

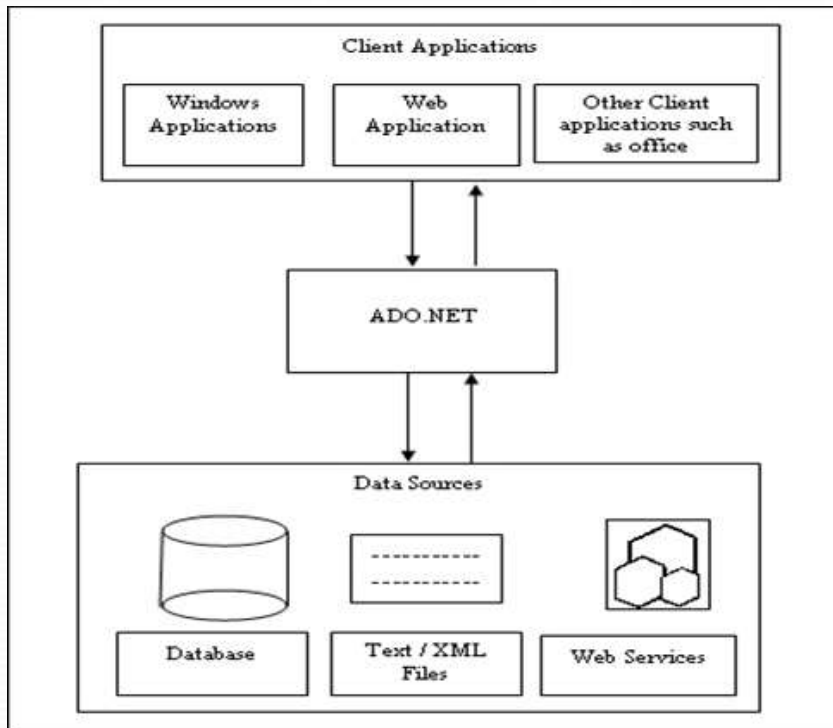
ADO . Net

What is Ado.net?

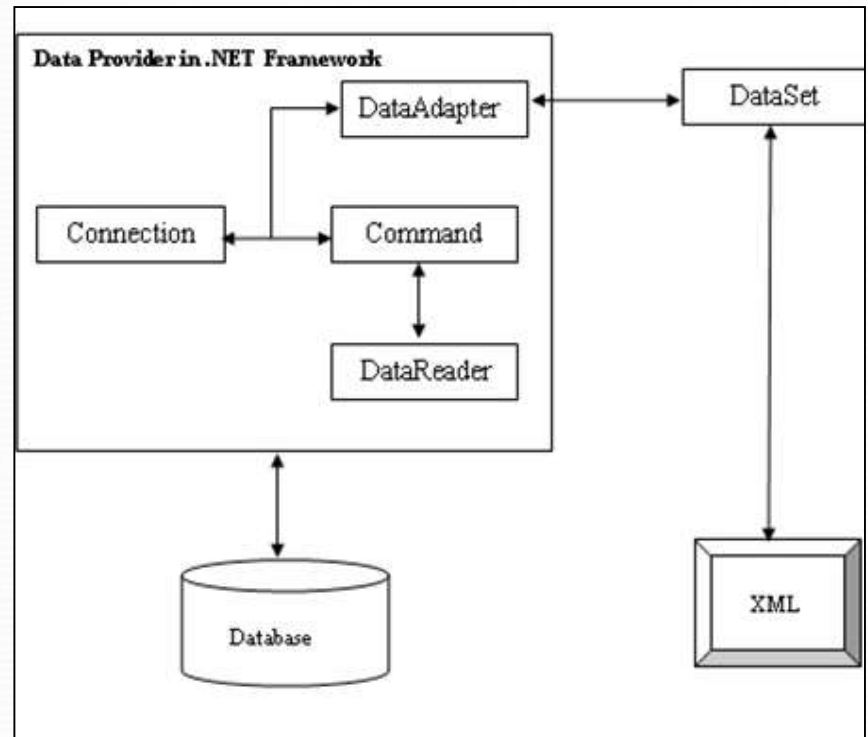
- ➔ *The full-form of ado.net is ActiveX Data Object.*
- ➔ *ADO.net is the data-access technology that enables applications to connect to data stores and manipulate data contained in them in various ways.*
- ➔ *It is based on the .NET framework and it is highly integrated with the rest of the framework class library.*
- ➔ *Ado.net is a collection of classes, interfaces, structures, enumerated type that manage data access from the relational data stores within the .net framework.*

ADO.NET

Understanding ADO.net



The Ado.net Object Model



Working with Ado.net

➔ *Many of web application development ,in many of the developers data situations involves opening data store, making request for specific data, and then populating the table in then browser with the data.*

➔ *To address the above situations Ado.net support the following classes:*


1)DataSet

2)DataReader

Working with Ado.net namespaces

➔ The six core Ado.net namespaces are;

- 1)System.Data
- 2)System.Data.Common
- 3)System.Data.OleDb
- 4)System.Data.Odbc
- 5)System.Data.SqlClient
- 6)System.Data.SqlTypes

- 
- ➔ The classes from the above namespaces are categorized as follow:
 - ➔ Disconnected : These Class provides basic structure of Ado.net includes DataTable, etc. The object of this class are capable of storing data without any dependency on specific data provide.
 - ➔ Shared : Set of base classes for data provide and shared amount all data provide. Includes DataSet, DataRow, DataTables, Constraint, etc.
 - ➔ Data Provide : These classes are meant to work with different data soures. Performs all database management operations a specific database.
 - ➔ E.g. : SqlClient data provide works only with SQL Server Database.

Type of Architecture

➔ *ADO.NET is both Connection oriented as well as Disconnection oriented*

* *Connected : Forward-only, Read-only, mode.*

*Application issues query then reads back results and processes them.
DataReader object.*

* *Disconnected :*

*Application issues query then retrieve and stores results for processing
Minimizes time connected to database DataSet object.*

➔ *Data Provider:- (Connected Environment)*

A data provider is used for connecting to a database, retrieving data, storing the data in a dataset, reading the retrieved data, and updating the database.

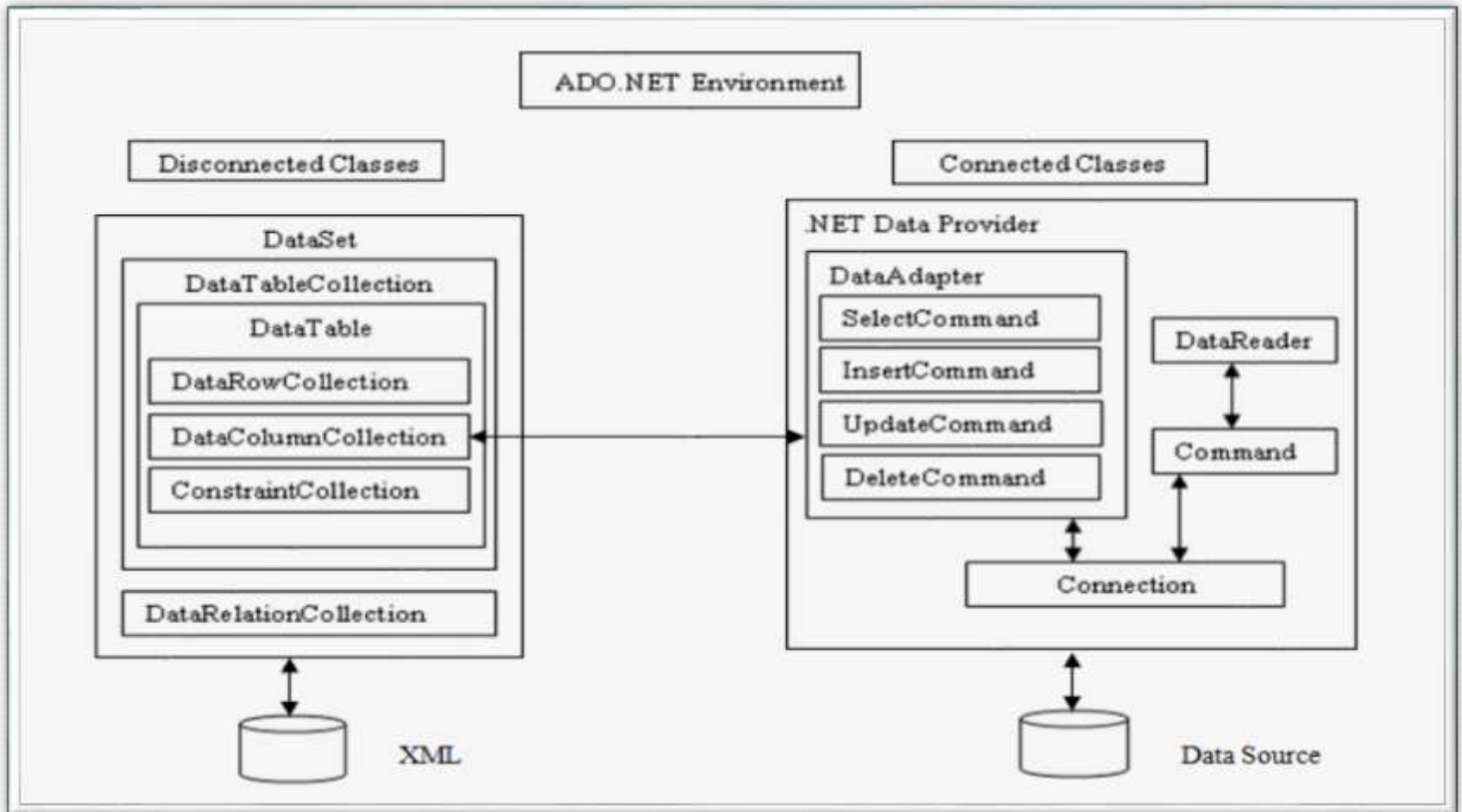
- * Connection:-This component is used to establish a connection with a data source such as database.*
- * Command:-This component is used to retrieve, insert, delete, or modify data in a data source.*
- * DataReader:-This component is used to retrieve data from a data source in a read-only and forward-only mode.*
- * DataAdapter:-This component is used to transfer data to and from database. A datareader retrieve data from a database into a dataset.*

➔ *Data Set:- (Disconnected Environment)*

The data set is a memory-based relational representation of data.

- * DataTableCollection:-It contains all table retrieved from the data source.*
- * DataRelationCollection:-It contains relationship and links between table in a dataset.*
- * DataTable:-It represent a table in the datatable collection of a dataset.*
- * DataRowCollection:-It contains all the row in a datatable.*
- * DataColumnCollection:-It contains all the column in a datatable.*

ADO.NET Environment



Data Reader

→ A data reader(*SqlDataReader*, etc.,) is a simple and fastest way of selecting some data from a data source but also least capable.

- * You can't directly instantiate a data reader object. An instance is returned from appropriate *Command* object after having call *ExecuteReader()* method.

- * This data reader is forward-only, read-only connected cursor. You can only travels the records in one direction. Database connection used is kept open until the data reader has been closed.

- * The *DataReader* object requires a live connection to the database. So when you are done with *DataReader* object, be sure to call *close()*. If not connection stays alive.

- * When *DataReader* object is closed (via an explicit call to *close*) or the object being garbage collected, the underlying connection(*con*) may also be closed depending on which of the *ExecuteReader()* method is called.

- * If you call *ExecuteReader()* and pass *CommandBehaviour.CloseConnection*

you can force the connection to be closed when reader is closed.

➔ For Data Reader object you have the following methods:

- * *Read():* Reads record by record in the data reader.
- * *Close():* Closes the SqlDataReader connection
- * *Dispose():* Disposes the data Reader object.

➔ Some of the important properties of Data Reader object are:

- * *Connection:* Gets the SqlConnection associated with the SqlDataReader.
- * *HasRows :* Gets the value that indicates whether the SqlDataReader contains one or more rows.
- * *IsClosed:* Retrieves a boolean value that indicates whether the specified SqlDataReader instance has been closed.

Data Adapter

- ➔ *The Data Adapter classes (Eg:SqlDataAdapter,..) bridges the gap between the disconnected DataTable objects and the physical data source.*
- ➔ *The SqlDataAdapter provides two way data transfer mechanisms:*
 - * *It is capable of executing SELECT statement on a data source and transferring the result set into a DataTable object.*
 - * *It is also capable of executing the standard INSERT, UPDATE and DELETE statements and extracting the input data from DataTable object.*

- ➔ The commonly used properties of SqlDataAdapter class are:
 - * Select Command: Gets or sets an object of type SqlCommand. This command is automatically executed to fill a Data Table with the result set.
 - * Insert Command: Gets or sets an object of type SqlCommand.
 - * Update Command: Gets or sets an object of type SqlCommand.
 - * Delete Command: Gets or sets an object of type SqlCommand.
- ➔ This SqlDataAdapter class also provide a method called Fill().
 - * Calling the Fill() method automatically execute the command provided by the Select Command property and retrieve the result set and copies it to DataTable.

Using Parameter

➔ If you requires SQL Statements that required you to configure using parameters then you need to instantiate Parameter (i.e., SqlParameter,..) object and providing it with necessary information such as name, value, type, size, direction , etc.,

➔ The following are properties of the SqlParameter class:-

* ParameterName, SqlDbType, Size, Direction, SourceColoumn, value

➔ Coding Snipped in using parameters are:

```
for e.g. : SqlCommand myCmd = new SqlCommand();  
myCmd.CommandText = "SELECT * FROM CUSTOMERS WHERE  
CITY=@CITY";  
SqlParameter parmCity = new SqlParameter();  
parmCity.ParameterName="@CITY";  
parmCity.SqlDbType=SqlDbType.varchar;  
parmCity.Direction=ParameterDirection.input;  
parmCity.Value="Ahmedabad"  
mycmd.Parameter.Add(parmCity);
```

DataSet

- ➔ *The DataSet class has been designed as an offline container of data.*
 - * Data held within the DataSet is not necessary to come from database. It can be recorded from database. It can be recorded from CSV file, XML source, etc.*
 - * A DataSet class contains a set of Data Table classes and links between tables.*
- for e.g. :*

```
foreach(DataRow row in ds.Tables["Customers"].rows);  
Console.WriteLine("{0} and {1}",row[0],row[1]);
```
- ➔ *Each DataTable class consists DataColumnn, DataRow, Constraints and ExtendedProperties classes.*
- ➔ *.csv=Comma Separated Values*

➔ Here DataColumn object defines the properties of a column within DataTable. These properties includes data type (Int32, Char, DateTime, etc.), read-only. You can also assign a name for column.

for e.g.:

```
DataColumn customerID=new DataColumn("CustomerID",typeof(int));
```

```
customerID.AllowDBNull=false;
```

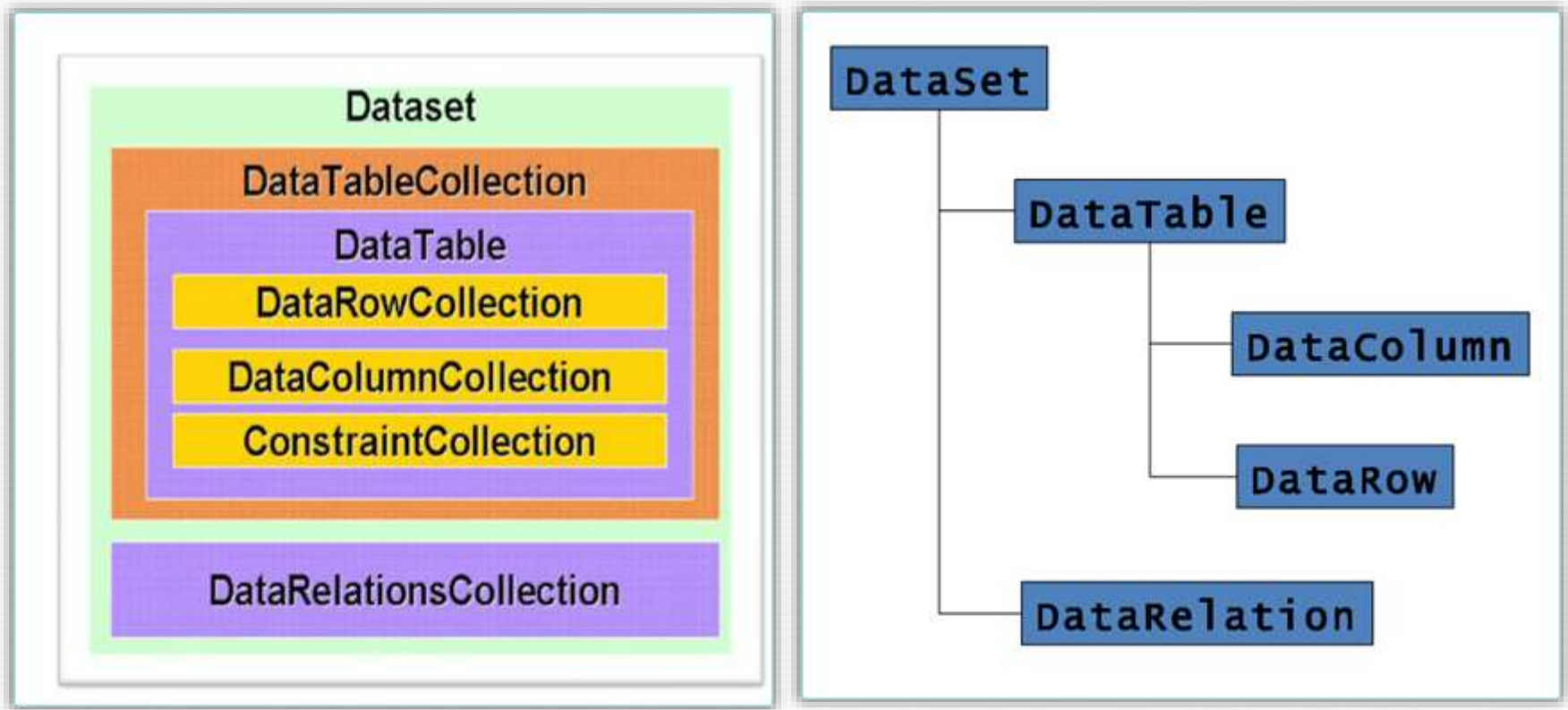
```
customerID.ReadOnly=false;
```

```
customerID.AutoIncrement=true;
```

```
customerID.Unique=true;
```

➔The actual data within a table is accessed via a DataRow object. One of the interesting aspect of DataRow is that it is versioned. This permits you to receive various values for a given column in a particular row.

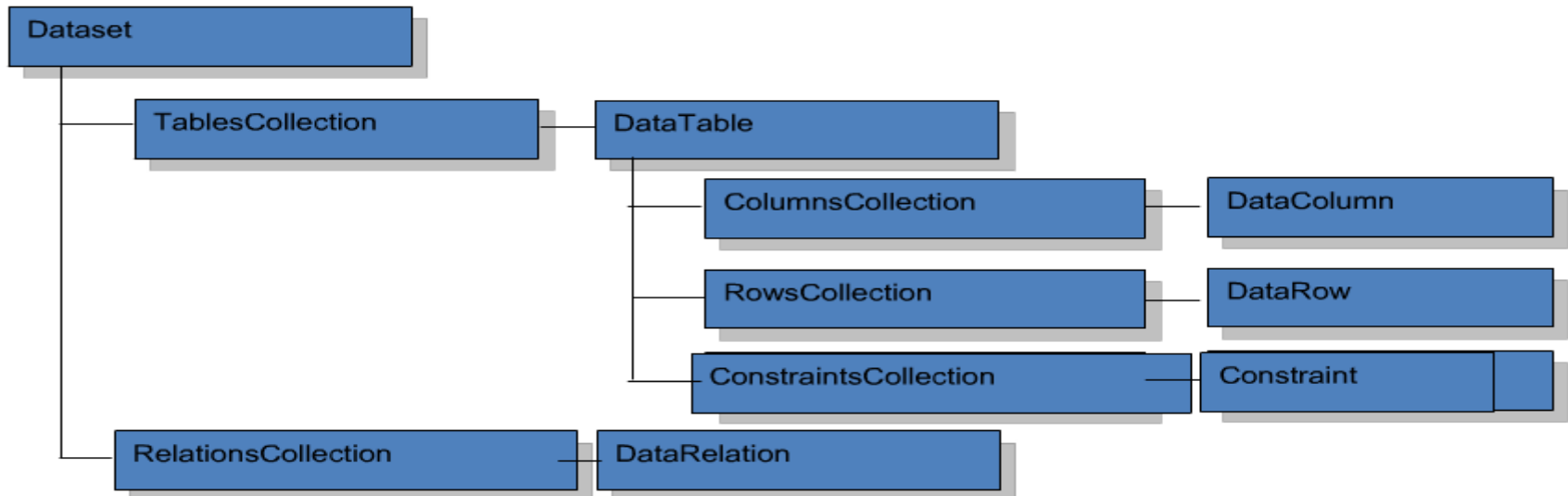
DataSet



DataSet Architecture

➔ You can create *DataSet* in any one of the following methods :

- * *Using Server Explorer*
- * *Using DataAdapter*
- * *Programmatically*
- * *Using Xml*



➔ Coding snippet of using DataSet :
for e.g. :

```
SqlConnection con = new SqlConnection(...);  
SqlCommand cmd = new SqlCommand(...);  
SqlDataAdapter da = new SqlDataAdapter (cmd, con);  
DataSet ds = new DataSet();  
da.Fill(ds, "MyTable");
```

➔ Better guideline regarding DataSet is :

- * Don't use DataSet object unless your project required to work with multiple DataTable object and you need a container for those object.
- * If you are working with single table then just use DataTable object rather than DataSet.

➔ You can create schemas for a DataTable in three ways.

- * Let the runtime do it for you.
- * Write code to create tables.
- * Use the XML schema generator.

➔ The *DataAdapter* class is used to place the data in a *DataSet* class. This Adapter class issues the SQL clause and fills a table within the *DataSet*.

* for e.g. :

```
SqlDataAdapter da = new SqlDataAdapter(strCmd , con);
```

```
DataSet ds = new DataSet();
```

```
da.Fill(ds, "Customers"); //Populating DataSet with DataAdapter.
```

➔ The *DataSet* class has been designed to establish relationships between data tables with ease.

➔ At present you can apply the *ForeignKeyConstraint* and *UniqueConstraint* on a columns in a *DataSet*.

➔ You can also set *Update & Delete* constraints on a columns that have constraint relationships.

➔ *These rules are :*

** Cascade, None, Set Default, Set Null.*

for e.g. :

```
DataTable dt = Get DataTable(); //It return DataTable  
DataColumn[] pk = new DataColumn[1];  
pk[0] = dt. Columns["StudentID"];  
dt.PrimaryKey = pk;
```

➔ *The Constrains are enforced within a DataSet class if the EnforcedConstraints property of the DataSet is true.*

Data Providers Classes

➔ *In .NET framework data provide is used for connecting to database, executing commands are retrieving results. By default the following data provides are included in .NET framework :*

- * Data Provider for SQL Server*
- * Data Provider for OLEDB*
- * Data Provider for ODBC*
- * Data Provider for Oracle*

➔ *In .NET to access the data set of classes defined in the namespaces System.Data and other namespaces defined within this namespaces.*

➔ *Each data provider contains the following objects:*

- * *Connection (for e.g. : SqlConnection, Oracleconnection, Odbcconnection, etc.)*
- * *Command, CommandBuilder (for e.g. : SqlConnection, SqlConnectionBuilder, etc.)*
- * *DataAdapter (for e.g. : SqlDataAdapter, etc.)*
- * *DataReader (for e.g. : SqlDataReader, etc)*
- * *Parameter (for e.g. : SqlParameter, etc)*
- * *Transaction (e.g. : SqlTransaction, etc)*

➔ *Each Provider use different namespaces. Their namespaces are in respective :*

- * *System.Data.SqlClient*
- * *System.Data.OleDb*
- * *System.Data.Odbc*
- * *System.Data.OracleClient*

➔ In addition to above namespaces few more important namespaces in ADO.NET are:

- * *System.Data*
- * *System.Data.Command*
- * *System.Data.SqlTypes*

➔ ADO.NET consists number of classes that are used regardless you are using the SQL Server Classes or OLEDB classes.

➔ The *System.Data* namespaces consists the following classes:

- * *DataSet*
- * *DataTables*
- * *DataRow*
- * *DataColumn*
- * *DataRelation*
- * *Constraint*

Classes in Data Providers

- ➔ *Each data provider supports the following types of core objects to communicate with database.*
- ➔ *The Connection object provides connectivity to a data source.*
- ➔ *The Command object enable access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.*
- ➔ *The DataReader provides a high performance stream of the data from the data source.*
- ➔ *Finally, 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 renconcile change made to the data in the data in the DataSet back to the data source.*

Connection Object

- ➔ *The different Connection classes are in different data provider models.*
 - * *for e.g. : SqlConnection, OleDbConnection, OracleConnection, etc.*
- ➔ *The properties of SqlConnection class are:*
 - * *ConnectionString, DataSource, Database, State.*
 - * *In above properties DataSource, Database, State are read-only mode.*
- ➔ *You can pass necessary connection string information to the SqlConnection in a string format or store in "web.config" file.*

➔ You can store database connection parameters in configuration file under <ConnectionString> section. Here you can specify name of connection, connection string parameter and provider for this connection.

** for e.g. usage of SqlConnection:*

```
SqlConnection con = new SqlConnection("Data  
Source=FARZADFWADIA-PC;Initial Catalog=FarzadWadia;  
Integrated Security=True");
```

```
con.Open();//Use connection object.
```

```
con.Close(); //you need to call this explicitly, because .net doesn't  
implicitly release the connection when they fall out of scope.
```

Command Object

➔ In ADO.NET a command is a string of the containing SQL statements that is to be issued to the database. It can also be a stored procedure or the name of the table which return all columns and rows.

➔ A Command can be constructed by using the instance of SqlCommand.

for e.g. :

```
SqlCommand cmd = new SqlCommand("Select * from  
Employee",con);
```

➔ After building the command object you need to execute with any one of the methods:

- * *ExecuteNonQuery()* : Executes the command but doesn't return any output. Only it will return no of rows effected.
- * *ExecuteReader()* : Executes the command and returns a type SqlDataReader.
- * *ExecuteRow()* : Executes the command and returns SqlRecord, (Single row)
- * *ExecuteScalar()* : Executes the command and returns a single values.
- * *ExecuteXmlReader()* : Executes the command and returns XmlReader.

➔ *Some of the important properties of Command object are :*

- * CommandText : Assign SQL statement or stored procedure name,*
- * CommandTimeout : Specifies no of second to wait for executing a command*
- * CommandType : Possible values are Text, Stored Procedure, Table direct.*
- * Connection : Gets or Sets SqlConnection object.*

➔ *Calling a stored procedure with a Command object
for e.g. :*

```
SqlCommand cmd = new SqlCommand("MyStoredProc", con);  
cmd.CommandType = CommandType.StoredProcedure;  
cmd.Parameters.Add(new SqlParameter(...));  
cmd.Parameters[o].Value = val;  
cmd.ExecuteNonQuery();
```

ADO.NET Programming

➔ ADO.NET Programming involves the following steps:

- * Create a Connection object and provide connection string Information.*
- * Create a Command object and provide details of SQL command that is to be executed.*
- * Command can be inline SQL Text command, Stored Procedure or direct table access. Also provide required parameters if needed.*
- * If your command doesn't return result set then execute the command simply by calling one of Execute method on Command object.*
- * If your command object returns a result set then you can store the result set in DataReader or DataAdapter object.*
- * To load the data from DataReader to DataTable just call Load() method on DataTable.*

```
SqlDataReader myReader=cmd.ExecuteReader(...);  
DataTable myTable=new DataTable();  
myTable.Load(myReader);
```

- * In reverse if you want a reader from DataTable then use:
DataTableReader myReader=myTable.CreateDataReader();*
- * If you want to retain the result set for future use without maintaining the connection to the database then use DataAdapter object. This DataAdapter object is used to fill DataSet or DataTable objects.*
- * If you don't want to retain the result set, rather simply process the command in a swift fashion, you use Command object needs a live connection to database and it works as forward-only read-only cursor.*



THANK YOU