Chapter Five

C# Database Connection

Introduction

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. **ADO.NET** consist of a set of Objects that expose data access services to the .NET environment. ADO.NET is designed to be easy to use, and Visual Studio provides several wizards and other features that you can use to generate **ADO.NET** data access code.

ADO.NET is a standard.NET class library for accessing databases, processing data and XML

- Allows executing SQL in RDBMS systems
- DB connections, data readers, DB commands

DataSet

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 D**ataAdapter** Object provides a **bridge** between the DataSet and the Data Source.

5. C# ADO.NET Connection String

In order to access the database, you need to provide connection parameters, such as the machine that the database is running on, and possibly your login credentials. Anyone who has worked with ADO will be familiar with the .NET connection classes, like OleDbConnection and SqlConnection. **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 includes mainly three Data Providers for ADO.NET. They are the

- ✓ Microsoft **SQL Server** Data Provider,
- ✓ OLEDB Data Provider and
- ✓ **ODBC** Data provider.

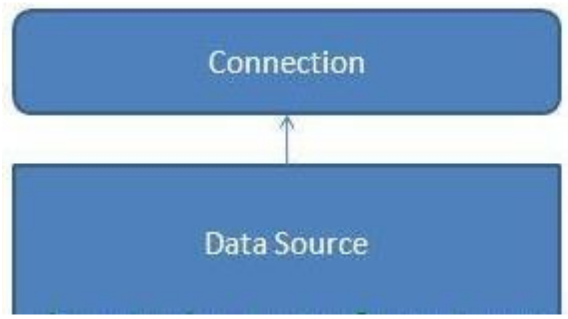
The four Objects from the .Net Framework provide the functionality of Data Providers in ADO.NET. They are **Connection** Object, **Command** Object, **DataReader** Object and **DataAdapter** Object.

5.1 C# ADO.NET Connection

The **Connection** Object is a part of **ADO.NET** Data Provider and it is a unique session with the Data Source.

The **Connection** Object connect to the specified Data Source and open a connection between the C# application and the Data Source, depends on the parameter specified in the **Connection String**. When the connection is established, SQL Commands will execute with the help of the Connection Object and retrieve or manipulate data in the Data Source.

Once the Database activity is over, **Connection** should be closed and release the Data Source resources.



In C# the type of the Connection is depend on which Data Source system you are working with. The following are the commonly used Connections from the C# applications.

5.1.1 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

```
connetionString="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.

```
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.

```
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 connetionString = null;
            SqlConnection cnn ;
connetionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User
ID=UserName;Password=Password"

            cnn = new SqlConnection(connetionString); try

            {
                cnn.Open();

                MessageBox.Show ("Connection Open ! ");
                cnn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
```

Connect via an IP address

```
connetionString="Data Source=IP_ADDRESS,PORT;

Network Library=DBMSSOCN;Initial Catalog=DatabaseName; User
ID=UserName;Password=Password"
```

1433 is the default port for SQL Server.

5.1.2 C# OLEDB Connection

The C# **OleDbConnection** instance takes Connection String as argument and pass the value to the Constructor statement. An instance of the C# OleDbConnection class is supported the **OLEDB Data Provider**.

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

cnn = new OleDbConnection(connetionString);
```

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

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

The Close() method in the OleDbConnection class is used to close the Database Connection. The Close method **Rolls Back** any pending transactions and releases the Connection from the Database connected by the OLEDB Data Provider.

```
using System;

using System.Windows.Forms;
using System.Data.OleDb;

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

      private void button1_Click(object sender, EventArgs e)
      {

         string connetionString = null;
         OleDbConnection cnn ;

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

         cnn = new OleDbConnection(connetionString); try

         {
            cnn.Open();

            MessageBox.Show ("Connection Open ! ");
            cnn.Close();
         }
         catch (Exception ex)
         {
            MessageBox.Show("Can not open connection ! ");
         }
      }
   }
}
```

5.1.3 C# ODBC Connection

An instance of the **OdbcConnection Class** in C# is supported the ODBC Data Provider. The OdbcConnection instance takes Connection String as argument and pass the value to the Constructor statement. When the connection is established between C# application and the Data Source the SQL Commands will execute with the help of the **Connection Object** and retrieve or manipulate data in the database.

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

cnn = new OdbcConnection(connetionString);
```

Once the Database activities is over you should be closed the Connection and release the Data Source resources . The Close() method in **OdbcConnection Class** is used to close the Database Connection.

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

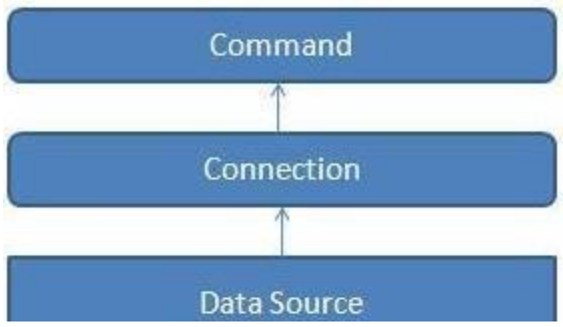
The Close method rolls back any pending transactions and releases the Connection from the Database connected by the **ODBC Data Provider**.

```
using System;

using System.Windows.Forms;
using System.Data.Odbc;

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

        private void button1_Click(object sender, EventArgs e)
        {
            string connetionString = null;
            OdbcConnection cnn ;

            connetionString = "Driver={Microsoft Access Driver
(*.mdb)};DBQ=yourdatabasename.mdb;";
            cnn = new OdbcConnection(connetionString);
try
        {
            cnn.Open();

            MessageBox.Show ("Connection Open ! ");
            cnn.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Can not open connection ! ");
        }
      }
    }
}
```

5.2 C# ADO.NET Command

The **Command 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 result set 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 **Command 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**.

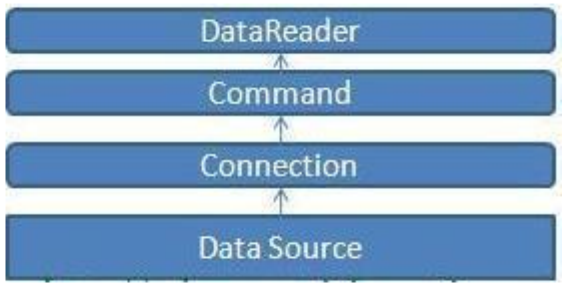


The **Command Object** has a property called **CommandText**, which contains a String value that represents the command that will be executed against the Data Source. When the

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

5.3 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 data. The DataReader cannot be created directly from code, they can created only by calling the **ExecuteReader** method of a Command Object.



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

C# ADO.NET OleDbDataReader

OleDbDataReader Object provides a connection oriented data access to the OLEDB Data Sources through C# applications. **ExecuteReader()** in the OleDbCommand Object sends the commands to OleDbConnection Object and populate an OleDbDataReader Object based on the SQL statements as well as Stored Procedures passed through the OleDbCommand Object.

OleDbDataReader oledbReader = oledbCmd.ExecuteReader();

When the ExecuteReader method in OleDbCommand Object execute, it will instantiate an **OleDb.OleDbDataReader** Object. When we started to read from an OleDbDataReader it should always be open and positioned prior to the first record. The **Read()** method in the OleDbDataReader is used to read the rows from the OleDbDataReader and it always moves forward to a new valid row, if any row exist. **DataReader.Read();**

5.3.1 ExecuteReader:

ExecuteReader used for getting the query results as a DataReader object. It is readonly forward only retrieval of records and it uses select command to read through the table from the first to the last.

```
SqlDataReader reader;
reader = Command.ExecuteReader();
while (reader.Read())
        {
        MessageBox.Show(reader.Item(0));
        }
 reader.Close();
```

OleDbDataReader.Read()

A Sample C# Program that read data from Microsoft Access db.

```
using System;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         string connetionString = null;
         OleDbConnection   oledbCnn
         ; OleDbCommand oledbCmd
         ; string sql = null;
         connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Your mdb
filename;";
         sql = "Your SQL Statement Here like Select * from product";
         oledbCnn = new OleDbConnection(connetionString);
         try
         {
            oledbCnn.Open();
            oledbCmd = new OleDbCommand(sql, oledbCnn);
            OleDbDataReader oledbReader = oledbCmd.ExecuteReader();
            while (oledbReader.Read ())
            {
               MessageBox.Show(oledbReader.GetValue(0) + " - " +
oledbReader.GetValue(1) + " - " + oledbReader.GetValue(2));
```

```
                }
                oledbReader.Close();
                oledbCmd.Dispose();
                oledbCnn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
```

5.3.2 C# ADO.NET OleDbCommand ExecuteNonQuery

ExecuteNonQuery : ExecuteNonQuery used for executing queries that does not return any data. It is used to execute the sql statements like update, insert, delete etc. ExecuteNonQuery executes the command and returns the number of rows affected.

```
int retValue = 0;
Command = new SqlCommand(Sql, Connection);
retValue = Command.ExecuteNonQuery();
```

The **ExecuteNonQuery()** is one of the most frequently used method in OleDbCommand Object and is used for executing statements that do not returns result sets. The ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also.

```
cmd.ExecuteNonQuery();
```

The Data Definition tasks like creating Stored Procedures, Views etc. perform by the **ExecuteNonQuery()** . Also Data Manipulation tasks like Insert , Update , Delete etc. perform by the ExecuteNonQuery() of OleDbCommand object.

```
using System;

using System.Windows.Forms;
using System.Data.OleDb;

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

        private void button1_Click(object sender, EventArgs e)
        {

            string connetionString = null;
            OleDbConnection cnn ;
            OleDbCommand cmd ;
            string sql = null;


connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Your mdb
filename;"
sql = "Your SQL Statement Here"


            cnn = new OleDbConnection(connetionString); try

            {

                cnn.Open();
                MessageBox.Show("Connection Opened
                "); cmd = new OleDbCommand(sql, cnn);
                cmd.ExecuteNonQuery();
                cmd.Dispose();
                cnn.Close();

                MessageBox.Show ("ExecuteNonQuery in OleDbConnection
executed !!");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! " + ex.ToString());
            }
        }
    }
}
```

5.3.3 C# ADO.NET OleDbCommand - ExecuteScalar

The **ExecuteScalar()** in C# OleDbCommand Object uses to retrieve a single value from the Database after the execution of SQL Statements or Stored Procedures.

The ExecuteScalar() executes SQL statements or Stored Procedure and returned a scalar value on first column of first row in the returned Result Set. If the **Result Set** contains more than one columns or rows , it will return only the first column of the first row value and ignore all other values . If the Result Set is empty it will return a **NULL** reference.

```
Int32 count = Convert.ToInt32(cmd.ExecuteScalar()); or
int count = int.Parse(cmd.ExecuteScalar());
```

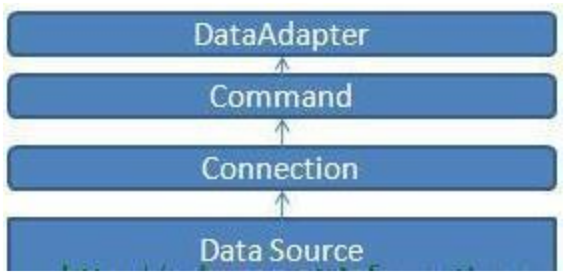
It is very useful to use with aggregate functions like **Count (*)** or **Sum ()** etc and also when compare to ExecuteReader() , ExecuteScalar() uses fewer System resources.

```
using System;

using System.Windows.Forms;
using System.Data.OleDb;

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

      private void button1_Click(object sender, EventArgs e)
      {

         string connetionString = null;
         OleDbConnection cnn ;
         OleDbCommand cmd ;
         string sql = null;


         connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Your mdb
filename;";
         sql = "Your SQL Statement Here like Select Count(*) from product";


         cnn = new OleDbConnection(connetionString); try

         {
            cnn.Open();
            cmd = new OleDbCommand(sql, cnn);

            Int32 count = Convert.ToInt32(cmd.ExecuteScalar());
            cmd.Dispose();
            cnn.Close();
```

```
                MessageBox.Show (" No of Rows " + count);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
```

5.4 C# ADO.NET DataAdapter

DataAdapter is a part of the ADO.NET Data Provider. DataAdapter provides the communication between the **Dataset** and the Datasource. We can use the DataAdapter in combination with the DataSet Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet. That is, these two objects combine to enable both data access and data manipulation capabilities.



The **DataAdapter** can perform Select, Insert, Update and Delete SQL operations in the Data Source. The Insert, Update and Delete SQL operations, we are using the continuation of the Select command perform by the DataAdapter. The **SelectCommand** property of the DataAdapter is a Command Object that retrieves data from the data source. The

InsertCommand, **UpdateCommand**, and **DeleteCommand** properties of the DataAdapter are Command objects that manage updates to the data in the data source according to modifications made to the data in the DataSet.

5.4.1 C# ADO.NET OleDbDataAdapter

The **OleDbDataAdapter** is a part of the C# ADO.NET Data Provider and it resides in the **System.Data.OleDb** namespace. The OleDbDataAdapter provides the communication between the Dataset and the OleDb Data Sources. We can use OleDbDataAdapter Object in combination with Dataset Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet.

```
OleDbDataAdapter oledbAdapter = new OleDbDataAdapter(sql, oledbCnn);
oledbAdapter.Fill(ds);
```

The **OleDbDataAdapter** Object and DataSet objects are combine to perform both Data Access and Data Manipulation operations in the OleDb Data Sources. When the user perform the SQL operations like Select , Insert etc. in the data containing in the **Dataset Object** , it won't directly affect the Database, until the user invoke the Update method in the OleDbDataAdapter.

```
using System;
using System.Windows.Forms;
using System.Data;
using System.Data.OleDb;
namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         string connetionString = null;
         OleDbConnection oledbCnn ;
         OleDbDataAdapter oledbAdapter
         ; DataSet ds = new DataSet();
         string sql = null;
         int i = 0;
         connetionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Your mdb
filename;";
         sql = "Your SQL Statement Here like Select * from product";
         oledbCnn = new OleDbConnection(connetionString);
         try
         {
            oledbCnn.Open();
            oledbAdapter = new OleDbDataAdapter(sql, oledbCnn);
```

```
                oledbAdapter.Fill(ds);
                for (i = 0; i <= ds.Tables[0].Rows.Count - 1; i++)
                {
MessageBox.Show(ds.Tables[0].Rows[i].ItemArray[0] + " -- "
+ ds.Tables[0].Rows[i].ItemArray[1]);
                }
                oledbAdapter.Dispose();
                oledbCnn.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Can not open connection ! ");
            }
        }
    }
}
```