Database Programming With ADO.NET

- ADO.NET – introduction and applications
- ADO.NET – architecture (connected and disconnected)
- Database connectivity using ADO.NET
- Use of Data sources, Server Explorer and working with DataSet
- Populating data in a DataGridView

<u>ADO.NET – introduction and applications</u>

- ADO.NET is a part of the Microsoft .Net Framework.
- The full form of ADO.Net is ActiveX® Data Objects.
- ADO.Net has the ability to separate data access mechanisms, data manipulation mechanisms and data connectivity mechanisms.
- ADO.Net is a set of classes that allow application to read and write information in databases.
- ADO.Net can be used by any .Net Language.
- It's concept. It's not a programming language.
- ADO.Net introduces the concept of disconnected architecture.
- We need to add System.Data namespace for work with ADO.Net
- It's a next version of ActiveX Data Objects (ADO) technology which was used in VB6.0.

ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.

ADO.NET separates data access from data manipulation into discrete components that can be used separately. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO.NET DataSet object in order to be exposed to the user in an ad hoc manner, combined with data from multiple sources, or passed between tiers. The **DataSet** object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. For sample code that connects to a database, retrieves data from it, and then displays that data in a console window.

ADO.NET provides functionality to developers who write managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO). We recommend that you use ADO.NET, not ADO, for accessing data in your .NET applications

Comparison between ADO and ADO.NET

	ADO	ADO.NET
Data access	ADO used connected data usage. (Connection-Oriented Models)	ADO.NET used disconnected data environment. (Disconnected Models)
XML Support	In ADO XML Support is limited.	In ADO.NET XML robust Support.
Format of data transferring	ADO used technology to access data and is COM – Based.	ADO.NET uses xml for transmitting data to and from your database and web application.
Data provider	In ADO disconnected data provide by Record Set.	In ADO.NET disconnected data provide by DataSet and DataAdpter.
Tables	In ADO, Record Set, is like a single table or query result.	In ADO.NET DataSet, can contain multiple tables.
Client connection	In ADO client connection model is very poor. Client application needs to be connected to data-sever while working on the data.	In ADO.NET client disconnected as soon as the data is fetched or processed. DataSet is always disconnected.

Advantages of ADO.NET

Interoperability

ADO.NET applications can take advantage of the flexibility and broad acceptance of XML. Because XML is the format for transmitting datasets across the network, any component that can read the XML format can process data. In fact, the receiving component need not be an ADO.NET component at all: The transmitting component can simply transmit the dataset to its destination without regard to how the receiving component is implemented. The destination component might be a Visual Studio application or any other application implemented with any tool whatsoever. The only requirement is that the receiving component be able to read XML. As an industry standard, XML was designed with exactly this kind of interoperability in mind.

Maintainability

In the life of a deployed system, modest changes are possible, but substantial, architectural changes are rarely attempted because they are so difficult. That is unfortunate, because in a natural course of events, such substantial changes can become necessary. For example, as a deployed application becomes popular with users, the increased performance load might require architectural changes. As the performance load on a deployed application server grows, system resources can become scarce and response time or throughput can suffer. Faced with this problem, software architects can choose to divide the server's business-logic processing and user-interface processing onto separate tiers on separate machines. In effect, the application server tier is replaced with two tiers, alleviating the shortage of system resources.

The problem is not designing a three-tiered application. Rather, it is increasing the number of tiers after an application is deployed. If the original application is implemented in ADO.NET using datasets, this transformation is made easier. Remember, when you replace a single tier with two tiers, you arrange for those two tiers to trade information. Because the tiers can transmit data through XML-formatted datasets, the communication is relatively easy.

Programmability

ADO.NET data components in Visual Studio encapsulate data access functionality in various ways that help you program more quickly and with fewer mistakes. For example, data commands abstract the task of building and executing SQL statements or stored procedures.

Similarly, ADO.NET data classes generated by the designer tools result in typed datasets. This in turn allows you to access data through typed programming. The code for the typed dataset is easier to read. It is also easier to write, because statement completion is provided.

Performance

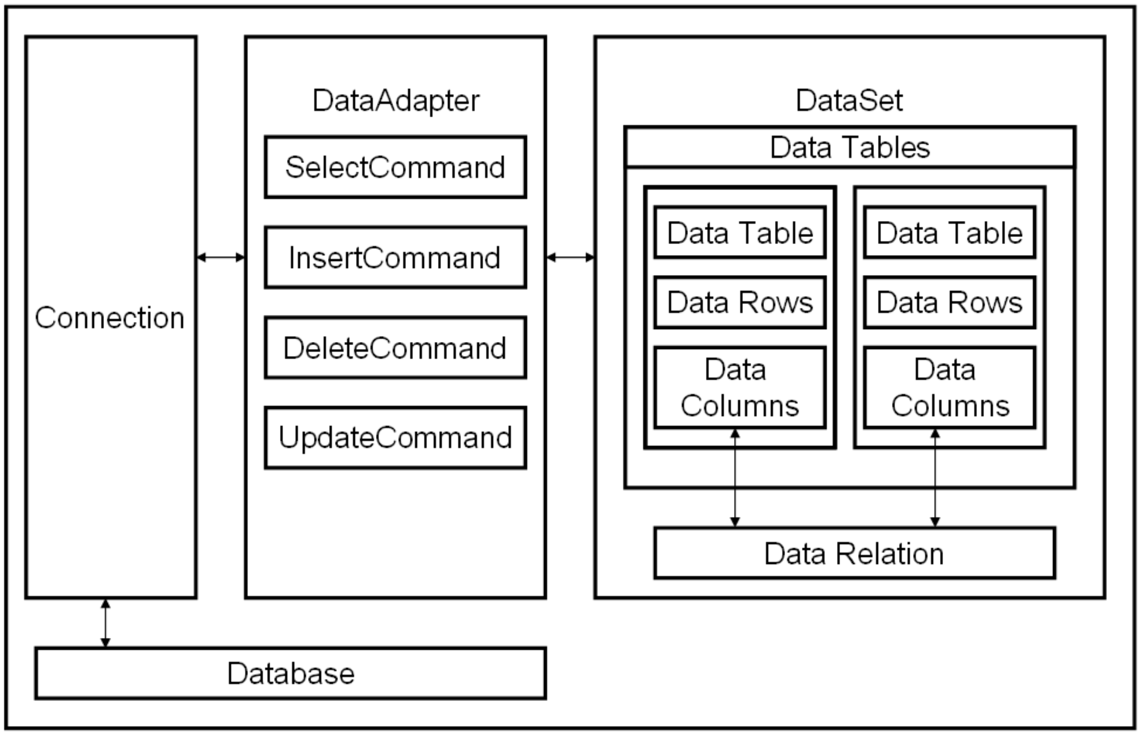
For disconnected applications, ADO.NET datasets offer performance advantages over ADO disconnected recordsets. When using COM marshalling to transmit a disconnected recordset among tiers, a significant processing cost can result from converting the values in the recordset to data types recognized by COM. In ADO.NET, such data-type conversion is not necessary.

Scalability

Because the Web can vastly increase the demands on your data, scalability has become critical. Internet applications have a limitless supply of potential users. Although an application might serve a dozen users well, it might not serve hundreds —or hundreds of thousands — equally well. An application that consumes resources such as database locks and database connections will not serve high numbers of users well, because the user demand for those limited resources will eventually exceed their supply.

ADO.NET accommodates scalability by encouraging programmers to conserve limited resources. Because any ADO.NET application employs disconnected access to data, it does not retain database locks or active database connections for long durations.

ADO.NET Architecture



Data processing has traditionally relied primarily on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

The two main components of ADO.NET 3.0 for accessing and manipulating data are the .NET Framework data providers and the DataSet.

.NET Framework Data Providers

The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The **Connection** object provides connectivity to a data source. The **Command** object enables 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 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 reconcile changes that were made to the data in the **DataSet** back to the data source.

The DataSet

The ADO.NET **DataSet** is explicitly 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 illustrates the relationship between a .NET Framework data provider and a **DataSet**.

Selecting a DataReader or a DataSet

When you decide whether your application should use a **DataReader** or a **DataSet**, consider the type of functionality that your application requires. Use a **DataSet** to do the following:

- Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the **DataReader** is the better choice.
- Remote data between tiers or from an XML Web service.
- Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
- Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the **DataSet**, you can improve the performance of your application by using the **DataReader** to return your data in a forward-only, read-only manner. Although the **DataAdapter** uses the **DataReader** to fill the contents of a **DataSet**, by using the **DataReader**, you can boost performance because you will save memory that would be consumed by the **DataSet**, and avoid the processing that is required to create and fill the contents of the **DataSet**.

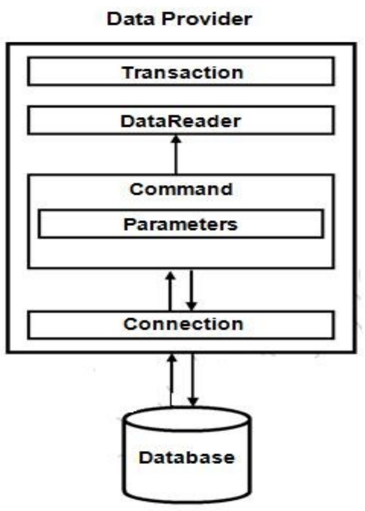
Connected Architecture of ADO.NET

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 was built on the classes connection, command, datareader and transaction.

Connection : in connected architecture also the purpose of connection is to just establish aconnection to database and it self will not transfer any data.

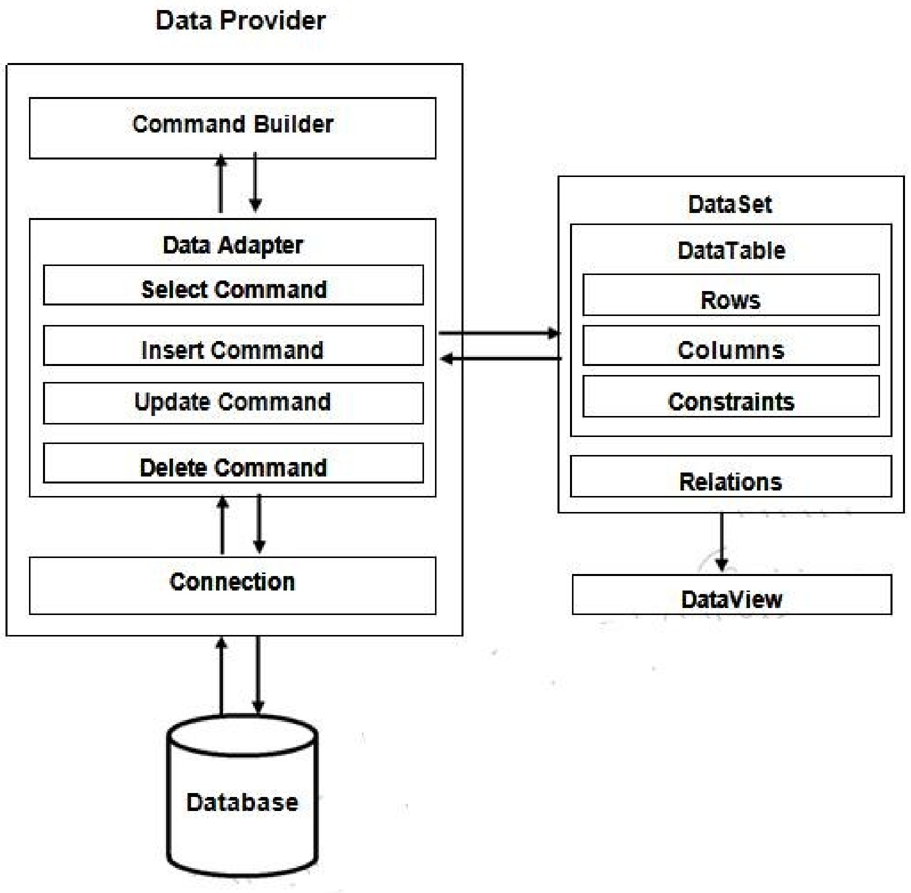
DataReader : DataReader is used to store the data retrieved by command object and make it available for .net application. Data in DataReader is read only and within the DataReader you can navigate only in forward direction and it also only one record at a time.

To access one by one record from the DataReader, call **Read()** method of the DataReader whose return type is **bool.** When the next record was successfully read, the Read() method will return true and otherwise returns false.



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 of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.



Connection : Connection object is used to establish a connection to database and connectionit self will not transfer any data.

DataAdapter : DataAdapter is used to transfer the data between database and dataset. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.

CommandBuilder : by default dataadapter contains only the select command and it doesn'tcontain insert, update and delete commands. To create insert, update and delete commands for the dataadapter, commandbuilder is used. It is used only to create these commands for the dataadapter and has no other purpose.

DataSet : Dataset is used to store the data retrieved from database by dataadapter and make it available for .net application.

To fill data in to dataset **fill()** method of dataadapter is used and has the following syntax.

Da.Fill(Ds,"TableName");

When fill method was called, dataadapter will open a connection to database, executes select command, stores the data retrieved by select command in to dataset and immediately closes the connection.

As connection to database was closed, any changes to the data in dataset will not be directly sent to the database and will be made only in the dataset. To send changes made to data in dataset to the database, **Update()** method of the dataadapter is used that has the following syntax.

Da.Update(Ds,"Tablename");

When Update method was called, dataadapter will again open the connection to database, executes insert, update and delete commands to send changes in dataset to database and immediately closes the connection. As connection is opened only when it is required and will be automatically closed when it was not required, this architecture is called disconnected architecture.

A dataset can contain data in multiple tables.

DataView : DataView is a view of table available in DataSet. It is used to find a record, sort the records and filter the records. By using dataview, you can also perform insert, update and delete as in case of a DataSet.

DataReader is **Connected Architecture** since it keeps the connection open until all rows are fetched one by one

DataSet is **DisConnected Architecture** since all the records are brought at once and there is no need to keep the connection alive

Difference between Connected and disconnected architecture

Connected (DataReader)	**Disconnected (DataSet)**
It is connection oriented.	It is disconnection oriented.
Connected methods gives faster performance	Disconnected get low in speed and performance.
connected can hold the data of single table	disconnected can hold multiple tables of data
connected you need to use a read only forward only data reader	disconnected you cannot
Data Reader can't persist the data	Data Set can persist the data
It is Read only, we can't update the data.	We can update data

ADO.NET Components

There are two major components of ADO.NET:

1. DataSet
2. DataProvider

DataSet : represents either an entire database or a subset of database. It can contain tables and relationships between those tables.

Data Provider is a collection of components like Connection, Command, DataReader, DataAdapter objects and handles communication with a physical data store and the dataset.

Data Provider

- The data provider is responsible for providing and maintaining the connection to the database.
- It is a set of classes that can be used for communicating with database, and holding/manipulating data
- The DataProvider connects to the data source on behalf of ADO.NET.
- The data source can be Microsoft SQL Server or Oracle database and OLEDB data provider.
- The data provider components are specific to data source.

The following lists the .NET Framework data providers that are included in the .NET Framework.

For SQL Server

- Imports System.Data.SqlClient namespace.
- It provides data access for Microsoft SQL Server.

For OLEDB

- Imports System.Data.OleDb namespace.
- It provides data sources exposed using OLEDB.
- We can use OLEDB for connect Microsoft Access

For ODBC

- Imports System.Data.Odbc namespace.
- It provides data sources exposed using ODBC

For Oracle

- Imports System.Data.OracleClient namespace.
- It provides data access for oracle.

Data Provider's common set of classes for all DataSource.

1. Connection
2. Command
3. DataAdapter
4. DataReader

Conncetion

- It establishes or connects a connection to the data source.
- In SQL Server the connection can establish using SqlConnection object.

Command

- Fires SQL commands or perform some action on the data source, such as insert, update, delete.
- In SQL Server command can fires using SqlCommand object.

DataAdapter

- It's a bride between Data source and DataSet object for transferring data.
- In SQL Server the Data Adapter can create using SqlDataAdapter object

DataReader

- Used when large list of results one record at a time.
- It reads records in a read-only, forward-only mode.

- In SQL Server the datareader can create using SqlDataReader object

Use of Server Explorer

Server Explorer/Database Explorer is the server management console for Visual Studio. Use this window to open data connections and to log on to servers and explore their system services.

Use **Server Explorer/Database Explorer** to view and retrieve information from all of the databases you are connected to. You can do the following:

- List database tables, views, stored procedures, and functions
- Expand individual tables to list their columns and triggers
- Right-click a table to perform actions, such as showing the table's data or viewing the table's definition, from its shortcut menu.

To access **Server Explorer/Database Explorer**, choose **Server Explorer** or **Database Explorer** on the **View** menu. To make the **Server Explorer/Database Explorer** window close automatically when not in use, choose **Auto Hide** on the **Window** menu.

Namespaces

System.Data	The System.Data namespace provides access to classes that represent the ADO.NET architecture. ADO.NET lets you build components that efficiently manage data from multiple data sources. Consists of the classes that constitute the ADO.NET architecture, which is the primary data access method for managed applications. The ADO.NET architecture enables you to build components that efficiently manage data from multiple data sources. ADO.NET also provides the tools to request, update, and reconcile data in distributed applications.
System.Data.OleDb	The System.Data.OleDb namespace is the.NET Framework Data Provider for OLE DB. Classes that make up the .NET Framework Data Provider for OLE DB-compatible data sources. These classes allow you to connect to an OLE DB data source, execute commands against the source, and read the results.
System.Data.SqlClient	The System.Data.SqlClient namespace is the.NET Framework Data Provider for SQL Server. Classes that make up the .NET Framework Data Provider for SQL Server, which allows you to connect to SQL Server 7.0, execute commands, and

	read results. The **System.Data.SqlClient** namespace is similar to the **System.Data.OleDb** namespace, but is optimized for access to SQL Server 7.0 and later.
System.Data.SqlTypes	The System.Data.SqlTypes namespace provides classes for native data types in SQL Server. These classes provide a safer, faster alternative to the data types provided by the .NET Framework common language runtime (CLR). Using the classes in this namespace helps prevent type conversion errors caused by loss of precision. Because other data types are converted to and from **SqlTypes** behind the scenes, explicitly creating and using objects within this namespace also yields faster code.

Connection Object

A primary function of any database application is connecting to a data source and retrieving the data that it contains. The .NET Framework data providers of ADO.NET serve as a bridge between an application and a data source, allowing you to execute commands as well as to retrieve data by using a **DataReader** or a **DataAdapter**. A key function of any database application is the ability to update the data that is stored in the database

In ADO.NET you use a **Connection** object to connect to a specific data source by supplying necessary information in a connection string. The **Connection** object you use depends on the type of data source.

Each .NET Framework data provider included with the .NET Framework has a **Connection** object: the .NET Framework Data Provider for OLE DB includes an OleDbConnection object, the .NET Framework Data Provider for SQL Server includes a SqlConnection object, the .NET Framework Data Provider for ODBC includes an OdbcConnection object, and the .NET Framework Data Provider for Oracle includes an OracleConnection object.

Steps to connect database:

1. Create Database.
2. Start writing connection string in VB.Net.
3. Set the provider.
4. Specify the data source.

Connection object have ConnectionString property. Depends on the parameter specified in the Connection String, ADO.Net Connection Object connect to the specified Database.

Properties :

ConnectionString	: Gets/sets the connection string to open a database.
Database	: Gets the name of the database to open
DataSource	: Gets the name of the SQL Server to use.
State	: Gets the connection's current state

Methods :

Close	: Closes the connection to the data provider.
Open	: Opens a database connection.

Command Object :

- It's depended on Connection Object.
- Command objects are used to execute commands to a database across a data connection.
- The Command object in ADO.NET executes SQL statements and stored procedures against the data source specified in the connection object.
- The Command objects has a property called Command Text, which contains a String (Query) value that represents the command that will be executed in the Data Source.

There are many ways to initialize Command object:

For Example

```
Dim con As New SqlConnection
Dim cmd As SqlCommand
Dim str1 As String

Str1 = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')
con.Open()
cmd = New SqlCommand(str1, con)
cmd.ExecuteNonQuery()
con.close()
```

OR

```
Dim con As New SqlConnection
Dim cmd As SqlCommand

con.Open()
cmd.Connection = con
cmd.CommandText = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')"
cmd.ExecuteNonQuery()
con.close()
```

Properties

Property	Meaning
CommandText	Gets/sets the SQL statement (or stored procedure) for this command to execute.
CommandType	Gets/sets the type of the CommandText property (typically set to text for SQL).
Connection	Gets/sets the SqlConnection to use.

Parameters	Gets the command parameters.

Methods

Methods	Meaning
ExecuteNonQuery	Executes a non-row returning SQL statement, returning the number of affected rows.
ExecuteReader	Creates a data reader using the command
ExecuteScalar	Executes the command and returns the value in the first column in the first row of the result.

DataAdapter Object :

The **SqlDataAdapter**, serves as a bridge between a DataSet and SQL Server for retrieving and saving data. The **SqlDataAdapter** provides this bridge by mapping Fill, 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, using the appropriate Transact-SQL statements against the data source. The update is performed on a by-row basis. For every inserted, modified, and deleted row, the Update method determines the type of change that has been performed on it (**Insert**, **Update**, or **Delete**). Depending on the type of change, the **Insert**, **Update**, or **Delete** command template executes to propagate the modified row to the data source.

When the **SqlDataAdapter** fills a DataSet, it creates the necessary tables and columns for the returned data if they do not already exist. **SqlDataAdapter** is used in conjunction with SqlConnection and SqlCommand to increase performance when connecting to a SQL Server database.

The **SqlDataAdapter** also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand properties to facilitate the loading and updating of data.

When an instance of **SqlDataAdapter** is created, the read/write properties are set to initial values. For a list of these values, see the **SqlDataAdapter** constructor.

The InsertCommand, DeleteCommand, and UpdateCommand are generic templates that are automatically filled with individual values from every modified row through the parameters mechanism.

For every column that you propagate to the data source on Update, a parameter should be added to the **InsertCommand**, **UpdateCommand**, or **DeleteCommand**. The SourceColumn property of the DbParameter object should be set to the name of the column. This setting indicates that the value of the parameter is not set manually, but is taken from the particular column in the currently processed row.

Properties

Name	Meaning
DeleteCommand	Gets or sets a Transact-SQL statement or stored procedure to delete records from the data set.
InsertCommand	Gets or sets a Transact-SQL statement or stored procedure to insert new records into the data source.

SelectCommand	Gets or sets a Transact-SQL statement or stored procedure used to select records in the data source.
TableMappings	Gets a collection that provides the master mapping between a source table and a DataTable. (Inherited from DataAdapter.)
UpdateCommand	Gets or sets a Transact-SQL statement or stored procedure used to update records in the data source.

Methods	**Meaning**
Fill	Adds or updates rows in a data set to match those in the data source. Creates a table named "Table" by default
Update	Updates the data store by calling the INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the given dataset.

For Example :

```
Dim da As SqlDataAdapter
Dim ds As New DataSet
str1 = "select * from stud"
da = New SqlDataAdapter(str1, con)
da.Fill(ds)
```

<u>DataSet Object</u>

- ADO.NET caches data locally on the client and store that data into DataSet.
- The dataset is a disconnected, I-memory representation of data.
- It’s not exact copy the database.
- It can be considered as a local copy of the some portions of the database.
- The DataSet contains a collection of one or more DataTable objects made up of rows and columns of data.
- Tables can be identified in DataSet using DataSet’s Tables property.
- It also contains primary key, foreign key, constraint and relation information about the data in the DataTable objects.
- DataSet are also fully XML-featured.
- Whaterer operations are made by the user it is stored temporary in the DataSet, when the use of this DataSet is finished, changes can be made back to the central database for updating.
- DataSet doesn’t “know” where the data it contains came from and if fact it can contain data from multiple sources.
- The DataSet is populated DataAdapter’s Fill method.

The **DataSet** is a major component of the ADO.NET architecture. The **DataSet** consists of a collection of DataTable objects that you can relate to each other with Data Relation objects. You can also enforce data integrity in the **DataSet** by using the UniqueConstraint and ForeignKeyConstraint objects. For further details about working with **DataSet** objects.

Whereas DataTable objects contain the data, the DataRelationCollection allows you to navigate though the table hierarchy. The tables are contained in a DataTableCollection accessed through the Tables property. When accessing DataTable objects, note that they are conditionally case sensitive.

For example, if one DataTable is named "mydatatable" and another is named "Mydatatable", a string used to search for one of the tables is regarded as case sensitive. However, if "mydatatable" exists and "Mydatatable" does not, the search string is regarded as case insensitive. For more information about working with DataTable objects.

A **DataSet** can read and write data and schema as XML documents. The data and schema can then be transported across HTTP and used by any application, on any platform that is XML-enabled. You can save the schema as an XML schema with the WriteXmlSchema method, and both schema and data can be saved using the WriteXml method. To read an XML document that includes both schema and data, use the ReadXml method.

In a typical multiple-tier implementation, the steps for creating and refreshing a **DataSet**, and in turn, updating the original data are to:

1. Build and fill each DataTable in a **DataSet** with data from a data source using a DataAdapter.
2. Change the data in individual DataTable objects by adding, updating, or deleting DataRow objects.
3. Invoke the GetChanges method to create a second **DataSet** that features only the changes to the data.
4. Call the Update method of the DataAdapter, passing the second **DataSet** as an argument.
5. Invoke the Merge method to merge the changes from the second **DataSet** into the first.
6. Invoke the AcceptChanges on the **DataSet**. Alternatively, invoke RejectChanges to cancel the changes.

Properties

Name	**Description**
Relations	Get the collection of relations that link tables and allow navigation from parent tables to child tables.
Tables	Gets the collection of tables contained in the DataSet.

For Example :

```
Dim da As SqlDataAdapter
Dim ds As New DataSet
str1 = "select * from stud"
da = New SqlDataAdapter(str1, con)
da.Fill(ds)
DataGridView1.DataSource = ds.Tables(0)
```

DataReader Object

Provides a way of reading a forward-only stream of rows from a SQL Server database

To create a **SqlDataReader**, you must call the ExecuteReader method of the SqlCommand object, instead of directly using a constructor.

While the **SqlDataReader** is being used, the associated SqlConnection is busy serving the **SqlDataReader**, and no other operations can be performed on the SqlConnection other than closing it. This is the case until the Close method of the **SqlDataReader** is called. For example, you cannot retrieve output parameters until after you call Close.

Changes made to a result set by another process or thread while data is being read may be visible to the user of the **SqlDataReader**. However, the precise behavior is timing dependent.

IsClosed and RecordsAffected are the only properties that you can call after the **SqlDataReader** is closed. Although the RecordsAffected property may be accessed while the **SqlDataReader** exists, always call Close before returning the value of RecordsAffected to guarantee an accurate return value.

Properties

Name	**Description**
Connection	Gets the SqlConnection associated with the SqlDataReader.
IsClosed	Retrieves a Boolean value that indicates whether the specified SqlDataReader instance has been closed. (Overrides DbDataReader.IsClosed.)
Item	Overloaded. Gets the value of a column in its native format.

Methods

Name	**Description**
Close	Closes the SqlDataReader object. (Overrides DbDataReader.Close().)
Read	Advances the SqlDataReader to the next record. (Overrides DbDataReader.Read().)

```
Dim reader As SqlDataReader
sql = " Select * from stud"
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        reader = cmd.ExecuteReader()
        While reader.Read()
                MsgBox(reader.Item(0) & "  -  " & reader.Item(1))
        End While
        reader.Close()
        cmd.Dispose()
        con.Close()
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

<u>DataGridView Control</u>

The **DataGridView** control provides a powerful and flexible way to display data in a tabular format. You can use the **DataGridView** control to show read-only views of a small amount of data, or you can scale it to show editable views of very large sets of data.

You can extend the **DataGridView** control in a number of ways to build custom behaviors into your applications. For example, you can programmatically specify your own sorting algorithms, and you can create your own types of cells. You can easily customize the appearance of the **DataGridView** control by choosing among several properties. Many types of data stores can be used as a data source, or the **DataGridView** control can operate with no data source bound to it.

The **DataGridView** control provides a customizable table for displaying data. The **DataGridView** class allows customization of cells, rows, columns, and borders through the use of properties such as DefaultCellStyle, ColumnHeadersDefaultCellStyle, CellBorderStyle, and GridColor.

You can use a **DataGridView** control to display data with or without an underlying data source. Without specifying a data source, you can create columns and rows that contain data and add them directly to the **DataGridView** using the Rows and Columns properties. You can also use the Rows collection to access DataGridViewRow objects and the DataGridViewRow.Cells property to read or write cell values directly. The Item indexer also provides direct access to cells.

As an alternative to populating the control manually, you can set the DataSource and DataMember properties to bind the **DataGridView** to a data source and automatically populate it with data. For more information, see Displaying Data in the Windows Forms DataGridView Control.

When working with very large amounts of data, you can set the VirtualMode property to **true** to display a subset of the available data. Virtual mode requires the implementation of a data cache from which the **DataGridView** control is populated. For more information, see Data Display Modes in the Windows Forms DataGridView Control.

For additional information about the features available in the **DataGridView** control, see DataGridView Control (Windows Forms). The following table provides direct links to common tasks.

<u>Explain the steps to bind DataGridView.</u>

Step : 1	Take New VB.NET Project
Step : 2	Add database in your project (Named : dbemp.mdf) Add New Table in database file (Named : emp) from the Server Explorer Window

	Add some records in that table
Step : 3	Now add emp table to the DBEmpDataSet.xsd file
Step : 4	Add DataGridView control on the form
Step : 5	Select the Choose Data Source In that click on Other Data Source Project Data Source DBEmpDataSet Emp ← Table Name
Step : 6	At the end, Run the project

Binding Data Grids

As we've already seen, you can use data grids to display entire data tables. To bind a data grid to a table, you can set the data grid's **DataSource** property (usually to a dataset, such as **dsDataSet**) and **DataMember** property (usually to text naming a table like "authors"). At run time, you can set both of these properties at once with the built-in data grid method **SetDataBinding** (data grids are the only controls that have this method):

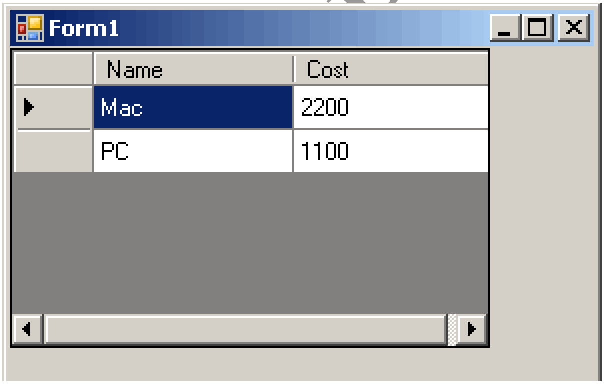
```
DataGrid1.SetDataBinding(dsDataSet, "authors")
```

You can use these data sources with the data grid's **DataSource** property:

- **DataTable** objects
- **DataView** objects
- **DataSet** objects
- **DataViewManager** objects
- single dimension arrays

To determine which cell was selected by the user, use the **CurrentCell** property. You can change the value of any cell using the **Item** property, which can take either the row or column indexes of the cell. And you can use the **CurrentCell Changed** event to determine when the user selects another cell.

Example



	Name	Cost
▶	Mac	2200
	PC	1100

First, you should add a DataGridView collection to your Windows Forms application by double-clicking on the control name in the Visual Studio designer panel. After you add the control, you can add the Load event on the form, which you can create from the Form's event pane.

```
Public Class Form1
   Private Sub Form1_Load(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) Handles MyBase.Load
         DataGridView1.DataSource = GetDataTable()
   End Sub

   Private Function GetDataTable() As DataTable
         Return New DataTable()
   End Function
End Class
```

This event handler is executed when the program starts up and when the DataGridView control is displayed. The Form1_Load autogenerated subroutine calls into the GetDataTable function, which would return a DataTable from your database in SQL Server.

Assigning the DataSource property on DataGridView does not copy any data, but instead allows the DataGridView to read in the DataTable and display all its contents on the screen in grid form. This is often the most efficient way to populate DataGridView.

Explain the steps to bind the application with the Database in ADO .net using Binding Navigator control

Step : 1	Take New VB.NET Project
Step : 2	Add two labels and two textbox on the form1
Step : 3	Add database in your project (Named : dbemp.mdf) Add New Table in database file (Named : emp) from the Server Explorer Window Add some records in that table
Step : 4	Now add emp table to the DBEmpDataSet.xsd file
Step : 5	After creating table Add BindingNavigator1 and BindingSource1 Control to the form

Step : 6	Now set the property of BindingSource1 control
	DataSource = DBEmpDataSet
	DateMember = emp

Step : 7	Now Set the property of BindingNavigator1 control Right click on BindingNavigator1 control and select Edit Items… Set BindingSource = BindingSource1 Then ok
Step : 8	Now Set the property of each textbox control Select TextBox1 and set the DataBinding property Click on Advanced property. The Formatting and Advanced Binding Window is appeared. In this window set Binding = BindingSource1 – empno Similarly, set all the textbox control
Step : 9	At the end, Run the project

Properties of BindingSource Control:

1) Datasource: Gets or sets the data source that the connector binds to.

   Syntax: instance.DataSource = *value*

2) Datamember: Gets or sets the specific list in the data source to which the connector currently binds to.

Syntax: instance.DataMember = *value*

Properties of BindingNavigator Control:

1) BindingSource: Gets or sets the System.Windows.Forms.BindingSource component that is the source of data.

   Syntax: instance.BindingSource = *value*

Explain use of "ExecuteScaler" , "ExecuteNonQuery" and "ExecuteReader" method in detail.

ExecuteNonQuery

ExecuteNonQuery() is one of the most frequently used method in SqlCommand Object and is used for executing statements that do not return result set. ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures and Views perform by ExecuteNonQuery() . Also Data Manipulation tasks like Insert , Update and Delete perform by ExecuteNonQuery().

The following example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

```
sql = "Insert into stud values('"+ TextBox1.Text +"', '"+ TextBox1.Text +"')
Try
        con.Open()
        cmd = New SqlCommand(Sql, con)
        cmd.ExecuteNonQuery()
        cmd.Dispose()
        con.Close()
        MsgBox(" ExecuteNonQuery in SqlCommand executed !!")
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

ExecuteReader

ExecuteReader() in SqlCommand Object send the SQL statements to Connection Object and populate a SqlDataReader Object based on the SQL statement. When the ExecuteReader method in SqlCommand Object execute, it instantiate a SqlClient.SqlDataReader Object.

The SqlDataReader Object is a stream-based, forward-only, read-only retrieval of query results from the Data Source, which do not update the data. The SqlDataReader cannot be created directly from code, they created only by calling the ExecuteReader method of a Command Object.

```
Dim reader As SqlDataReader
sql = " Select * from stud"
```

```
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        reader = cmd.ExecuteReader()
        While reader.Read()
                MsgBox(reader.Item(0) & "  -  " & reader.Item(1))
        End While
        reader.Close()
        cmd.Dispose()
        con.Close()
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

ExecuteScaler

ExecuteScalar() in SqlCommand Object is used for get a single value from Database after its execution. It executes SQL statements or Stored Procedure and returned a scalar value on first column of first row in the Result Set. If the Result Set contains more than one columns or rows , it takes only the first column of first row, all other values will ignore. If the Result Set is empty it will return a Null reference.

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

```
sql = " Select Count(*) from stud"
Try
        con.Open()
        cmd = New SqlCommand(sql, con)
        Dim count As Int32 = Convert.ToInt32(cmd.ExecuteScalar())
        cmd.Dispose()
        con.Close()
        MsgBox(" No. of Rows " & count)
Catch ex As Exception
        MsgBox("Can not open connection ! ")
End Try
```

What are the four common SQL commands used to retrieve and modify data in a SQL Database? Also explain each of them.

Four Common SQL Commands :

1. Select
2. Insert
3. Update
4. Delete

Select

Retrives data from one or more tables or views.
Systax :

```
Select * from TableName
Select Columnname1, Columnname2 from TableName
Select * from TableName Where <Condition>
Select * from TableName Order by Columnnane
```

Example :

```
Select * from stud
Select Sno,Sname from stud
Select * from stud where city = “Anand”
Select * from stud order by total
```

Insert

Create a new row and inserts values into specified columns

Syntax:

```
Insert into TableName values(Value1,Value2,…)
```

Example :

```
Insert into stud values (1,’ABC’)
```

Update

Changes the values of individual columns in one or more existing rows in a table.

Syntax :

```
Update TableName set Columnname = value, ….
Update TableName set Columnname = value where <condition>
```

Example :

```
Update stud set sname = ‘xyz’ where sno = 1
```

Delete

Delete one or more rows from a table

Syntax :

```
Delete from TableName
Delete from TableName where <condition>
```

Example :

```
Delete from stud
Delete from stud where sno = 5
```

Unit-4

	MCQ	
1	___________ is disconnected, in-memory representation of data.	
	a. DataReader	b. **DataSet**
	c. DataAdapter	d. DataCommand
2	What is the major component of connected data architecture?	
	a. **DataReader**	b. DataSet
	c. DataAdapter	d. DataCommand
3	______________ method is used to populate DataSet.	
	a. Populate	b. **Fill**
	c.Open	d. Store
4	______________ method returns result set by way of DataReader object.	
	a. ExecuteDataReader	b) ExecuteScalar
	c. ExecuteNonQuery	d) **ExecuteReader**
5	For insert, update, and delete SQL commands, ______________ method is used.	
	a. ExecuteDataReader	b) ExecuteScalar
	c. **ExecuteNonQuery**	d) ExecuteReader
6	To check whether connection is open or not, ___________ property is used.	
	a. ConnectionStatus	b) **State**
	c. ConnectionState	d) Status
7	To use DataSet __________________ namespace needs to be included.	
	a. System.Data.SqlClient	b) System.Sql
	c. System.Oledb	d) **System.Data**
8	__________________ object provide connection to the database.	
	a. Command	b) **Connection**
	c. DataReader	d) DataAdapter
9	In a connection string, __________________ represents name of the database.	
	a. DataSource	b) Initial Database
	c. **Initial Catalog**	d) Catalog Database
10	If we are not returning any records from the database which method is used	
	a. ExecuteReader ()	b). ExecuteScalar ()
	c. ExecuteXmlReader()	**d). ExecuteNonQuery()**
11	Which namespace defines the interfaces implemented by a .NET Data Provider?	
	a. System.Data.	b. System.Data.OleDb and System.Data.SqlClient.
	c. System.Data.SqlTypes.	d. System.Data.Provider.
12	What are the two Data Providers provided with the .NET Framework?	
	a. OLEDB Data Provider and XML Data Provider.	b. SQL Server Data Provider and XML Data Provider.
	c. OLEDB Data Provider and SQL Server Data Provider.	d. SQL Server Data Provider and Oracle Data Provider.
13	DataSet changes are saved to the database using the:	
	a. Save method.	**b. Update method.**
	c. Fill method.	d. None of the above.

14	DataSets are loaded from the database using the:	
	a. Load method.	b. Read method.
	c. Fill method.	d. None of the above.
15	Sorting supports:	
	a. Ascending order.	b. Descending order.
	c. Ascending and descending order.	d. None of the above.
16	SQL SELECT statements are sent to a database using:	
	a. SqlConnection and its ExecuteReader method.	b. SqlCommand object and its ExecuteNonQuery method.
	c. SqlCommand object and its ExecuteReader method.	d. SqlParameter object and its Execute method.
17	SQL INSERT statements are sent to a database using:	
	a. SqlConnection and its ExecuteReader method.	**b. SqlCommand object and its ExecuteNonQuery method.**
	c. SqlCommand object and its ExecuteReader method.	d. SqlParameter object and its Execute method.
18	SQL UPDATE statements are sent to a database using:	
	a. SqlConnection and its ExecuteReader method.	**b. SqlCommand object and its ExecuteNonQuery method.**
	c. SqlCommand object and its ExecuteReader method.	d. SqlParameter object and its Execute method.
19	SQL DELETE statements are sent to a database using:	
	a. SqlConnection and its ExecuteReader method.	**b. SqlCommand object and its ExecuteNonQuery method.**
	c. SqlCommand object and its ExecuteReader method.	d. SqlParameter object and its Execute method.
20	Stored procedures are invoked using:	
	a. SqlConnection and its ExecuteReader method.	b. SqlCommand object and its ExecuteNonQuery method.
	c. SqlCommand object and its ExecuteReader method.	d. SqlParameter object and its Execute method.
21	Retrieving a single value from a returned row involves using:	
	a. SqlConnection and its ExecuteReader method.	b. SqlCommand object and its ExecuteNonQuery method.
	c. SqlCommand object and its ExecuteScalar method.	d. SqlParameter object and its Execute method.

Short Questions		
1	What are the features of ADO.Net?	
2	What are the applications of ADO .net?	
3	Explain the use of server explorer in data access in .net.	
4	Mention the namespace that is used to include .NET Data Provider for SQL server in .NET code.	
5	Mention different types of data providers available in ADO .NET Framework.	
6	Which namespaces are required to enable the use of databases in ADO .net?	
7	What is the use of the Connection object?	
8	What are the usages of the Command object in ADO.NET?	
9	What is difference between DataSet and DataReader?	
10	What is Dataset object?	
11	What is connection string? Explain in brief.	
12	Which properties are used to bind a DataGridView control?	
13	Mention different types of data providers available in ADO .NET Framework.	
14	Explain DataGrid control.	
15	Explain any one of following namespaces • System.Data • System.Data.SqlClient • System.Data.OleDB • System.Data.SqlTypes	
16	Explain any one of following *.NET Data Provider Main Classes* • Connection • Command • DataReader • DataAdapter	
Long Questions:--		
1	Explain the connected architecture of ADO.NET in brief.	
2	Describe the disconnected architecture of ADO.NET's data access model.	
3	Explain major ADO .net objects.	
4	Explain the steps to bind the application with the Database in ADO .net.	
5	Explain the step, how can we retrieve data in DataSet?	
6	Explain public methods of SqlCommand objects	
7	What are the four common SQL commands used to retrieve and modify data in a SQL Database? Also explain each of them	
8	Explain use of “ExecuteScalar” , “ExecuteNonQuery” and “ExecuteReader” method in detail	