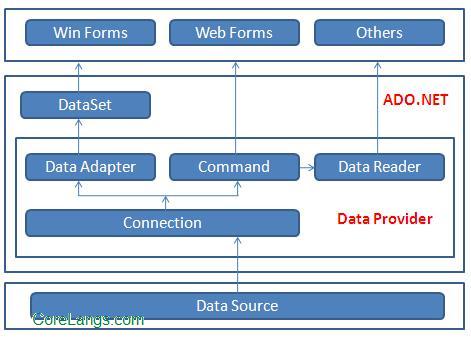
**Q.2 Explain Components of ADO.NET?**

**Ans:**



ADO.NET consists of a set of Objects that expose data access services to the .NET environment. It is a data access technology from Microsoft .Net Framework, which provides communication between relational and non-relational systems through a common set of components .System.Data namespace is the core of ADO.NET and it contains classes used by all data providers. 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.

The two key components of ADO.NET are **Data Providers** and **Dataset**. The Data Provider classes are meant to work with different kinds of data sources. They are used to perform all data-management operations on specific databases. DataSet class provides mechanisms for managing data when it is disconnected from the data source.

**A. Data Providers –**

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. SQL Server uses the SqlConnection object, OLEDB uses the OleDbConnection Object and ODBC uses OdbcConnection Object respectively. A data provider contains Connection, Command, DataAdapter, and DataReader objects. These four objects provide the functionality of Data Providers in the ADO.NET.

**1. Connection**

The Connection Object provides physical connection to the Data Source. Connection object needs the necessary information to recognize the data source and to log on to it properly, this information is provided through a connection string.

**2. Command**

The Command Object uses to perform SQL statement or stored procedure to be executed at the Data Source. The command object provides a number of Execute methods that can be used to perform the SQL queries in a variety of fashions.

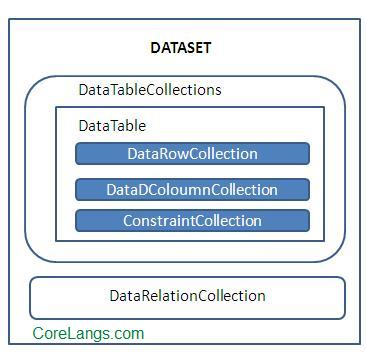
**3. DataReader**

The DataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data. DataReader requires a live connection with the databse and provides a very intelligent way of consuming all or part of the result set.

**4. DataAdapter**

**DataAdapter** Object populate a Dataset Object with results from a Data Source . It is a special class whose purpose is to bridge the gap between the disconnected Dataset objects and the physical data source.

**A. 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 contains rows, columns,primary keys, constraints, and relations with other DataTable objects. It consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. The DataAdapter Object provides a bridge between the DataSet and the Data Source.

**Q. Explain difference between Classic ADO and ADO.NET in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | Classic ADO | ADO.NET |
| 1 | Classic ADO requires active connection with the data store. | ADO.NET architecture works while the data store is disconnected. |
| 2 | Data is stored in binary format. | Data is stored in XML. |
| 3 | In ADO we have recordset . | In ADO.NET we have dataset. |
| 4 | In recordset we can only have one table. If we want to accommodate more than one table we need to do inner join and fill the recordset. | Dataset can have multiple tables. |
| 5 | XML integration is not possible. | XML integration is possible. |
| 6 | In ADO, it is sometime problematic because firewall prohibits many types of request | In ADO.net there is no such problem because XML is completely firewall-proof. |
| 7 | ADO is based on COM Technology and it used OLEDB data provider for accessing data. It has a limited number of data types which are defined by the COM standard. | ADO.NET basically designed by .NET Framework for smooth interaction of application and database, it support large and rich datatypes. |
| 8 | Using Classic ADO, we can obtain information from one table or set of tables through join. We cannot fetch records from multiple tables independently. | Dataset object of ADO.NET includes collection of DataTables wherein each DataTable will contain records fetched from a particular table. Hence multiple table records are maintained independently. |
| 9 | Classic ADO architecture includes client side cursor and server side cursor. | ADO.NET architecture doesn't include such cursors. |

**Q. Explain difference between DataReader and DataSet in asp.net.**

**ANS**:

|  |  |  |
| --- | --- | --- |
| SR No | DataReader | DataSet |
| 1 | DataReader is Connection oriented architecture. | DataSet is disconnected architecture. |
| 2 | DataReader is used to read the data from database and it is a read and forward only connection oriented architecture during fetch the data from database | DataSet is a disconnected orient architecture that means there is no need of active connections during work with datasets and it is a collection of DataTables and relations between tables. |
| 3 | Data Reader can hold one table at a time. | Data Set can hold **multiple** tables at a time. It represents a computer set of data including related tables, constraints and relationships among the tables. |
| 4 | Data Reader provides **Read only access** to data i.e. we cannot update data. | Data Set allows access on data i.e with that you have read and update access. |
| 5 | Data Reader improves application performance. Hence faster in performance | Data Set is always a bulky object that requires lot of memory space compare to Data Reader.Hence slower in performance. |
| 6 | It occupies **less** memory. | It occupies **more** memory |
| 7 | No XML storage available. If data does not need to be modified, cached or serialized consider using a reader. | Can be stored as **XML**. |
| 8 | DataReader fetches data from a single table | DataSet can fetch data from multiple tables. |
| 9 | As DataReader can have data from a single table so no relationship can be maintained | The relationship between multiple tables can be maintained in DataSet. |
| 10 | DataReader is read only so no transaction like insert, update and delete is possible | Transactions like inser,update,delete is possible in DataSet |
| 11 | Does not require lot of memory space. | DataSet is a bulky object that requires lot of memory space as compared to DataReader . |
| 12 | DataReader requires connection to be open and close manually in code | DataSet manually handles open and close connection |
| 13 | DataReader can't be serialized. | DataSet can be serialized and represented in XML so easily passed around to other tiers |
| 14 | DataReader will be the best choice where we need to show the data to the user which requires no manipulation | DataSet is best suited where there is possibility of manipulation on the data. |
| 15 | DataReader can’t be serialized so can’t be used in wcf services and web services. | Since DataSet can be serialized it, can be used in wcf services and web service that will return retrieved data. |
| 16 | DataReader can only be read once so it can be bound to a single control and requires data to be retrieved for each control. | When you need to navigate through the data multiple times then DataSet is better choice e.g.  we can fill data in multiple controls |

**Q. Explain difference between DataReader, DataSet,DataAdapter in asp.net.**

**ANS**:

**DataReader –**

DataReader is used to read the data from database and it is a read and forward only connection oriented architecture during fetch the data from database. DataReader will fetch the data very fast when compared with dataset. Generally we will use ExecuteReader object to bind data to datareader. DataReader is used to iterate through resultset that came from server and it will read one record at a time because of that memory consumption will be less and it will fetch the data very fast when compared with dataset.

**Example –**

*// This method is used to bind gridview from database*

*protected void BindGridview()*

*{*

*using (SqlConnection con = new SqlConnection("Data Source=SureshDasari;Integrated Security=true;Initial Catalog=MySampleDB"))*

*{*

*con.Open();*

*SqlCommand cmd = new SqlCommand("Select UserName,LastName,Location FROM UserInformation", con);*

*SqlDataReader dr = cmd.ExecuteReader();*

*gvUserInfo.DataSource = dr;*

*gvUserInfo.DataBind();*

*con.Close();*

*}*

*}*

**DataSet –**

DataSet is a disconnected orient architecture that means there is no need of active connections during work with datasets and it is a collection of DataTables and relations between tables. It is used to hold multiple tables with data. You can select data form tables, create views based on table and ask child rows over relations. Also DataSet provides you with rich features like saving data as XML and loading XML data.

**Example –**

*// This method is used to bind gridview from database*

*protected void BindGridview()*

*{*

*SqlConnection con = new SqlConnection("Data Source=SureshDasari;Integrated Security=true;Initial Catalog=MySampleDB");*

*con.Open();*

*SqlCommand cmd = new SqlCommand("select UserName,LastName,Location from UserInformation", con);*

*SqlDataAdapter da = new SqlDataAdapter(cmd);*

*DataSet ds = new DataSet();*

*da.Fill(ds);*

*gvUserInfo.DataSource = ds;*

*gvUserInfo.DataBind();*

*}*

**DataAdapter –**

DataAdapter will acts as a Bridge between DataSet and database. This dataadapter object is used to read the data from database and bind that data to dataset. Dataadapter is a disconnected oriented architecture. Check below sample code to see how to use DataAdapter in code.

**Example –**

*protected void BindGridview()*

*{*

*SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");*

*conn.Open();*

*SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);*

*SqlDataAdapter sda = new SqlDataAdapter(cmd);*

*DataSet ds = new DataSet();*

*da.Fill(ds);*

*gvUserInfo.DataSource = ds;*

*gvUserInfo.DataBind();*

*}*

**DataTable –**

DataTable represents a single table in the database. It has rows and columns. There is no much difference between dataset and datatable, dataset is simply the collection of datatables. A DataAdapter is used to populate DataTable from records returned from an SQL statement and also a DataTable can be created in memory and data can be added to it.

**Example –**

*protected void BindGridview()*

*{*

*SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");*

*conn.Open();*

*SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);*

*SqlDataAdapter sda = new SqlDataAdapter(cmd);*

*DataTable dt = new DataTable();*

*da.Fill(dt);*

*gridview1.DataSource = dt;*

*gvidview1.DataBind();*

*}*

**Q. What is SqlCommandBuilder in ADO.NET ?**

**ANS**:

The SqlCommandBuilder can be used to build and execute SQL queries based on the select command that you will supply. It provides the feature of reflecting the changes made to a DataSet or an instance of the SQL server data. The CommandBuilder opens the Connection associated with the DataAdapter Object and makes a round trip to the server each and every time and it's asked to construct the action queries.

SqlCommandBuilder class in [ADO.NET](http://www.dotnetuncle.com/Adonet/114_sqlcommandbuilder.aspx) provides the feature of reflecting the changes made to a DataSet or an instance of the [SQL server](http://www.dotnetuncle.com/Adonet/114_sqlcommandbuilder.aspx) data. When an instance of the SqlCommandBuilder class is created, it automatically generates Transact-SQL statements for the single table updates that occur. The object of the SqlCommandBuilder acts as a listener for RowUpdating events, whenever the DataAdapter property is set. CommandBuilder generates insert/update/delete commands for Data adapter based on select command. Automatic creation of insert/update/delete commands hinders performance. In case one knows the contents of insert/update/delete, should create those explicitly. Better to create explicit stored procedures for insert/update/delete and assign those.

**Example –**

*SqlDataAdapter adapter = new SqlDataAdapter("your sql statement", connection);*

*SqlCommandBuilder cmdBuilder = new SqlCommandBuilder(adapter);*

*adapter.Fill(ds);*

*//Here you can do any modification to the selected data*

*ds.Tables[0].Rows[i].ItemArray[2] = Convert.ToInt32 (ds.Tables[0].Rows[i].ItemArray[2]) + 100;*

*//Here save the data to the datasource*

*adapter.Update(ds.Tables[0]);*

**Q. Explain difference between Clone () and Copy () in ASP.NET.**

**ANS**:

|  |  |  |
| --- | --- | --- |
|  | Clone() | Copy() |
|  | **Clone()** will copy only structure of Data | **Copy()** will copy structure as well as data. |

**Example-**

*string[] arr1 = new string[] { "one", "two", "three","four" };*

*string[] arr2 = arr1; //copy*

*string[] arr3 = new string[] { "one", "two", "three", "four" };*

*string[] arr4 = (string[])arr3.Clone(); //clone*

|  |  |  |
| --- | --- | --- |
| SR No | DataSet.Clone() | DataSet.Copy() |
|  | Dataset.clone():  Copies the structure of the dataset, including all schemas, relations, and constraints. Does not copy any data. | Dataset.copy():  Copies both the structure and data. |

**Example -**

*DataTable dt=new DataTable();*

*dt=ds.Tables[0].****copy();***

It will copy all the data and structure to dt table

*DataTable dt=new DataTable();*

*dt=ds.Table[0].clone();*

It will create only the structure of table not data.

**Q. Explain difference between SQLCommand and SQLCommandBuilder in ADO.NET.**

**ANS**:

**SqLCommand -**

The SQLCommand Object is used to execute SQL statements and Stored Procedures against the data source. It execute all kind of SQL queries like Insert, update etc. SqlCommand is used to execute Query involving Select, Update, Delete operation and to Execute SQL Stored procedures.

**Example-**

*SqlCommand cmd = new SqlCommand("your sql statements", Connection);*

*Conn.Open();*

*cmd.ExecuteNonQuery();*

*Conn.Close();*

**SqlCommandBuilder** –

SqlCommandBuilder provides the feature of reflecting the changes made to a DataSet or an instance of the SQL server data. When an instance of the SqlCommandBuilder class is created, it automatically generates Transact-SQL statements for the single table updates that occur. The object of the SqlCommandBuilder acts as a listener for RowUpdating events, whenever the DataAdapter property is set. Sql command Builder is automatically builds commands like select,update...etc by using the data table which u can give as input to commandbuilder object. sqlcommandbuilder is used to generate queries rather than executing the queries.

**Example –**

*SqlDataAdapter adapter = new SqlDataAdapter("your sql statement", connection);*

*SqlCommandBuilder cmdBuilder = new SqlCommandBuilder(adapter);*

*adapter.Fill(ds);*

*//Here you can do any modification to the selected data*

*ds.Tables[0].Rows[i].ItemArray[2] = Convert.ToInt32 (ds.Tables[0].Rows[i].ItemArray[2]) + 100;*

*//Here save the data to the datasource*

*adapter.Update(ds.Tables[0]);*

**Q. Explain the DataAdapter.Update() and DataSet.AcceptChanges() methods.**

**ANS**:

**DataAdapter.Update() –**

The DataAdapter.Update() method calls any of the DML statements, such as the UPDATE, INSERT, or DELETE statements, as the case may be to update, insert, or delete a row in a DataSet.

Update () method is for updation of any changes made to the dataset in the database. This function checks the row version of each row with in a table. If it finds any row with added row state then that particular row is inserted else if it is modified it is updated if deleted then a delete statement is executed.  
But if Acceptchanges() is done before an update function it would not update anything to database since row state becomes unchanged

**DataAdapter. AcceptChanges () –**

The DataSet.Acceptchanges() method reflects all the changes made to the row since the last time the AcceptChanges() method was called.

Dataset maintains the row state of each row with in a table. As a dataset is loaded its row state version is unchanged . Whenever there is a modification in a particular row with in a datatable , dataset changes the row version as modified, Added or deleted based on the particular action performed on the particular row.  
AcceptChanges() method again change the Rowversion back to Unchanged.

**Q. Difference between ExecuteNonQuery,ExecuteScalar and ExecuteReader**

**ANS**:

**ExecuteNonQuery**

ExecuteNonQuery method will return number of rows effected with INSERT, DELETE or UPDATE operations. This ExecuteNonQuery method will be used only for insert, update and delete, Create, and SET statements.

**ExecuteScalar**

ExecuteScalar will return first row first column value i.e. it will return single value and ignore other values on execution of SQL Query or Stored procedure using command object. It’s very fast to retrieve single values from database.

**ExecuteReader**

Execute Reader will be used to return the set of rows, on execution of SQL Query or Stored procedure using command object. This one is forward only retrieval of records and it is used to read the table values from first to last.

**Q. What is Connection Pooling in ADO.NET?**

**ANS**:

**Connection** **pooling** is the ability of re-use your connection to the Database. This means if you enable Connection pooling in the connection object, actually you enable the re-use of the connection to more than one user.

Generally connecting to database server is a time consuming process because whenever we request to connect database first it will establish network handshaking with server and then connection string will be parsed and it will check whether given connection credentials correct or not to connect server and so on.

In our applications mostly we use one or two connection configurations and repeatedly same connection configuration will be opened and closed, automatically huge time will be consumed to open and close same database connection.

To reduce the cost of opening and closing the same connection repeatedly, ADO.NET uses an optimization technique called connection pooling.

Connection pooling is the place where it will maintain all the active database connections in one place to reduce the cost of opening and closing database connections. Whenever user send new request to Open a database connection the pooler will looks for an available connection in the pool in case if a pooled connection available then it will return pooled connection instead of opening new connection otherwise the new connection pool is created with the connection string in the connection for next time reuse.

Once we finished operations on database we need to Close the connection then that connection will be returned to the pool and its ready to be reused on the next Open call.

**Create Connection Pooling in Asp.Net**

To enable this connection pooling in asp.net we don’t need to do anything by default connection pooling is enabled in ADO.NET. Unless we manually disable the connection pooling, the pooler will optimize the connections when they are opened and closed in our application.

First time if we are opening a new connection, a distinct new connection pool is created based on the matching connection string in the connection. While creating connection pool it will check is there any connection pool created with that connection string or not by using keywords supplied in connection. In case if we send connection strings keywords in different order then it will treat it as separate connection string and same connection will be pooled separately.

**Example –**

*using (SqlConnection con = new SqlConnection("Data Source=Suresh;Integrated security=SSPI;Initial Catalog=SampleDB"))*

*{*

*con.Open();*

*// Connection Pool A will be created.*

*}*

*using (SqlConnection con = new SqlConnection("Data Source=Suresh;Integrated security=SSPI;Initial Catalog=aspdotnetDB"))*

*{*

*con.Open();*

*// Separate connection pool B will create because connection string different.*

*}*

*using (SqlConnection con = new SqlConnection("Data Source=Suresh;Initial Catalog=aspdotnetdb;Pooling=false;"))*

*{*

*con.Open();*

*// No connection pool will create because we defined Pooling = false.*

*}*

*using (SqlConnection con = new SqlConnection("Data Source=Suresh;Integrated security=SSPI;Initial Catalog=SampleDB"))*

*{*

*con.Open();*

*// This connection string matches with Connection Pool A.*

*}*

**Q. What is Object Pooling?**

**ANS**:

**Object Pooling -**

Object Pooling is something that tries to keep a pool of objects in memory to be re-used later and hence it will reduce the load of object creation to a great extent. This article will try to explain this in detail. The example is for an Employee object, but you can make it general by using Object base class.

**What does it mean?**

Object Pool is nothing but a container of objects that are ready for use. Whenever there is a request for a new object, the pool manager will take the request and it will be served by allocating an object from the pool.

**How it works?**

We are going to use Factory pattern for this purpose. We will have a factory method, which will take care about the creation of objects. Whenever there is a request for a new object, the factory method will look into the object pool (we use Queue object). If there is any object available within the allowed limit, it will return the object (value object), otherwise a new object will be created and give you back.

The below code is just an example to give you an idea, and is neither tested nor error-proof. You can modify it as you wish: up-to your creativity.

Code 1: Object Pool and Employee class.

using System;

using System.Collections;

namespace ObjectPooling

{

class Factory

{

// Maximum objects allowed!

private static int \_PoolMaxSize = 2;

// My Collection Pool

private static readonly Queue objPool = new Queue(\_PoolMaxSize);

public Employee GetEmployee()

{

Employee oEmployee;

// check from the collection pool. If exists return object else create new

if (Employee.Counter >= \_PoolMaxSize && objPool.Count>0)

{

// Retrieve from pool

oEmployee = RetrieveFromPool();

}

else

{

oEmployee = GetNewEmployee();

}

return oEmployee;

}

private Employee GetNewEmployee()

{

// Creates a new employee

Employee oEmp = new Employee();

objPool.Enqueue(oEmp);

return oEmp;

}

protected Employee RetrieveFromPool()

{

Employee oEmp;

// if there is any objects in my collection

if (objPool.Count>0)

{

oEmp = (Employee)objPool.Dequeue();

Employee.Counter--;

}

else

{

// return a new object

oEmp = new Employee();

}

return oEmp;

}

}

class Employee

{

public static int Counter = 0;

public Employee()

{

++Counter;

}

private string \_Firstname;

public string Firstname

{

get

{

return \_Firstname;

}

set

{

\_Firstname = value;

}

}

}

}

Code 2: How to use it?

private void button1\_Click(object sender, EventArgs e)

{

Factory fa = new Factory();

Employee myEmp = fa.GetEmployee();

Console.WriteLine("First object");

Employee myEmp1 = fa.GetEmployee();

Console.WriteLine("Second object")

}