**Introduction**

A month or so ago, I wrote a [few articles](http://www.codeguru.com/columns/vb/doing-data-extraction-with-visual-basic.net.htm) introducing you to the world of databases and VB. If you have followed them, you have seen that it is not particularly difficult to use databases properly from VB. Today's article is no exception. Today I will talk about using stored procedures built into SQL Server, with your VB.NET programs.

**What are Stored Procedures?**

Well, technically, [here](http://technet.microsoft.com/en-us/library/aa174792%28v=sql.80%29.aspx) is a great definition; but, a **stored procedure** is simply a means of keeping your database layer apart from your actual program's code. You store whatever SQL commands you'd like to execute in a stored procedure that you can use externally. Another benefit of a stored procedure is that it eliminates the risk of SQL Injection, as all the parameters used will be populated dynamically.

**Creating the Stored Procedures**

Before we can create a stored procedure to do all of the above mentioned tasks, we obviously need a database. I may be captain obvious here, but open SQL Server and create a new database named Test. If you do not want to create a database, leave it at the default SQL Server database, which is **Master**.

**Create Table Stored Procedure**

Create a new SQL Query and type the following:

1. CREATE PROCEDURE CreateTable
3. AS
5. BEGIN
7. CREATE TABLE tblSPExample(
8. StudentID int IDENTITY(1,1),
9. Name varchar(50) NOT NULL,
10. Surname varchar(50) NOT NULL,
11. Age int NOT NULL,
12. PRIMARY KEY (StudentID) )
14. END

This creates a procedure in SQL named CreateTable. Then it tells the procedure what to do. This is a standard SQL statement creating a table with the fields, their data types and each field's constraints.

**Insert Into Stored Procedure**

Create a new SQL Query and type the following:

1. CREATE PROCEDURE InsertDataIntoTable
2. @Name varchar(50),
3. @Surname varchar(50),
4. @Age int
6. AS
8. BEGIN
10. INSERT INTO tblSPExample(Name, Surname, Age)
12. VALUES ( @Name, @Surname, @Age)

15. END

Before you create the Insert statement, you must specify the **parameters** that this query will expect. These parameters will be populated during runtime of your application.

**Update  Stored Procedure**

Create a new SQL Query and type the following:

1. CREATE PROCEDURE UpdateDataInsideTable
2. @Name varchar(50),
3. @Surname varchar(50),
4. @Age int,
5. @StudentID int
7. AS
9. BEGIN
11. UPDATE tblSPExample
13. SET Name = @Name, Surname = @Surname, Age = @Age
15. WHERE StudentID = @StudentID

18. END

**Delete  Stored Procedure**

Create a new SQL Query and type the following:

1. CREATE PROCEDURE DeleteDataFromTable
3. @StudentID int
5. AS
7. BEGIN
9. DELETE FROM tblSPExample
11. WHERE StudentID = @StudentID

14. END

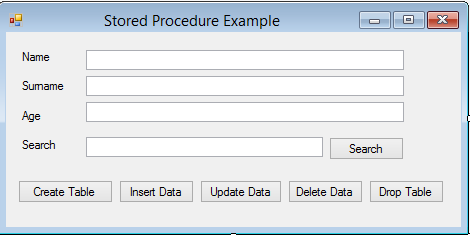
**Drop Table Stored Procedure**

Create a new SQL Query and type the following:

1. CREATE PROCEDURE DropTable
3. AS
5. BEGIN
7. DROP TABLE tblSPExample
9. END

**Our VB.NET Project**

Now we can move on to create the VB.NET project. Name it anything you like and design the form to resemble Figure 1.

  
*Figure 1 - Our Design*

**Coding**

In all honesty, most of the code I will demonstrate today will be more or less the same. I live by the principle: Keep it simple stupid. This has brought me very far, even if I am not the sharpest tool in the shed.

As always, let me start with the **Imports** statement(s) needed for today's example. Add this line on top of your class definition:

1. Imports System.Data.SqlClient 'Import SQL Capabilities

Now, create the **modular variables**:

1. Private strConn As String = "Data Source=HANNES;Initial Catalog=Master;Integrated Security=True"
2. Private sqlCon As SqlConnection

The first object, **strConn** specifies our database connection string. It is broken down into three parts:

1. The database server, which in this case is called Hannes
2. The database name which is Master
3. Security settings

The next object, **sqlCon** will be used as a SQLConnection object, which assists in connecting to the physical data store specified earlier.

Now, let's get the party started!

This next sub you will be creating is used just to load the data from the database and display it to the user. There is no use of a stored procedure as that would have been overkill. Add this sub:

1. Private Sub LoadData()
3. Dim strQuery As String
5. strQuery = "SELECT \* FROM tblSPExample"
7. sqlCon = New SqlConnection(strConn)
9. Using (sqlCon)
11. Dim sqlComm As SqlCommand = New SqlCommand(strQuery, sqlCon)
13. sqlCon.Open()
15. Dim sqlReader As SqlDataReader = sqlComm.ExecuteReader()
17. If sqlReader.HasRows Then
19. While (sqlReader.Read())
21. txtName.Text = sqlReader.GetString(1)
22. txtSurname.Text = sqlReader.GetString(2)
23. txtAge.Text = sqlReader.GetValue(3)
25. End While

28. End If
30. sqlReader.Close()
32. End Using
34. End Sub

This should look familiar if you have read my previous article concerning databases. If you haven't read any of them yet, here is an introduction to get you started. Anyways, it is quite simple really. This is what happened in the **LoadData** sub:

1. I created a string object to host the ultimate query string.
2. I stored the query inside the object. This query selects all the data from the **tblSPExample** table. This table you will create later via the Stored Procedure you created later.
3. I open the connection to the database.
4. Created a command object. This object will execute the query, to return the desired data to the **DataReader**.
5. If the DataReader can identify data, it will return each row into the associated textboxes.
6. I close all the connections to the database.

Create the **CreateTable** sub now:

1. Private Sub CreateTable()
3. sqlCon = New SqlConnection(strConn)

6. Using (sqlCon)
8. Dim sqlComm As New SqlCommand
10. sqlComm.Connection = sqlCon
12. sqlComm.CommandText = "CreateTable"
13. sqlComm.CommandType = CommandType.StoredProcedure

16. sqlCon.Open()
18. sqlComm.ExecuteNonQuery()
20. End Using
22. End Sub

It's the same principle here, except that we modify the Command object a little to specify the stored procedure we will be using; then, we execute it as a NonQuery. NonQuery means that there is no data being returned, just a simple action the SQL code should take.

Now that you have a table, you can insert data into it. Create the **InsertNewRecord** sub now:

1. Private Sub InsertNewRecord()
3. sqlCon = New SqlConnection(strConn)
5. Using (sqlCon)
7. Dim sqlComm As New SqlCommand()
9. sqlComm.Connection = sqlCon
11. sqlComm.CommandText = "InsertDataIntoTable"
12. sqlComm.CommandType = CommandType.StoredProcedure
14. sqlComm.Parameters.AddWithValue("FirstName", txtName.Text)
15. sqlComm.Parameters.AddWithValue("Surname", txtSurname.Text)
16. sqlComm.Parameters.AddWithValue("Age", Integer.Parse(txtAge.Text))
18. sqlCon.Open()
20. sqlComm.ExecuteNonQuery()
22. End Using

25. LoadData()
27. End Sub

Again, still following the same principle - see, I told you it is easy! There are a couple of new statements in there. The statements in the middle add parameters to the command object. These parameters were specified inside the Insert Stored Procedure. By using parameters it easier to manage the data being sent into the SQL Command object. Here, you also gave each parameter a value, which is the associated textbox.

Now add the following two subs. One sub is to edit the information inside a database table; the other is to delete certain information:

1. Private Sub UpdateRecord()
3. sqlCon = New SqlConnection(strConn)
5. Using (sqlCon)
7. Dim sqlComm As New SqlCommand
9. sqlComm.Connection = sqlCon

12. sqlComm.CommandText = "UpdateDataInsideTable"
13. sqlComm.CommandType = CommandType.StoredProcedure
15. sqlComm.Parameters.AddWithValue("Name", txtName.Text)
16. sqlComm.Parameters.AddWithValue("Surname", txtSurname.Text)
17. sqlComm.Parameters.AddWithValue("Age", Integer.Parse(txtAge.Text))
19. sqlCon.Open()
21. sqlComm.ExecuteNonQuery()
23. End Using
25. LoadData()
27. End Sub
29. Private Sub DeleteRecord()
31. sqlCon = New SqlConnection(strConn)

34. Using (sqlCon)
36. Dim sqlComm As New SqlCommand
38. sqlComm.Connection = sqlCon
40. sqlComm.CommandText = "DeleteDataFromTable"
41. sqlComm.CommandType = CommandType.StoredProcedure
43. sqlComm.Parameters.AddWithValue("StudentID", Integer.Parse(txtSearch.Text))

46. sqlCon.Open()
48. sqlComm.ExecuteNonQuery()
50. End Using
52. LoadData()
54. End Sub

The next sub is to drop a desired database table from a database:

1. Private Sub DropTable()
3. sqlCon = New SqlConnection(strConn)

6. Using (sqlCon)
8. Dim sqlComm As New SqlCommand
10. sqlComm.Connection = sqlCon
12. sqlComm.CommandText = "DropTable"
13. sqlComm.CommandType = CommandType.StoredProcedure

16. sqlCon.Open()
18. sqlComm.ExecuteNonQuery()
20. End Using
22. End Sub

Very simple indeed, agree?

**Conclusion**

As you can see, if you have the basic fundamentals of database design and VB.NET, you can create very powerful yet small programs to extract and manipulate data. Obviously, this is just a small example; but it can be used in any major application. I hope you have enjoyed this little article. Until next time, cheers!