15. C# Database Connection Fundamentals

**ADO.NET** is a data access technology from Microsoft .Net Framework, which provides communication between relational and non-relational systems through a common set of components.

15.1 C# database programming

The .Net Framework includes mainly three Data Providers for ADO.NET. The Microsoft SQL Server, OLEDB and ODBC are the main Data Providers in the .Net Framework. From the following pages you can see each component of ADO.NET and database programming in details with C# Source Code.

The .Net Framework includes mainly three Data Providers for ADO.NET. They are the

- Microsoft SQL Server Data Provider,
- OLEDB Data Provider
- ODBC Data Provider.

SQL Server uses the SqlConnection object, OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively.
15.2 DataSet

**DataSet** provides a disconnected representation of result sets from the Data Source, and it is completely independent from the Data Source. DataSet provides much greater flexibility when dealing with related Result Sets.

**DataSet** consists of a collection of **DataTable** objects that you can relate to each other with DataRelation objects. The DataTable contains a collection of **DataRow** and **DataColumn** Object which contains Data. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

15.3 C# ADO.NET Connection String

**Connection String** is a normal String representation which contains Database connection information to establish the connection between Database and the Application. The **Connection String** includes parameters such as the name of the driver, Server name and Database name, as well as security information such as user name and password.

Usually **Data Providers** use a connection string containing a collection of parameters to establish the connection with the database through applications. The **.NET Framework** provides mainly three data providers, they are
Here you can see how to make a connection string to the following ADO.NET Data Providers.

**Microsoft SQL Server Connection String**

```
connectionString="Data Source=ServerName;Initial Catalog=DatabaseName;
User ID=UserName;Password=Password"
```

**OLEDB Data Provider Connection String**

```
connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=yourdatabasename.mdb;"
```

**ODBC Connection String**

```
connectionString="Driver={Microsoft Access Driver (*.mdb)};
DBQ=yourdatabasename.mdb;"
```

You have to provide the necessary connection information to the Connection String attributes.

From the following section you can see how to these ADO.NET Data Providers establish connection to the Data Source through C# language.

### 15.3.1 C# SQL Server Connection

You can connect your C# application to data in a SQL Server database using the .NET Framework Data Provider for SQL Server. The first step in a C# application is to create an instance of the Server object and to establish its connection to an instance of Microsoft SQL Server.

The SqlConnection Object is Handling the part of physical communication between the C# application and the SQL Server Database. An instance of the SqlConnection class in C# is supported the Data Provider for SQL Server Database. The SqlConnection instance takes Connection String as argument and pass the value to the Constructor statement.

**Sql Server connection string**

```
connectionString="Data Source=ServerName;
Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
```
If you have a named instance of SQL Server, you'll need to add that as well.

"Server=localhost\sqlexpress"

When the connection is established, SQL Commands will execute with the help of the Connection Object and retrieve or manipulate the data in the database. Once the Database activities is over, Connection should be closed and release the Data Source resources.

```csharp
    cnn.Close();
```

The Close() method in SqlConnection Class is used to close the Database Connection. The Close method rolls back any pending transactions and releases the Connection from the SQL Server Database.

A Sample C# Program that connect SQL Server using connection string.

```csharp
using System;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = null;
            SqlConnection cnn;
            connectionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password"
            cnn = new SqlConnection(connectionString);
            try
            {
                cnn.Open();
                MessageBox.Show("Connection Open ! ");
                cnn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
Connecting to SQL Server using windows authentication

"Server= localhost; Database= employeedetails; Integrated Security=SSPI;"

A sample c# program that demonstrate how to execute sql statement and read data from SQL server.

```csharp
using System;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = null;
            SqlConnection connection;
            SqlCommand command;
            string sql = null;
            SqlDataReader dataReader;
            connectionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password";
            sql = "Your SQL Statement Here , like Select * from product";
            connection = new SqlConnection(connectionString);
            try
            {
                connection.Open();
                command = new SqlCommand(sql, connection);
                dataReader = command.ExecuteReader();
                while (dataReader.Read())
                {
                    MessageBox.Show(dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + " - " + dataReader.GetValue(2));
                }
                dataReader.Close();
                command.Dispose();
                connection.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
```
15.4 C# ADO.NET SqlCommand

The SqlCommand Object in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the C# Connection Object. The Command Object requires an instance of a C# Connection Object for executing the SQL statements.

In order to retrieve a resultset or execute an SQL statement against a Data Source, first you have to create a Connection Object and open a connection to the Data Source specified in the connection string. Next step is to assign the open connection to the connection property of the SqlCommand Object. Then the Command Object can execute the SQL statements. After the execution of the SQL statement, the Command Object will return a result set. We can retrieve the result set using a Data Reader.

The SqlCommand Object has a property called CommandText, which contains a String value that represents the command that will be executed against the Data Source.

When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure.

15.4.1 SqlCommand - ExecuteNonQuery

The ExecuteNonQuery() is one of the most frequently used method in SqlCommand Object, and is used for executing statements that do not return result sets (ie. statements like insert data, update data etc.).

```
Command.ExecuteNonQuery();
```

The ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures, Views etc. perform by the ExecuteNonQuery(). Also Data
Manipulation tasks like Insert, Update, Delete etc. also perform by the ExecuteNonQuery() of SqlCommand Object.

The following C# example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

```csharp
using System;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = null;
            SqlConnection cnn;
            SqlCommand cmd;
            string sql = null;
            connectionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password";
            sql = "Your SQL Statement Here";
            cnn = new SqlConnection(connectionString);
            try
            {
                cnn.Open();
                cmd = new SqlCommand(sql, cnn);
                cmd.ExecuteNonQuery();
                cmd.Dispose();
                cnn.Close();
                MessageBox.Show("ExecuteNonQuery in SqlCommand executed !!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
    }
}
```

15.4.1 SqlCommand - ExecuteReader

The ExecuteReader() in C# SqlCommand Object sends the SQL statements to the Connection Object and populate a SqlDataReader Object based on the SQL statement. When the ExecuteReader method in
SqlCommand Object execute, it will instantiate a SqlClient.SqlDataReader Object.

using System;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            string connectionString = null;
            SqlConnection cnn;
            SqlCommand cmd;
            string sql = null;
            SqlDataReader reader;
            connectionString = "Data Source=ServerName;InitialCatalog=DatabaseName;User ID=UserName;Password=Password";
            sql = "Select * from product";

            cnn = new SqlConnection(connectionString);
            try
            {
                cnn.Open();
                cmd = new SqlCommand(sql, cnn);
                reader = cmd.ExecuteReader();
                while (reader.Read())
                {
                    MessageBox.Show(reader.GetValue(0) + " - " + reader.GetValue(1) + " - " + reader.GetValue(2));
                }
                reader.Close();
                cmd.Dispose();
                cnn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}

15.5. C# ADO.NET DataReader

DataReader Object in ADO.NET is a stream-based, forward-only, read-only retrieval of query results from the Data Sources, which do not update
The DataReader cannot be created directly from code, they can created only by calling the **ExecuteReader** method of a Command Object.

```csharp
SqlDataReader sqlReader = sqlCmd.ExecuteReader();
```

The DataReader Object provides a connection oriented data access to the Data Sources. A **Connection Object** can contain only one DataReader at a time and the connection in the DataReader remains open, also it cannot be used for any other purpose while data is being accessed. The **Read()** method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist.

```csharp
DataReader.Read();
```

Usually we are using two types of DataReader in ADO.NET. They are **SqlDataReader** and the **OleDbDataReader**.

### 15.6 C# ADO.NET SqlDataReader

**SqlDataReader Object** provides a connection oriented data access to the SQL Server data Sources from C# applications. **ExecuteReader()** in the SqlCommand Object sends the SQL statements to the SqlConnection Object and populate a SqlDataReader Object based on the SQL statement or Stored Procedures.

```csharp
SqlDataReader sqlReader = sqlCmd.ExecuteReader();
```

The Read() method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist.

```csharp
SqlDataReader.Read()
public Form1()
{
    InitializeComponent();
}

private void button1_Click(object sender, EventArgs e)
{
    string connectionString = null;
    SqlConnection sqlCnn;
    SqlCommand sqlCmd;
    string sql = null;

    connectionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=UserName;Password=Password";
    sql = "Your SQL Statement Here, like Select * from product";

    sqlCnn = new SqlConnection(connectionString);
    try
    {
        sqlCnn.Open();
        sqlCmd = new SqlCommand(sql, sqlCnn);
        SqlDataReader sqlReader = sqlCmd.ExecuteReader();
        while (sqlReader.Read())
        {
            MessageBox.Show(sqlReader.GetValue(0) + " - " + sqlReader.GetValue(1) + " - " + sqlReader.GetValue(2));
        }
        sqlReader.Close();
        sqlCmd.Dispose();
        sqlCnn.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Can not open connection ! ");
    }
}
}