```

8. Highlight the written T-SQL code and click **Execute**.

9. After the previous T-SQL code, type the following T-SQL code to execute the stored procedure without specifying a parameter:

```
EXECUTE Sales.GetTopCustomers;
```

10. Highlight the written T-SQL code and click **Execute**.

11. Observe the error message:

    Procedure or function 'GetTopCustomers' expects parameter '@orderyear', which was not supplied.

    This error message is telling you that the @orderyear parameter was not supplied.

12. Suppose that an application named MyCustomers is using the exercise 1 version of the stored procedure. Would the modification made to the stored procedure in this exercise impact the usability of the GetCustomerInfo application? Yes. The exercise 1 version of the stored procedure did not need a parameter, whereas the version in this exercise does not work without a parameter. To avoid problems, you can add a default parameter to the stored procedure. That way, the MyCustomers application does not have to be changed to support the @orderyear parameter.

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

1. Highlight the following T-SQL code under the **Task 2** description:

```
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. Click **Execute**. You have modified the Sales.GetTopCustomers stored procedure to have a default value (NULL) for the @orderyear parameter. You have also included an additional logical expression to the WHERE clause.

3. In the query pane, type the following T-SQL code after the previous one:

```
EXECUTE Sales.GetTopCustomers;
```

This code tests the modified stored procedure by executing it without specifying a parameter.

4. Highlight the written query and click **Execute**.

5. Observe the result. How do the changes to the stored procedure in task 2 influence the MyCustomers application and the design of future applications? The changes enable the MyCustomers application to use the modified stored procedure, and no changes need to be made to the application. The changes add new possibilities for future applications because the modified stored procedure accepts the order year as a parameter.

Task 3: Pass Multiple Parameters to the Stored Procedure

1. Highlight the following T-SQL code under the **Task 3** description:

```
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. Click **Execute**. You have modified the Sales.GetTopCustomers stored procedure to have an additional parameter named @n. You can use this parameter to specify how many customers to retrieve. The default value is 10.

3. After the previous T-SQL code, type the following T-SQL code to execute the modified stored procedure:

```
EXECUTE Sales.GetTopCustomers;
```

4. Highlight the written query and click **Execute**.
5. After the previous T-SQL code, type the following T-SQL code to retrieve the top five customers for the year 2008:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2008, @n = 5;
```

6. Highlight the written query and click **Execute**.
7. After the previous T-SQL code, type the following T-SQL code to retrieve the top 10 customers for the year 2007:

```
EXECUTE Sales.GetTopCustomers @orderyear = 2007;
```

8. Highlight the written query and click **Execute**.
9. After the previous T-SQL code, type the following T-SQL code to retrieve the top 20 customers:

```
EXECUTE Sales.GetTopCustomers @n = 20;
```

10. Highlight the written query and click **Execute**.
11. Do the applications using the stored procedure need to be changed because another parameter was added?
No changes need to be made to the application.

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

1. Highlight the following T-SQL code under the **Task 4** description:

```
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. Click **Execute**.
3. Find the following DECLARE statement in the provided code:

```
DECLARE @outcustomername nvarchar(30);
```

This statement declares a parameter named @outcustomername.

4. After the DECLARE statement, add code that uses the OUTPUT clause to return the stored procedure's result as a variable named @outcustomername. Your code, together with the provided DECLARE statement, should look like this:

```
DECLARE @outcustomername nvarchar(30);
EXECUTE Sales.GetTopCustomers @customerpos = 1, @customername = @outcustomername
OUTPUT;
SELECT @outcustomername AS customername;
```

5. Highlight all three T-SQL statements and click **Execute**.

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

Exercise 3: Executing System Stored Procedures

Task 1: Execute the Stored Procedure sys.sp_help

1. In Solution Explorer, double-click the query **71 - Lab Exercise 3.sql**.
2. In the query window, highlight the statement **USE TSQL;** and click **Execute**.
3. In the query pane, type the following T-SQL code after the **Task 1** description:

```
EXEC sys.sp_help;
```

4. Highlight the written query and click **Execute**.
5. In the query pane, type the following T-SQL code after the previous T-SQL code:

```
EXEC sys.sp_help N'Sales.Customers';
```

6. Highlight the written query and click **Execute**.

Task 2: Execute the Stored Procedure sys.sp_helptext

1. In the query pane, type the following T-SQL code after the **Task 2** description:

```
EXEC sys.sp_helptext N'Sales.GetTopCustomers';
```

2. Highlight the written query and click **Execute**.

Task 3: Execute the Stored Procedure sys.sp_columns

1. In the query pane, type the following T-SQL code after the **Task 3** description:

```
EXEC sys.sp_columns @table_name = N'Customers', @table_owner = N'Sales';
```

2. Highlight the written query and click **Execute**.

Task 4: Drop the Created Stored Procedure

- Highlight the provided T-SQL statement under the **Task 4** description and click **Execute**.

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