Unit 3 – ADO.NET and Database

Introduction to ADO.NET

ADO.NET stands for **ActiveX Data Objects.NET**.
ADO.NET is the main data access system and protocol that is used by all the programming languages supporting .NET.
ADO.NET is an evolution of ADO data access model that directly answers user's requirements for developing powerful applications.
ADO.NET uses disconnected data architecture.
ADO.NET is a set of objects that provide data access services.
ADO.NET provides access to data sources such as Microsoft SQL Server, Object Linking and Embedding Database (OLEDB) and XML.
ADO.NET can be used to connect to these data sources and retrieve and update data.

Architecture of ADO.NET

ADO.NET is much simpler, less dependent on the data source, more flexible and the format of data is textual instead of binary.
Because, ADO.NET is based on XML, it only required that a data provider serve the data in XML. Once you write your data access code, you only need to change a few parameters to connect to a different data source.
ADO.NET is based on a connection-less principle i.e. designed to make easy the connection limitation. We no longer have to maintain a connection constant.

The two main components of ADO.NET for accessing and manipulating data are –
1) The .NET Framework data providers
2) The DataSet.

1) .NET Framework Data Providers

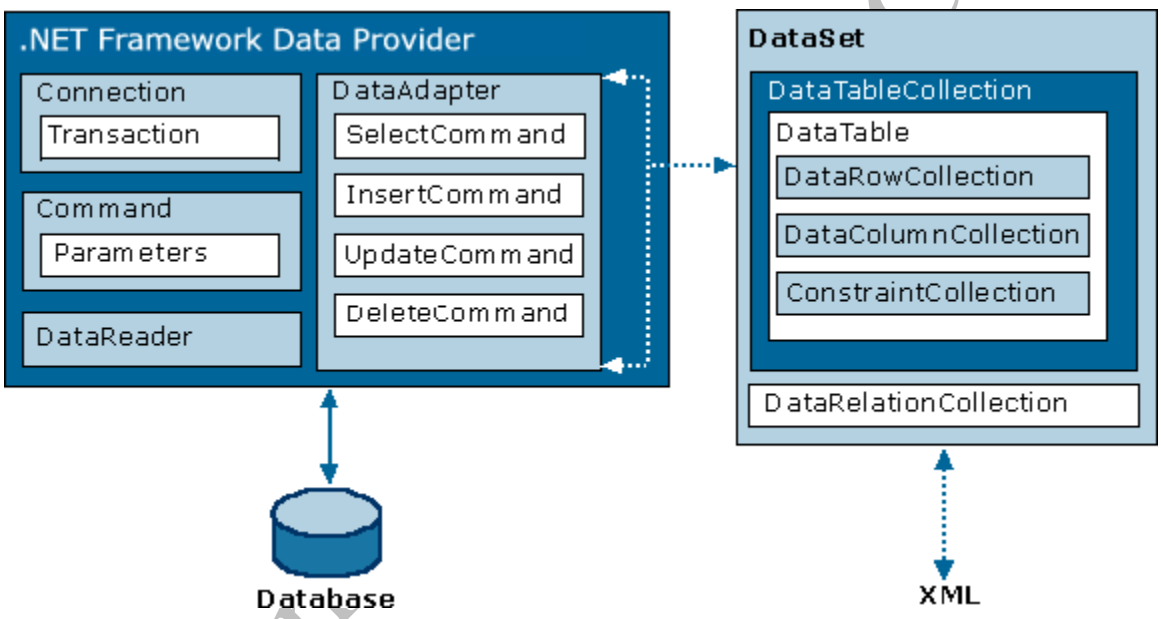
The .NET Framework Data Providers are components that have been designed for data manipulation and fast, forward-only, read-only access to data.
The core objects in ADO.NET are as follows –
1. The Connection Object - The Connection object provides connectivity to a data source.
2. The Command Object - The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
3. The DataReader Object - The DataReader provides read-only and forward-only data from the data source.
4. The DataAdapter Object - The DataAdapter provides the bridge between the DataSet object and the data source. The DataAdapter uses Command objects to execute SQL commands at the data source to both load the DataSet with data and commit changes that were made to the data in the DataSet back to the data source.

These above 4 objects form the basis for all operations on data in ADO.NET. These objects are created from following data providers -
(1) System.Data.OleDb
(2) System.Data.SqlClient
(3)System.Data.Odbc

2) The DataSet

The ADO.NET DataSet is designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The DataSet contains a collection of one or more DataTable objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the DataTable objects.

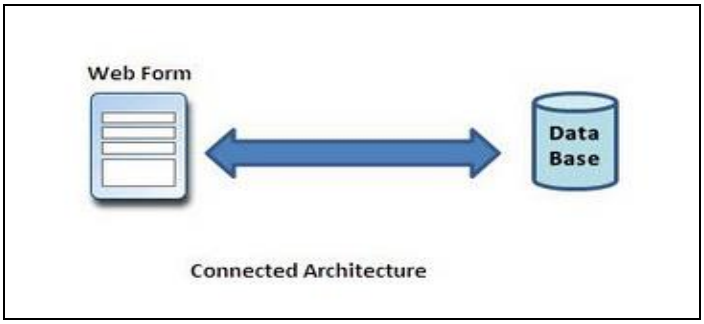
The following diagram shows the architecture of ADO.NET –



ADO.NET Architecture

Connected Architecture of ADO.NET

The architecture of ADO.NET in which connection must be opened to access the data retrieved from database is called as connected architecture.
Connected architecture is built on the classes like Connection, Command, DataReader etc.
Connected architecture is when you constantly make trips to the database for any operations like Create, Read, Update and Delete (CRUD). This creates more traffic to the database. DataReader is used in Connected Architecture since it keeps the connection open until all rows are fetched one by one.

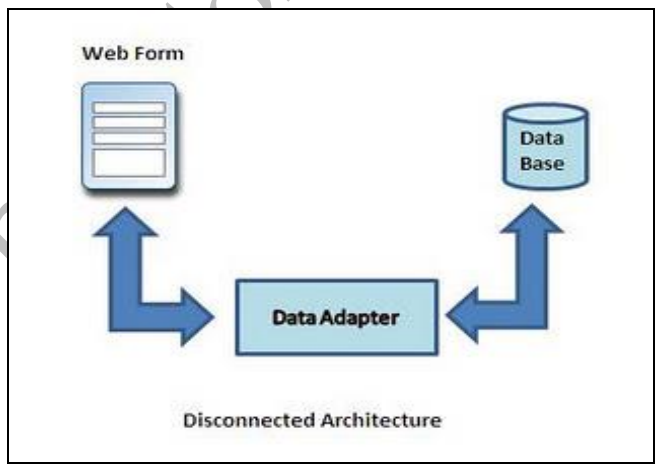


Disconnected Architecture in ADO.NET

The architecture of ADO.NET in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture. Disconnected architecture is built on classes like Connection, DataAdapter, CommandBuilder, DataSet and DataView.

Disconnected architecture is a method of retrieving a record set from the database and storing it giving you the ability to do many operations (Create, Read, Update and Delete) on the data in memory, then it can be re-synchronized with the database when reconnecting.

A method of using disconnected architecture is using a Dataset. DataSet retrieves all the records in memory at once and there is no need to keep the connection alive



Difference between Connected and Disconnected architecture

Connected Architecture	Disconnected Architecture
In connected architecture, the connection to the database has to be in	In disconnected architecture data can be accessed from database even when the

opened state the access the database	connection to database is closed
Connected architecture is built on the classes like Connection, Command, and DataReader.	Disconnected architecture is built on classes like Connection, DataAdapter, CommandBuilder and DataSet.
DataReader is used to retrieve data	DataSet is used to retrieve data
Connected method gives faster performance	Disconnected get low in speed and performance.
Connected can hold(store) the data of single table	Disconnected can hold multiple tables of data
DataReader can't persist(continue/keep it on) the data	DataSet can persist the data
It is Read only, we can't update the data.	We can update data

ADO.NET Classes

1. Connection Class

The Connection class creates connection to the database. It contains all information required to open a connection to the database.

Connections are responsible for handling communication between a data source and a .NET application, because the connection object is a part of data provider. Each data provider implements its own version.

The two data providers supported by .NET framework implements –

1) The **OledbConnection** in System.Data.Oledb namespace.
2) The **SqlConnection** in System.Data.Sqlclient namespace.

The OledbConnection uses Oledb and can be used with any database including Microsoft SQL Server. The SqlConnection goes directly to Microsoft SQL Server without going to the Oledb provider and so it is more efficient.

2. Command Class

The Command class is used to execute commands to a database across data connection and retrieve a DataReader and DataSet.

The Command class executes insert, update or delete command on a data source.

The base class for all command objects is DbCommand Class. This class is represented by 2 classes –

1) **SqlCommand**
2) **OledbCommand**

You can use SqlCommand with Microsoft SQL Server and OledbCommand with all other types of databases.

You can create a command object in one of the following two ways –

1) Use the Command constructor passing the connection name as an argument.
2) Use the CreateCommand method of connection object.

You can use the **CommandText** property of Command object to set and retrieve the SQL command being executed.
The Command Class gives 3 methods to execute commands. They are as follows –

1) **ExecuteNonQuery**() : It is used to fire (run) insert, update or delete queries to the database. It requires an open connection.
2) **ExecuteScalar**() : This method is used to run aggregate functions like min(), max(), avg(), sum() and count(). It returns only one value at a time and retrieves data from the database.
3) **ExecuteReader** : This method is used to create a DataReader object. DataReader object represents forward-only and read-only data.

3. DataReader Class

You can use the DataReader class to retrieve data from a database.
"A DataReader object represents read-only, forward-only set of data."
DataReader gives you an efficient way to access data that you only need to read once. In other words the DataReader can be used for searching records.
A DataReader object stores only one record at a time, there by giving faster performance.
There are 2 versions of DataReader object –
1) **SqlDataReader** for Microsoft SQL Server database.
2) **OledbDataReader** for all types of databases.
The SqlDataReader contains some methods that are not available to the OledbDataReader. These are GetSqlType methods that you can use to retrieve SQL Server specific data type columns from the database.
You can create object of DataReader class using the **ExecuteReader**() method of Command Class.
After you create the DataReader object, you can call its **Read**() method to get data in the rows. You must ensure that you use the **Close**() method of DataReader object before accessing any output or return parameters from a stored procedure.

4. DataAdapter Class

The DataAdapter Class is the core of ADO.NET's disconnected data architecture. It is like a bridge between the database and DataSet.
The DataAdapter provides a set of methods and properties to retrieve and save data between database and DataSet.
The DataAdapter uses **Fill**() method to fill a DataTable or DataSet with records of database.
The DataAdapter can commit the changes to the database by using **Update**() method. The DataAdapter provides 4 properties that represents SQL commands –

1) **SelectCommand**
2) **InsertCommand**
3) **UpdateCommand**
4) **DeleteCommand**

When the DataAdapter fills a DataSet, it will create the required tables and columns for the returned data.

5. DataSet Class

DataSets are the primary objects that you will work with when accessing disconnected data. They are similar in concept to groups of ADO objects like RecordSets but in ADO.NET there are many new features.

"A DataSet object is a disconnected, memory resident cache of data."

The DataSet is not the exact copy of the database. It can be considered as a local copy of some part of the database.

It is structured in a similar manner to a database as it contains 3 objects –

1) **DataTable**
2) **DataRelation**
3) **DataConstraint**

In other words, A DataSet object is a collection of one or more DataTables, DataRelations and DataConstraints objects. DataTable object can be accessed by using **Tables** property of DataSet object.

Whatever operations are made by the user, it is stored temporary in the DataSet. When the use of this DataSet is completed, changes can be made back to the central database for updation. DataSets are fully XML featured.

6. DataTable Class

DataSets are in-memory representation of relational data. DataTables contain the actual data.

DataTables can exist as a part of DataSet's "Tables" Collection property or can be created independently. The DataTable object has its own properties, methods and events. A DataTable object has 3 collections –

1) The **Columns** collection
2) The **Rows** collection
3) The **Constraint** collection

The DataTable's **PrimaryKey** property is used to set a primary key field. We can use different methods to create DataTable object as a part of DataSet –

1) Use the **Fill**() method of DataAdapter
2) Use the **Add**() method of DataSet
3) Use the "**Table Collection Editor**" that is the part of Microsoft Visual Studio.NET.

7. DataColumn Class

The DataTable's Columns collection property contains zero or more DataColumn objects that define the structure of the table.

If the DataTable is created by a DataAdapter's **Fill**() or **FillSchema**() methods, the columns collections is generated automatically.
You can create a DataColumn object independently by using one of the following **new** constructors –

No.	Constructor	Description
1	new()	Creates a new DataColumn with no column name
2	new(column name)	Creates a new DataColumn with the given name
3	new(column name, data type)	Creates a new DataColumn with given name and given data type.

8. DataRow Class

The DataTable's rows collections contains the actual data that is contained in the DataTable, in the form of zero or more DataRow objects.
The RowState property of DataRow object shows the actions that have been taken since the DataTable was created or since the **AcceptChanges**() method was called.
The following are the DataRow properties –

Properties	Description
HasErrors	Indicates whether there are any errors in the row
Item	The value of a column in the DataRow
ItemArray	The value of all columns in the DataRow represented as an array
RowState	The state of DataRow object
Table	The DataTable to which the DataRow belongs

9. DataView Class

"A DataView is a filtered and sorted set of records of a single table."
A DataView gives the same functionality as the DataTable's select method but it has many advantages because it is a different object.
DataViews can be created and configured at both design time and run time, making them easier to implement in many situations.
DataViews can be used as the DataSource for DataBound controls. You can create multiple DataViews from any given single DataTable. Every DataTable contains at least one DataView in its "DefaultDataView" property.
The following are its properties –

Properties	Description
AllowDelete	Determines whether rows in the DataView can be deleted
AllowEdit	Determines whether rows in the DataView can be changed
AllowNew	Determines whether new rows can be inserted in the DataView
Count	It gives the number of records in the DataView
Sort	It gives the expression used to sort the records of DataView
Table	It determines the name of the DataTable from which DataView is created

10. DataConstraints Class :

ADO.NET allows you to create a set of constraints on one or more columns. You can use constraints to enforce restrictions on the data in a DataTable, in order to maintain the integrity of the data. A constraint is an automatic rule, applied to a column or related columns, that determines the course of action when the value of a row is changed.
There are two kinds of constraints in ADO.NET:

1. The ForeignKeyConstraint
2. The UniqueConstraint.

Constraint	Description
ForeignKeyConstraint	Enforces a link between two DataTables within a DataSet
UniqueConstraint	Ensures that entries in a given column are unique

DataSource Controls of ASP.NET

The following are the DataSource controls :

1) The SqlDataSource Control
2) The AccessDataSource Control
3) The LINQDataSource Control
4) The ObjectDataSource Control
5) The XMLDataSource Control

The SqlDataSource Control

This control is used when the database is Microsoft SQL Server. This DataSource control loads the provider for MS SQL Server.
The wizard of this control does the following –

1) Creating provider specific connection to MS SQL Server.
2) Building SQL query or use stored procedures.
3) Getting the data from the database.

The following are the properties of SqlDataSource control –

Properties	Description
ConnectionString	Obtains or sets the connection string parameters into the connection class constructor. The string contains the server name, database name, security, userid, password.
SelectCommand	Obtains or sets the query string for retrieving records
InsertCommand	Obtains or sets the string containing insert query for adding records

The following are the methods of SqlDataSource control –

Methods	Description
DataBind()	Binds SqlDataSource to MS SQL Server
Select()	Retrieves the record from MS SQL Server database
Insert()	Inserts the records into the MS SQL Server database

Data Bound Controls

1) The GridView Control

The GridView control shows the data in grid or tabular format. It is widely used to display the required data.
The GridView can also be used to edit as well as delete the records. The records can be displayed in different pages. The records can also be displayed in sorted manner.
The following are the properties of GridView control –

Properties	Description
AllowPaging	Obtains or sets a value indicating whether the paging option is enabled.
AllowSorting	Obtains or sets a value indicating whether the sorting option is enabled.
AutoGenerateDeleteButton	Obtains or sets a value indicating whether a delete button for each row is added to the control
AutoGenerateEditButton	Obtains or sets a value indicating whether edit button for each row is added to the control
DataKeyNames	Obtains or sets the name of primary key field

The following are the methods of GridView Control –

Methods	Description
DataBind()	It binds the Data Source to the control
Sort()	It sorts the records based on given field
UpdateRow()	It updates the record at the given row index

2) The Repeater Control

The Repeater Control is a Data Bound control that allows custom layout by repeating a specified template for each item displayed in the list.
This control exists within System.Web.UI.WebControls namespace. The following are the properties of Repeater control –

Properties	Description
AlternatingItemTemplate	Obtains or sets the object that defines how the alternating items are displayed
DataMember	Obtains or sets the table name
DataSource	Obtains or sets the Data source name
EnableTheming	Obtains or sets a value indicating whether themes are applied to this control

The following are the events of the Repeater Control –

Events	Description
ItemCommand	Occurs when a button is clicked in the control
ItemCreated	Occurs when an item is created in the control
ItemDataBound	Occurs after an item in the control is data bound

DataBinding Expressions

Data-binding syntax allows you to bind control property values to data and specify values for retrieving, updating, deleting, and inserting data.
DataBinding Expressions are given within **<%# and %>** delimiters.

They use two functions –

1) **Eval**() – This function is used to define one way (read only) binding
2) **Bind**() – This function is used for two way (updatabale) binding

In addition to calling Eval() and Bind() methods to perform data binding, you can call any code within <%# and %> delimiters to run the code and return a value.
DataBinding Expressions are used when DataBind() method of a control or Page class is called. For controls such as GridView, FormView and DetailsView, the DataBinding Expressions are given automatically and you are not required to call the DataBind() method explicitly.

Example :

Consider a table named "product" with pno and pname fields. The following code shows the use of data-binding expressions with a Repeater control –

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <table>
    <asp:Repeater ID="Repeater1" runat="server"
     DataSourceID="SqlDataSource2">

    <HeaderTemplate>
    <tr style ="background-color:red">
```

```
    <th>pno</th>
    <th>pname</th>
    </tr>
    </HeaderTemplate>

    <ItemTemplate>
    <tr>
    <td>
    <%#DataBinder.Eval(Container.DataItem,"pno")%>
    </td>
    <td>
    <%#DataBinder.Eval(Container.DataItem,"pname")%>
    </td>
    </tr>
    </ItemTemplate>

    <AlternatingItemTemplate>
    <tr>
    <td style ="background-color:cyan">
        <%#DataBinder.Eval(Container.DataItem,"pno")%>
    </td>
    <td style ="background-color:cyan">
        <%#DataBinder.Eval(Container.DataItem,"pname")%>
     </td>
    </tr>
    </AlternatingItemTemplate>

    </asp:Repeater>
    </table>

    <asp:SqlDataSource ID="SqlDataSource2" runat="server"
     ConnectionString="<%$ ConnectionStrings:
     companyConnectionString %>" SelectCommand = "SELECT *
     FROM [product]">
    </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```