ADO.NET

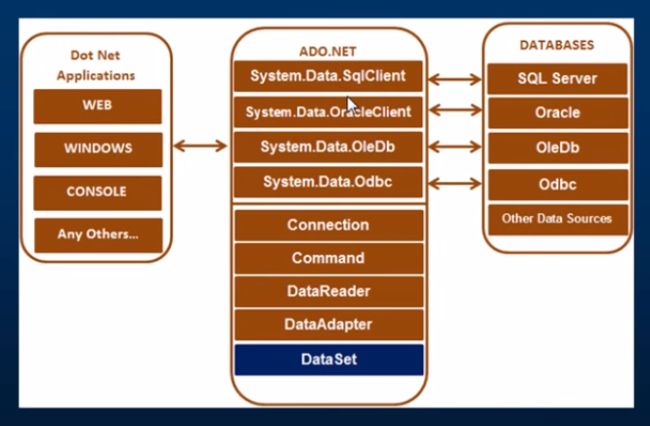
ADO.NET is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

DO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is the acronym for ActiveX Data Objects. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data.  
  
The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.

* ASP.NET Web Applications
* Console Applications
* Windows Applications.

**Various Connection Architectures**  
  
There are the following two types of connection architectures:

1. **Connected architecture:** the application remains connected with the database throughout the processing.
2. **Disconnected architecture:** the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet.

**Understanding ADO.NET and its class library  
  
**  
  
In this diagram, we can see that there are various types of applications (Web Application, Console Application, Windows Application and so on) that use ADO.NET to connect to databases (SQL Server, Oracle, OleDb, ODBC, XML files and so on).  
  
**Important Classes in ADO.NET**  
We can also observe various classes in the preceding diagram. They are:

1. Connection Class
2. Command Class
3. DataReader Class
4. DataAdaptor Class
5. DataSet.Class

**1. Connection Class**In ADO.NET, we use these connection classes to connect to the database. These connection classes also manage transactions and connection pooling.  
  
**2. Command Class**  
The Command class provides methods for storing and executing SQL statements and Stored Procedures. The following are the various commands that are executed by the Command Class.

* **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
* **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
* **ExecuteScalar:**This method only returns a single value. This kind of query returns a count of rows or a calculated value.
* **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.

**3. DataReader Class**The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.  
  
**4. DataAdapter Class**  
The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures.  
  
**5. DataSet Class**

The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

**How to Connect to a Database using ADO.NET**  
Now let us learn how to connect to a database using ADO.NET. To create a connection, you must be familiar with connection strings. A connection string is required as a parameter to SQLConnection. A ConnectionString is a string variable (not case sensitive).  
  
This contains key and value pairs, like provider, server, database, userid and word as in the following:

Server="nameof the server or IP Address of the server"  
  
Database="name of the database"  
  
userid="user name who has permission to work with database"  
  
word="the word of userid"

**Example**

**SQL Authentication**String constr="server=.;database=institute;user id=rakesh;word=abc@123";  
  
Or:  
  
String constr="data source=.;initial catalog=institute;uid=rakesh;pwd=abc@213";  
  
**Windows Authentication**  
  
String constr="server=.;database=institute;trusted\_connection=true"   
  
Or:  
  
String constr="server=.;initial catalog=institute;integrated security=true"

**How to retrieve and display data from a database**  
Procedure:

1. Create a SqlConnection object using a connection string.
2. Handle exceptions.
3. Open the connection.
4. Create a SQLCommand. To represent a SQLCommand like (select \* from studentdetails) and attach the existing connection to it. Specify the type of SQLCommand (text/storedprocedure).
5. Execute the command (use executereader).
6. Get the Result (use SqlDataReader). This is a forwardonly/readonly dataobject.
7. Close the connection
8. Process the result
9. Display the Result

INSERT

**C#**

string name = "Bibekananda";

string city = "BBSR";

string constring = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;

using (SqlConnection con = new SqlConnection(constring))

{

    using (SqlCommand cmd = new SqlCommand("INSERT INTO Persons (Name, City) VALUES (@Name, @City)", con))

    {

        cmd.CommandType = CommandType.Text;

        cmd.Parameters.AddWithValue("@Name", name);

        cmd.Parameters.AddWithValue("@City", city);

        con.Open();

        int rowsAffected = cmd.ExecuteNonQuery();

        con.Close();

    }

}

Update

string name = "hello World";

string city = "BBSR";

string constring = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;

using (SqlConnection con = new SqlConnection(constring))

{

    using (SqlCommand cmd = new SqlCommand("UPDATE Persons SET City = @City WHERE Name = @Name", con))

    {

        cmd.CommandType = CommandType.Text;

        cmd.Parameters.AddWithValue("@Name", name);

        cmd.Parameters.AddWithValue("@City", city);

        con.Open();

        int rowsAffected = cmd.ExecuteNonQuery();

        con.Close();

    }

}

DELETE

string name = "HelloWorld";

string constring = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;

using (SqlConnection con = new SqlConnection(constring))

{

    using (SqlCommand cmd = new SqlCommand("DELETE FROM Persons WHERE Name = @Name", con))

    {

        cmd.CommandType = CommandType.Text;

        cmd.Parameters.AddWithValue("@Name", name);

        con.Open();

        int rowsAffected = cmd.ExecuteNonQuery();

        con.Close();

    }

}

Print Data

private DataTable GetData()

{

    string constr = ConfigurationManager.ConnectionStrings["constr"].ConnectionString;

    using (SqlConnection con = new SqlConnection(constr))

    {

        using (SqlCommand cmd = new SqlCommand("SELECT CustomerId, Name, Country FROM Customers"))

        {

            using (SqlDataAdapter sda = new SqlDataAdapter())

            {

                cmd.Connection = con;

                sda.SelectCommand = cmd;

                using (DataTable dt = new DataTable())

                {

                    sda.Fill(dt);

                    return dt;

                }

            }

        }

    }

}

//Populating a DataTable from database.

        DataTable dt = this.GetData();

        //Building an HTML string.

        StringBuilder html = new StringBuilder();

        //Table start.

        html.Append("<table border = '1'>");

        //Building the Header row.

        html.Append("<tr>");

        foreach (DataColumn column in dt.Columns)

        {

            html.Append("<th>");

            html.Append(column.ColumnName);

            html.Append("</th>");

        }

        html.Append("</tr>");

        //Building the Data rows.

        foreach (DataRow row in dt.Rows)

        {

            html.Append("<tr>");

            foreach (DataColumn column in dt.Columns)

            {

                html.Append("<td>");

               html.Append(row[column.ColumnName]);

                html.Append("</td>");

            }

            html.Append("</tr>");

        }

        //Table end.

        html.Append("</table>");

**ADO.NET DataView Control**A major function of the DataView is to allow data binding on both Windows Forms and Web Forms. Additionally, a DataView can be customized to present a subset of data from the DataTable. This capability allows you to have two controls bound to the same DataTable, but showing different versions of the data. For example, one control may be bound to a DataView showing all of the rows in the table, while a second may be configured to display only the rows that have been deleted from the DataTable. The DataTable also has a DefaultView property which returns the default DataView for the table. For example, if you wish to create a custom view on the table, set the RowFilter on the DataView returned by the DefaultView.  
  
A DataView object serves a similar purpose to views in the SQL context. Both are ways to select a subset of the columns and rows in a table. In SQL Server, it is possible to create a view based on the result of almost any query, but a DataView in ADO.NET is limited to operation on one table only. However, unlike views in some versions of SQL Server, a DataView can be sorted into a given order. The DataViewManager and the DataViewSetting classes do allow multiple table views.   
  
A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications. Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression. A DataView provides a dynamic view of data whose content, ordering, and membership reflect changes to the underlying DataTable as they occur. This is different from the Select method of the DataTable, which returns a DataRow array from a table per a particular filter and/or sort order and whose content reflects changes to the underlying table, but whose membership and ordering remain static. The dynamic capabilities of the DataView make it ideal for data-binding applications.  
  
A DataView provides you with a dynamic view of a single set of data to which you can apply different sorting and filtering criteria, similar to the view provided by a database. However, a DataView differs significantly from a database view in that the DataView cannot be treated as a table and cannot provide a view of joined tables. You also cannot exclude columns that exist in the source table, nor can you append columns, such as computational columns, that do not exist in the source table. You use a DataViewManager to manage view settings for all the tables in a DataSet. The DataViewManager provides you with a convenient way to manage default view settings for each table. When binding a control to more than one table of a DataSet, binding to a DataViewManager is the ideal choice.