8

The Command Object

In the earlier chapter we learned how to use the Recordset object to perform some complex database operations. In this chapter we will learn about the Command object and how it is used to execute stored procedures in MS SQL Server.

WHAT ARE STORED PROCEDURES?

Stored procedures are SQL statements that are syntax-checked, compiled and stored in the database. A stored procedure can contain one SQL statement or a set of SQL statements. In MS SQL Server you can also include Transact-SQL statements.

Transact-SQL is Microsoft's extension of the SQL language. It extends the SQL language into a programming language by adding support for variables and conditional programming techniques such as the WHILE loop.

Following are some key benefits of using stored procedures:

- Since stored procedures are syntax-checked and compiled when they are created, they execute faster. The difference is noticeable if the SQL statement is complex. Normal SQL statements have to be syntax-checked and then compiled before execution.
- If you have a complicated set of SQL statements that you wish to execute in different ASP files, you can store them in a stored procedure and execute the procedure itself. This reduces a lot of code in your script and ensures that the same procedure is executed each time.
- By executing a stored procedure which has multiple SQL statements, you can reduce the number of trips back and forth from the web server to the database, thus reducing load on the server.
- You can define both output and input parameters in a stored procedure, which makes it flexible. The same procedure can return a different result based on these parameters.
- You can configure the database such that users can modify the contents of only the table by using stored procedures, which increases security of your data.

- You can execute one procedure from another. This means that you can build a complex procedure-based operation and reuse smaller procedures for many different purposes.

Therefore, when you wish to execute any SQL statement from your script, you must consider a stored procedure; you will notice the difference in the long run.

CREATING A STORED PROCEDURE

Unfortunately MS Access does not support stored procedures, so you will need MS SQL Server to try out the examples or pieces of code given in this chapter.

You can create a stored procedure using the ***CREATE PROCEDURE*** statement of the SQL language. A simple stored procedure would be as follows:

```
CREATE PROCEDURE getBooks AS SELECT * FROM Books
```

You can create this stored procedure either by using the ***Microsoft Query Analyser***, ***MS SQL Server Manager*** or within an ASP script by using the ***Execute*** method of the Connection object. When this statement is executed, it will retrieve all the rows in the table named Books. You can execute the stored procedure from within SQL Server by using the ***EXECUTE*** statement, i.e.

```
EXECUTE getBooks
```

Most stored procedures will not be as simple as the above procedure. They will usually contain variables and more than one SQL statement. The stored procedure shown below executes an SQL statement depending on the random number that is generated.

Sample 8.1

```
1. CREATE PROCEDURE RandomSQL
2. AS
3.    DECLARE @rndval INT
4.    SELECT @rndval = RAND() * 100
5.    IF @rndval > 50
6.       SELECT * FROM Customers
7.    ELSE
8.       SELECT * FROM Products
```

A variable 'rndval' of type INTEGER is declared in Line 3 (variables in SQL server are defined with an @ symbol at the beginning of the variable name) and is assigned a random number between 0 and 100 generated by the function ***RAND*** (Line 4). Depending on the value of the rndval variable, the appropriate SQL statement is executed.

To delete a stored procedure, use the SQL statement ***DROP PROCEDURE*** like this,

```
DROP PROCEDURE RandomSQL
```

The system stored procedure ***sp_help*** will list all the stored procedures in the current database along with the tables and views that are created.

STORED PROCEDURES WITH PARAMETERS

You can also pass input and output parameters to a stored procedure. The simplest form is a stored procedure that returns an integer return value. The following example creates a procedure that checks if the book exists and returns a return code.

Sample 8.2

```
1. CREATE PROCEDURE CheckBook
2. AS
3.    IF EXISTS (SELECT Book_Name FROM Books WHERE Book_Name = 'C Premiere')
4.       RETURN (1)
5.    ELSE
6.       RETURN (0)
```

The ***RETURN*** statement stops all further execution of a procedure and returns the value passed as a parameter to the caller. In this case, 1 is returned if the book with the name 'C Premiere' exists in the table Books, else 0 is returned.

You can execute this procedure from SQL Server by executing the following statements:

```
1. DECLARE @StatusCode INT
2. EXECUTE @StatusCode = CheckBook
3. SELECT @StatusCode
```

Notice that the book name has been stored in the procedure itself, which does not make it flexible. It would be great if the procedure checks for any book that you specify. To do this we need to modify the script slightly.

Sample 8.3

```
1. CREATE PROCEDURE CheckBook
2. ( @bookname VARCHAR(40) )
3. AS
4.    IF EXISTS (SELECT Book_Name FROM Books WHERE Book_Name = @bookname)
5.       RETURN (1)
6.    ELSE
7.       RETURN (0)
```

We have modified the procedure to accept an input parameter named @bookname. The SELECT statement now checks if there exists a book that has the name passed to the variable and then returns an appropriate 1 or 0 value.

You can now execute the procedure.

Sample 8.4

```
1. DECLARE @StatusCode INT
```

```
2. EXECUTE @StatusCode = CheckBook 'The Memphis Blues'
3. SELECT @StatusCode
```

The above examples illustrated a simple return value, but what if you needed to return more than one value, for example author name and publisher. In such situations we have to define the output parameters. The output parameter unlike the return code can have data of any type supported by SQL server. An example is shown below:

Sample 8.5

```
1. CREATE PROCEDURE CheckBook
2. (@bookname VARCHAR(40), @authorname VARCHAR(60) OUTPUT,
   @pubname VARCHAR(60) OUTPUT)
3. AS
4.     SELECT @authorname = Author_Name,
5.            @pubname = Publisher
6.     FROM Books
7.     WHERE Book_Name = @bookname
```

The procedure accepts one input parameter and returns two output parameters (@authorname and @pubname). The output parameters are identified by the ***OUTPUT*** keyword (Line 2). If the SQL statement on Line 4 does not return any rows, the output parameters are set to NULL.

The execution of the procedure is shown below:

Sample 8.6

```
1. DECLARE @au_name VARCHAR(60)
2. DECLARE @pu_name VARCHAR(60)
3. EXECUTE CheckBook 'The Memphis Blues', @au_name OUTPUT, @pu_name OUTPUT
4. PRINT @au_name
5. PRINT @pu_name
```

The ***OUTPUT*** keyword has to be used while executing the stored procedure. The *PRINT* statement on Lines 4 and 5 will print the values stored in the variables in the SQL statement if the stored procedure returns a row. The maximum number of input and output parameters that can be passed to a stored procedure is 1,024.

STORED PROCEDURES AND THE COMMAND OBJECT

You can execute a stored procedure using the Execute method of the Connection object like any SQL statement. Consider the following stored procedure:

Sample 8.7

```
1. CREATE PROCEDURE Insert_Book
2. (@bookname VARCHAR(40), @authorname VARCHAR(60), @pubname VARCHAR(60))
3. AS
4. @book_no INT
5.
6.    SELECT @book_no = MAX(Book_Id)
7.    FROM Books
8.
9.    SELECT @book_no = @book_no + 1
10.
11.    INSERT INTO Books
12.    (Book_Id, Book_Name, Author_Name, Publisher)
13.    VALUES (@book_no, @bookname, @authorname, @pubname)
```

This stored procedure inserts a record into the table Books. It accepts three input parameters, @bookname, @authorname and @pubname. Line 6 retrieves the maximum value of book_id field and increments it by one in Line 9. The record is then inserted into the table using the ***INSERT INTO SQL*** statement.

The ASP script to execute this stored procedure is given below:

Sample 8.8

```
1. <!–#include file="adovbs.inc"–>
2. <%
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Conn.Execute "Insert_Book 'My Last Days', 'Freud', 'Mad Publication'"
7.
8. Conn.Close
9. Set Conn = Nothing
10. %>
```

Executing a stored procedure does not return a Recordset object like a normal SQL statement. It is simple to execute a stored procedure using the Connection object. However, there is one major drawback, you cannot retrieve return codes or output parameters. So what is the solution?

THE COMMAND OBJECT

The command object takes over where the Connection object fails. The main use of this object is to execute stored procedures although it can be used for many other things.

You can use the Command object to execute simple to very complex stored procedures. Let us take a simple example.

Sample 8.9

```
1. CREATE PROCEDURE UpdateHits
2. AS
3.   UPDATE Hitsdb
4.   SET hits = hits + 1
```

The above script updates the Hitsdb table by incrementing the value of the field hits by 1. The ASP script that executes this stored procedure using the Command object is given below:

Sample 8.10

```
1. <!-#include file="adovbs.inc"->
2. <%
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Set MyComm = Server.CreateObject("ADODB.Command")
7. MyComm.ActiveConnection = Conn
8. MyComm.CommandType = adCMDStoredProc
9. MyComm.CommandText = "UpdateHits"
10. MyComm.Execute
11.
12. Set MyComm = Nothing
13. Conn.Close
14. Set Conn = Nothing
15. %>
```

An instance of the Command object is created in Line 6, while Line 7 sets the ***ActiveConnection*** property of the Command object to the open Connection object 'Conn'. The ***CommandType*** property on Line 8 tells the Command object that a stored procedure is to be executed. The name of the stored procedure is passed to the ***CommandText*** property (Line 9) and the Execute method (Line 10) is called to execute the stored procedure and update the Hitsdb table.

RETURN CODES AND THE COMMAND OBJECT

In this section we will discuss how both input as well as output parameters are passed to the Command object. But first we will study how the Command object traps the return code. Let us take the stored procedure we created earlier, the existence of a book named 'C Premiere'.

Sample 8.11

```
1. <!-#include file="adovbs.inc"->
2. <%
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Set MyComm = Server.CreateObject("ADODB.Command")
7. MyComm.ActiveConnection = Conn
8. MyComm.CommandType = adCMDStoredProc
9. MyComm.CommandText = "CheckBook"
10. Set MyParam = MyComm.CreateParameter("RetCode", adInteger,
                                         adParamReturnValue)
11. MyComm.Parameters.Append MyParam
12. MyComm.Execute
13.
14. Response.Write MyComm("RetCode")
15.
16. Set MyComm = Nothing
17. Conn.Close
18. Set Conn = Nothing
19. %>
```

In the above example, the ***CreateParameter*** method of the Command object is used to define a parameter that will return a return code (adParameterReturnValue), named 'RetCode' of data type integer (Line 10). The CreateParameter method returns a Parameter object, which stores all parameters (input, output and return codes). The ***Parameter*** object is then appended to the ***Parameters*** collection of the Command object (Line 11). The stored procedure is then executed and the return code is printed to the user from the variable 'RetCode'.

Let us visit the CreateParameter method of the Command object again. This method is used for defining both input and output parameters. The syntax of the CreateParameter method is as follows:

```
<Command_object>.CreateParameter (<parameter_name>, <data type_of_parameter>,
                                  <type_of_parameter>, <size_of_data>)
```

A brief explanation of each argument is given in Table 8.1.

I would like to remind the readers that all the constants beginning with 'ad' are defined in the '***adovbs.inc***' file, which must be included at the beginning of the script, else the script will generate an error.

USING THE COMMAND OBJECT WITH INPUT PARAMETERS

Now that we have learned to retrieve a return code from the stored procedure, let us move on to defining and passing input parameters to the stored procedure. This script is also very similar to the previous one. Consider the stored procedure in sample 8.12.

Table 8.1

Argument	*Description*
parameter_name	Specifies the name of the parameter. For an input parameter this value is set before the stored procedure is executed, while for an output parameter the name of the parameter stores the output value after the execution of the stored procedure
data type_of_parameter	Specifies the data type of the parameter. Common data types are adVarChar, adInteger and adLongVarChar which correspond to the VARCHAR, INTEGER and TEXT data types in SQL Server
type_of_parameter	Indicates the type of parameter. The common types of parameters are adParamReturnValue, adParamInput and adParamOutput which correspond to the Return code, Input and Output type of parameters
size_of_data	Specifies the data size of the parameter. This parameter is specified only if you define an input or output parameter of any data type other than the adInteger family

Sample 8.12

```
1. CREATE PROCEDURE Show_Books
2. (@pubname VARCHAR(60))
3. AS
4.    SELECT Book_Name
5.    FROM Books
6.    WHERE Publisher = @pubname
```

Suppose we wish to retrieve all the book names whose publishers are Tata McGraw-Hill. The ASP script for this will be as follows:

Sample 8.13

```
1. <!-#include file="adovbs.inc"->
2. <%
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Set MyComm = Server.CreateObject("ADODB.Command")
7. MyComm.ActiveConnection = Conn
8. MyComm.CommandType = adCMDStoredProc
9. MyComm.CommandText = "Show_Books"
10. MyComm.Parameters.Append MyComm.CreateParameter("PubName", adVarchar,
    AdParamInput, 60)
11. MyComm.Parameters("PubName") = 'Tata McGraw-Hill'
12.
```

```
13. Set RecBooks = MyComm.Execute()
14.
15. While Not RecBooks.EOF
16.    Response.Write RecBooks("Book_Name") & "<BR>"
17.    RecBooks.MoveNext
18. Wend
19.
20. Set MyComm = Nothing
21. Conn.Close
22. Set Conn = Nothing
23. %>
```

This example is similar to the previous one, except that the parameters variable PubName (Line 10) has an additional parameter. The last parameter defines the size of the input or output parameter and is mandatory if you have defined an input or output parameter. Since this is an input parameter, its value is set on Line 11 and the stored procedure is executed.

The result from the Execute method of the Command object is a Recordset object (Line 13) and iterate (Line 15) through the recordset. The returned Recordset object is a forward-only cursor and supports a read-only lock.

USING THE COMMAND OBJECT WITH OUTPUT PARAMETERS

Since we have discussed passing input parameters to the Command object, let us move on to output parameters. If you wish to retrieve the publisher's name based on the name of the book, do not open a Recordset, instead use a stored procedure. It is much faster.

A sample stored procedure is given below:

Sample 8.14

```
1. CREATE PROCEDURE Show_Publisher
2. (@bookname VARCHAR(60), @pubname VARCHAR(60) OUTPUT)
3. AS
4.    SELECT @pubname = Publisher
5.    FROM Books
6.    WHERE Book_Name = @bookname
```

To retrieve the value of the publisher, you must make use of the output parameter of the Command object. The script to do this is given below:

Sample 8.15

```
1. <!-#include file="adovbs.inc"->
2. <%
```

```
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Set MyComm = Server.CreateObject("ADODB.Command")
7. MyComm.ActiveConnection = Conn
8. MyComm.CommandType = adCMDStoredProc
9. MyComm.CommandText = "Show_Publisher"
10. MyComm.Parameters.Append MyComm.CreateParameter("BookName", adVarchar,
                                                    AdParamInput, 60)
11. MyComm.Parameters.Append MyComm.CreateParameter("PubName", adVarchar,
                                                    AdParamOutput, 60)
12. MyComm.Parameters("BookName") = 'C Premiere'
13.
14. Set RecBooks = MyComm.Execute()
15.
16. Response.Write MyComm("PubName")
17.
18. Set MyComm = Nothing
19. Conn.Close
20. Set Conn = Nothing
21. %>
```

The script works exactly like the previous one. The only addition is Line 11 where the output parameter is appended in the Parameter collection. The stored procedure executes the SQL statement using the input parameter and returns the output to the PubName output parameter, which is displayed to the user (Line 16).

RETRIEVING PARAMETER INFORMATION FROM A STORED PROCEDURE

You may want to use a stored procedure but might not know its parameters. For example, it is important to know the data type and size before you use the procedure. The Parameters collection of the Command object allows you to list out all the parameters of a stored procedure. The script is given below:

Sample 8.16

```
1. <!-#include file="adovbs.inc"->
2. <%
3. Set Conn = Server.CreateObject("ADODB.Connection")
4. Conn.Open "MyBooks", "sa", ""
5.
6. Set MyComm = Server.CreateObject("ADODB.Command")
7. MyComm.ActiveConnection = Conn
```

```
 8. MyComm.CommandType = adCMDStoredProc
 9. MyComm.CommandText = "Show_Publisher"
10. MyComm.Parameters.Refresh
11. %>
12.
13. <HTML>
14. <HEAD><TITLE>List of parameters of a stored procedure</TITLE></HEAD>
15. <BODY>
16. <H2>Lists of parameters for stored procedure 'Show_Publisher'</H2>
17. <TABLE BORDER=1>
18.   <TR>
19.     <TH>Name of parameter</TH>
20.     <TH>Data type</TH>
21.     <TH>Type of parameter</TH>
22.     <TH>Data size</TH>
23.   </TR>
24.
25. <% For Each MyParams in MyComm.Parameters %>
26.   <TR>
27.     <TD> <% = MyParams.Name %> </TD>
28.     <TD> <% = MyParams.Type %> </TD>
29.     <TD> <% = MyParams.Direction %> </TD>
30.     <TD> <% = MyParams.Size %> </TD>
31.   </TR>
32. <% Next
33.
34. Set MyComm = Nothing
35. Conn.Close
36. Set Conn = Nothing
37. %>
38.
39. </TABLE>
40. </BODY>
41. </HTML>
```

The above script lists out all the parameter names, data types, type of parameter and the size. When the ***Refresh*** method (Line 10) is executed, the information about the stored procedure is retrieved from the database.

The script retrieves numeric codes, which you will have to map to constants from the adovbs.inc file to get meaningful values.