

ADO.NET

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

All the ADO.NET classes are located into **System.Data.dll** and integrated with XML classes located into **System.Xml.dll**.

ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.

.NET Framework Data Providers

These are the components that are designed for data manipulation and fast access to data. It provides various objects such as **Connection, Command, DataReader and DataAdapter** that are used to perform database operations. We will have a detailed discussion about **Data Providers** in new topic.

The DataSet

It is used to access data independently from any data resource. DataSet contains a collection of one or more DataTable objects of data. The following diagram shows the relationship between .NET Framework data provider and DataSet.

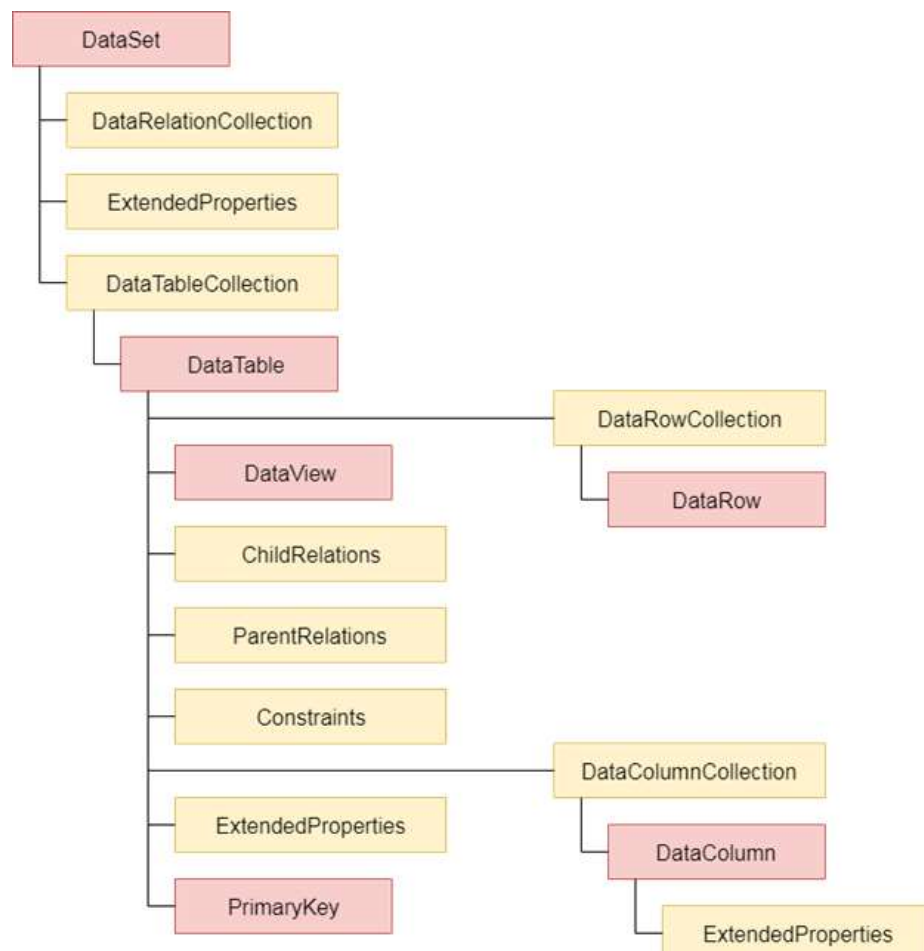


Fig: ADO.NET Architecture

Which one should we use DataReader or DataSet?

We should consider the following points to use DataSet.

It caches data locally at our application, so we can manipulate it.

It interacts with data dynamically such as binding to windows forms control.

It allows performing processing on data without an open connection. It means it can work while connection is **disconnected**.

If we required some other functionality mentioned above, we can use **DataReader** to improve performance of our application.

DataReader does not perform in disconnected mode. It requires DataReader object to be **connected**.

ADO.NET Framework Data Providers

Data provider is used to connect to the database, execute commands and retrieve the record. It is lightweight component with better performance. It also allows us to place the data into DataSet to use it further in our application.

The .NET Framework provides the following data providers that we can use in our application.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	It provides data access for Microsoft SQL Server. It requires the System.Data.SqlClient namespace.
.NET Framework Data Provider for OLE DB	It is used to connect with OLE DB. It requires the System.Data.OleDb namespace.
.NET Framework Data Provider for ODBC	It is used to connect to data sources by using ODBC. It requires the System.Data.Odbc namespace.
.NET Framework Data Provider for Oracle	It is used for Oracle data sources. It uses the System.Data.OracleClient namespace.
EntityClient Provider	It provides data access for Entity Data Model applications. It requires the System.Data.EntityClient namespace.

.NET Framework Data Provider for SQL Server Compact 4.0.

It provides data access for Microsoft SQL Server Compact 4.0. It requires the **System.Data.SqlServerCe** namespace.

.NET Framework Data Providers Objects

Following are the core object of Data Providers.

Object	Description
Connection	It is used to establish a connection to a specific data source.
Command	It is used to execute queries to perform database operations.
DataReader	It is used to read data from data source. The DbDataReader is a base class for all DataReader objects.
DataAdapter	It populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class.

.NET Framework Data Provider for SQL Server

Data provider for SQL Server is a lightweight component. It provides better performance because it directly access SQL Server without any middle connectivity layer. In early versions, it interacts with ODBC layer before connecting to the SQL Server that created performance issues.

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

```
using System.Data.SqlClient;
```

This namespace contains the following important classes.

Class	Description
SqlConnection	It is used to create SQL Server connection. This class cannot be inherited.
SqlCommand	It is used to execute database queries. This class cannot be inherited.
SqlDataAdapter	It represents a set of data commands and a database connection that are used to fill the DataSet. This class cannot be inherited.

SqlDataReader	It is used to read rows from a SQL Server database. This class cannot be inherited.
SqlException	This class is used to throw SQL exceptions. It throws an exception when an error is occurred. This class cannot be inherited.

.NET Framework Data Provider for Oracle

It is used to connect with Oracle database through Oracle client. The data provider supports Oracle client software version 8.1.7 or a later version. This data provider supports both local and distributed transactions.

Oracle Data Provider classes are located into **System.Data.OracleClient** namespace. We must use both **System.Data.OracleClient** and **System.data** to connect our application with the Oracle database.

```
using System.Data;
using System.Data.OracleClient;
```

Which .NET Framework Data Provider is better

Selection of data provider is depends on the design and data source of our application. Choice of optimum .NET Framework data provider can improve the performance, capability and integrity of our application. The following table demonstrates advantages and disadvantages of data provider.

Data Provider	Note
.NET Framework Data Provider for SQL Server	It is good for middle-tier applications, single-tier applications that use Microsoft SQL Server.
.NET Framework Data Provider for OLE DB	It is good for single-tier applications that use Microsoft Access databases.
.NET Framework Data Provider for ODBC	It is good for middle and single-tier applications that use ODBC data sources.
.NET Framework Data Provider for Oracle	It is good for middle and single-tier applications that use Oracle data sources.

ADO.NET SqlConnection Class

It is used to establish an open connection to the SQL Server database. It is a sealed class so that cannot be inherited. SqlConnection class uses SqlDataAdapter and SqlCommand classes together to increase performance when connecting to a Microsoft SQL Server database.

Connection does not close explicitly even it goes out of scope. Therefore, you must explicitly close the connection by calling Close() method.

SqlConnection Signature

```
public sealed class SqlConnection : System.Data.Common.DbConnection, ICloneable, IDisposable
```

SqlConnection Constructors

Constructors	Description
SqlConnection()	It is used to initializes a new instance of the SqlConnection class.
SqlConnection(String)	It is used to initialize a new instance of the SqlConnection class and takes connection string as an argument.
SqlConnection(String, SqlConnectionCredential)	It is used to initialize a new instance of the SqlConnection class that takes two parameters. First is connection string and second is sql credentials.

SqlConnection Methods

Method	Description
BeginTransaction()	It is used to start a database transaction.
ChangeDatabase(String)	It is used to change the current database for an open SqlConnection.

ChangePassword(String, String)	It changes the SQL Server password for the user indicated in the connection string.
Close()	It is used to close the connection to the database.
CreateCommand()	It enlists in the specified transaction as a distributed transaction.
GetSchema()	It returns schema information for the data source of this SqlConnection.
Open()	It is used to open a database connection.
ResetStatistics()	It resets all values if statistics gathering is enabled.

SqlConnection Example

Now, let's create an example that establishes a connection to the SQL Server. We have created a **Student** database and will use it to connect. Look at the following C# code.

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
}
```

Using block is used to close the connection automatically. We don't need to call close () method explicitly, **using** block do this for ours implicitly when the code exits the block.

// Program.cs

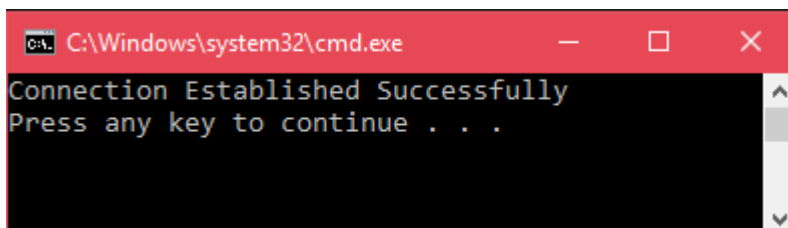
```
using System;
```

```

using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            new Program().Connecting();
        }
        public void Connecting()
        {
            using (
                // Creating Connection
                SqlConnection con = new SqlConnection("data source=.; database=
student; integrated security=SSPI")
            )
            {
                con.Open();
                Console.WriteLine("Connection Established Successfully");
            }
        }
    }
}

```

Output:



The screenshot shows a Windows command prompt window with a red title bar. The title bar text is "C:\Windows\system32\cmd.exe". The command prompt area is black with white text. It displays "Connection Established Successfully" on the first line and "Press any key to continue . . ." on the second line. A vertical scrollbar is visible on the right side of the command prompt area.

What, if we don't use **using** block.

If we don't use using block to create connection, we have to close connection explicitly. In the following example, we are using try-block instead of using block.

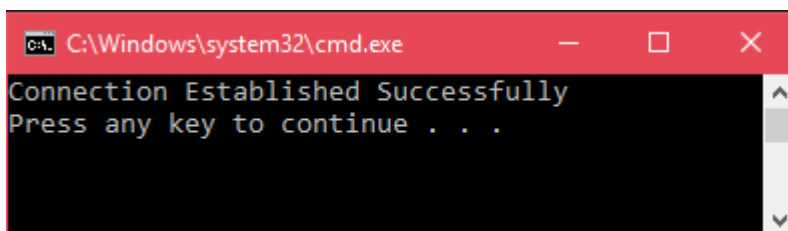
// Program.cs

```

using System;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            new Program().Connecting();
        }
        public void Connecting()
        {
            SqlConnection con = null;
            try
            {
                // Creating Connection
                con = new SqlConnection("data source=.; database=student; integrate
d security=SSPI");
                con.Open();
                Console.WriteLine("Connection Established Successfully");
            }
            catch (Exception e)
            {
                Console.WriteLine("OOPs, something went wrong.\n"+e);
            }
            finally
            {
                // Closing the connection
                con.Close();
            }
        }
    }
}

```

Output:



A screenshot of a Windows command prompt window. The title bar is red and shows the path 'C:\Windows\system32\cmd.exe'. The window has standard minimize, maximize, and close buttons. The command prompt area is black with white text. It displays the message 'Connection Established Successfully' followed by 'Press any key to continue . . .' on the next line. A vertical scrollbar is visible on the right side of the window.

ADO.NET SqlCommand Class

This class is used to store and execute SQL statement for SQL Server database. It is a sealed class so that cannot be inherited.

SqlCommand Signature

```
public sealed class SqlCommand : System.Data.Common.DbCommand, ICloneable, IDisposable
```

Constructors

This class provides the following constructors.

Constructor	Description
SqlCommand()	It is used to initialize a new instance of the SqlCommand class.
SqlCommand(String)	It is used to initialize a new instance of the SqlCommand class with a string parameter.
SqlCommand(String, SqlConnection)	It is used to initialize a new instance of the SqlCommand class. It takes two parameters, first is query string and second is connection string.
SqlCommand(String, SqlConnection, SqlTransaction)	It is used to initialize a new instance of the SqlCommand class. It takes three parameters query, connection and transaction string respectively.
SqlCommand(String, SqlConnection, SqlTransaction, SqlCommandColumnEncryptionSetting)	It Initializes a new instance of the SqlCommand class with specified command text, connection, transaction, and encryption setting.

Methods

Method	Description
--------	-------------

BeginExecuteNonQuery()	It is used to Initiate the asynchronous execution of the SQL statement described by this SqlCommand.
Cancel()	It tries to cancel the execution of a SqlCommand.
Clone()	It creates a new SqlCommand object that is a copy of the current instance.
CreateParameter()	It creates a new instance of a SqlParameter object.
ExecuteReader()	It is used to send the CommandText to the Connection and builds a SqlDataReader.
ExecuteXmlReader()	It is used to send the CommandText to the Connection and builds an XmlReader object.
ExecuteScalar()	It executes the query and returns the first column of the first row in the result set. Additional columns or rows are ignored.
Prepare()	It is used to create a prepared version of the command by using the instance of SQL Server.
ResetCommandTimeout()	It is used to reset the CommandTimeout property to its default value.

Example

In this example, we are creating a SqlCommand instance and executing a SQL statement.

// Program.cs

```

using System;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            new Program().CreateTable();
        }
    }

```

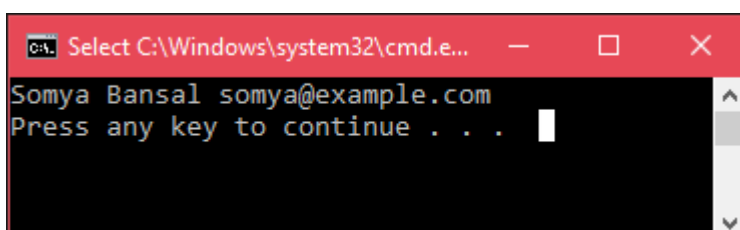
```

public void CreateTable()
{
    SqlConnection con = null;
    try
    {
        // Creating Connection
        con = new SqlConnection("data source=.; database=student; integrate
d security=SSPI");
        // writing sql query
        SqlCommand cm = new SqlCommand("select * from student", con);
        // Opening Connection
        con.Open();
        // Executing the SQL query
        SqlDataReader sdr = cm.ExecuteReader();
        while (sdr.Read())
        {
            Console.WriteLine(sdr["name"]+" "+ sdr["email"]);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("OOPs, something went wrong." + e);
    }
    // Closing the connection
    finally
    {
        con.Close();
    }
}
}

```

Output:

Execute this program by combination of **Ctrl+F5** and it will produce the following output.



The screenshot shows a Windows command prompt window with the title bar 'Select C:\Windows\system32\cmd.e...'. The window has a black background and white text. The first line of output is 'Somya Bansal somya@example.com'. The second line is 'Press any key to continue . . .', followed by a white cursor character.

It prints name and email of the student.

ADO.NET SqlDataReader Class

This class is used to read data from SQL Server database. It reads data in forward-only stream of rows from a SQL Server database. it is sealed class so that cannot be inherited. It inherits DbDataReader class and implements IDisposable interface.

SqlDataReader Signature

```
public class SqlDataReader : System.Data.Common.DbDataReader, IDisposable
```

SqlDataReader Properties

Property	Description
Connection	It is used to get the SqlConnection associated with the SqlDataReader.
Depth	It is used to get a value that indicates the depth of nesting for the current row.
FieldCount	It is used to get the number of columns in the current row.
HasRows	It is used to get a value that indicates whether the SqlDataReader contains one or more rows.
IsClosed	It is used to retrieve a boolean value that indicates whether the specified SqlDataReader instance has been closed.
Item[String]	It is used to get the value of the specified column in its native format given the column name.
Item[Int32]	It is used to get the value of the specified column in its native format given the column ordinal.
RecordsAffected	It is used to get the number of rows changed, inserted or deleted by execution of the Transact-SQL statement.
VisibleFieldCount	It is used to get the number of fields in the SqlDataReader that are not hidden.

Methods

Method	Description
Close()	It is used to closes the SqlDataReader object.
GetBoolean(Int32)	It is used to get the value of the specified column as a Boolean.
GetByte(Int32)	It is used to get the value of the specified column as a byte.
GetChar(Int32)	It is used to get the value of the specified column as a single character.
GetDateTime(Int32)	It is used to get the value of the specified column as a DateTime object.
GetDecimal(Int32)	It is used to get the value of the specified column as a Decimal object.
GetDouble(Int32)	It is used to get the value of the specified column as a double-precision floating point number.
GetFloat(Int32)	It is used to get the value of the specified column as a single-precision floating point number.
GetName(Int32)	It is used to get the name of the specified column.
GetSchemaTable()	It is used to get a DataTable that describes the column metadata of the SqlDataReader.
GetValue(Int32)	It is used to get the value of the specified column in its native format.
GetValues(Object[])	It is used to populate an array of objects with the column values of the current row.
NextResult()	It is used to get the next result, when reading the results of SQL statements.
Read()	It is used to read record from the SQL Server database.

To create a SqlDataReader instance, we must call the ExecuteReader method of the SqlCommand object.

Example

In the following program, we are using SqlDataReader to get data from the SQL Server. A C# code is given below.

// Program.cs

```
using System;
using System.Data.SqlClient;
namespace AdoNetConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            new Program().GetData();
        }
        public void GetData()
        {
            SqlConnection con = null;
            try
            {
                // Creating Connection
                con = new SqlConnection("data source=.; database=student; integrated security=SSPI");
                // writing sql query
                SqlCommand cm = new SqlCommand("select * from student", con);
                // Opening Connection
                con.Open();
                // Executing the SQL query
                SqlDataReader sdr = cm.ExecuteReader();
                while (sdr.Read())
                {
                    Console.WriteLine(sdr["name"]+" "+ sdr["email"]);
                }
            }
            catch (Exception e)
            {
            }
```

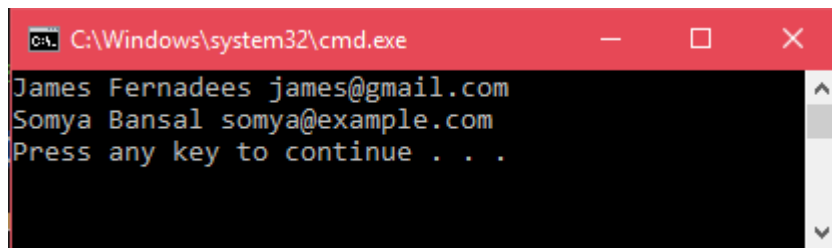
```

        Console.WriteLine("Oops, something went wrong." + e);
    }
    // Closing the connection
    finally
    {
        con.Close();
    }
}
}
}

```

Output:

Execute this program by combination of **Ctrl+F5** and it will produce the following output.



```

C:\Windows\system32\cmd.exe
James Fernadees james@gmail.com
Somya Bansal somya@example.com
Press any key to continue . . .

```

ADO.NET DataSet

It is a collection of data tables that contain the data. It is used to fetch data without interacting with a Data Source that's why, it also known as **disconnected** data access method. It is an in-memory data store that can hold more than one table at the same time. We can use DataRelation object to relate these tables. The DataSet can also be used to read and write data as XML document.

ADO.NET provides a DataSet class that can be used to create DataSet object. It contains constructors and methods to perform data related operations.

DataSet Class Signature

```

public class DataSet : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource,
System.ComponentModel.ISupportInitializeNotification, System.Runtime.Serialization.ISerializable,
System.Xml.Serialization.IXmlSerializable

```

DataSet Constructors

Constructor	Description
DataSet()	It is used to initialize a new instance of the DataSet class.
DataSet(String)	It is used to initialize a new instance of a DataSet class with the given name.
DataSet(SerializationInfo, StreamingContext)	It is used to initialize a new instance of a DataSet class that has the given serialization information and context.
DataSet(SerializationInfo, StreamingContext, Boolean)	It is used to initialize a new instance of the DataSet class.

DataSet Properties

Properties	Description
CaseSensitive	It is used to check whether DataTable objects are case-sensitive or not.
DataSetName	It is used to get or set name of the current DataSet.
DefaultViewManager	It is used to get a custom view of the data contained in the DataSet to allow filtering and searching.
HasErrors	It is used to check whether there are errors in any of the DataTable objects within this DataSet.
IsInitialized	It is used to check whether the DataSet is initialized or not.
Locale	It is used to get or set the locale information used to compare strings within the table.
Namespace	It is used to get or set the namespace of the DataSet.
Site	It is used to get or set an ISite for the DataSet.
Tables	It is used to get the collection of tables contained in the DataSet.

DataSet Methods

The following table contains some commonly used methods of DataSet.

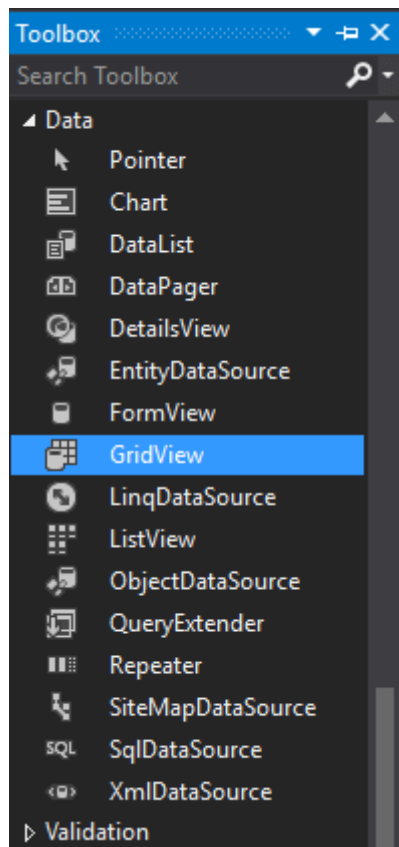
Method	Description
BeginInit()	It is used to begin the initialization of a DataSet that is used on a form.
Clear()	It is used to clear the DataSet of any data by removing all rows in all tables.
Clone()	It is used to copy the structure of the DataSet.
Copy()	It is used to copy both the structure and data for this DataSet.
CreateDataReader(DataTable[])	It returns a DataAdapter with one result set per DataTable.
CreateDataReader()	It returns a DataAdapter with one result set per DataTable.
EndInit()	It ends the initialization of a DataSet that is used on a form.
GetXml()	It returns the XML representation of the data stored in the DataSet.
GetXmlSchema()	It returns the XML Schema for the XML representation of the data stored in the DataSet.
Load(IDataReader, LoadOption, DataTable[])	It is used to fill a DataSet with values from a data source using the supplied IDataReader.
Merge(DataSet)	It is used to merge a specified DataSet and its schema into the current DataSet.
Merge(DataTable)	It is used to merge a specified DataTable and its schema into the current DataSet.
ReadXml(XmlReader, XmlReadMode)	It is used to read XML schema and data into the DataSet using the specified XmlReader and XmlReadMode.

Reset()	It is used to clear all tables and removes all relations, foreign constraints, and tables from the DataSet.
WriteXml(XmlWriter, XmlWriteMode)	It is used to write the current data and optionally the schema for the DataSet using the specified XmlWriter and XmlWriteMode.

Example:

Here, in this example, we are implementing **DataSet** and displaying data into a gridview. Create a web form and drag a gridview from the **toolbox** to the form. We can find it inside the data category.

Competitive questions on Structures in HindiKeep Watching



// DataSetDemo.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataSetDem
o.aspx.cs"
Inherits="DataSetExample.DataSetDemo" %>
```

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            </div>
            <asp:GridView ID="GridView1" runat="server" CellPadding="4" ForeColor="#3333
33" GridLines="None">
                <AlternatingRowStyle BackColor="White" />
                <EditRowStyle BackColor="#2461BF" />
                <FooterStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
                <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
                <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="
Center" />
                <RowStyle BackColor="#EFF3FB" />
                <SelectedRowStyle BackColor="#D1DDF1" Font-
Bold="True" ForeColor="#333333" />
                <SortedAscendingCellStyle BackColor="#F5F7FB" />
                <SortedAscendingHeaderStyle BackColor="#6D95E1" />
                <SortedDescendingCellStyle BackColor="#E9EBEF" />
                <SortedDescendingHeaderStyle BackColor="#4870BE" />
            </asp:GridView>
        </form>
    </body>
</html>

```

CodeBehind

// DataSetDemo.aspx.cs

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample

```

```

{
    public partial class DataSetDemo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (SqlConnection con = new SqlConnection("data source=.; database=student; integrated security=SSPI"))
            {
                SqlDataAdapter sda = new SqlDataAdapter("Select * from student", con);
                DataSet ds = new DataSet();
                sda.Fill(ds);
                GridView1.DataSource = ds;
                GridView1.DataBind();
            }
        }
    }
}

```

Output:

Execute this code by the combination of **Ctrl+F5**. It will produce the following output.

ADO.NET DataAdapter

The DataAdapter works as a bridge between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

DataAdapter Class Signature

```
public class DataAdapter : System.ComponentModel.Component, System.Data.IDataAdapter
```

DataAdapter Constructors

Constructors	Description
DataAdapter()	It is used to initialize a new instance of a DataAdapter class.
DataAdapter(DataAdapter)	It is used to initialize a new instance of a DataAdapter class from an existing object of the same type.

Methods

Method	Description
CloneInternals()	It is used to create a copy of this instance of DataAdapter.
Dispose(Boolean)	It is used to release the unmanaged resources used by the DataAdapter.
Fill(DataSet)	It is used to add rows in the DataSet to match the data source.
FillSchema(DataSet, SchemaType, String, IDataReader)	It is used to add a DataTable to the specified DataSet.
GetFillParameters()	It is used to get the parameters set by the user when executing an SQL SELECT statement.
ResetFillLoadOption()	It is used to reset FillLoadOption to its default state.
ShouldSerializeAcceptChangesDuringFill()	It determines whether the AcceptChangesDuringFill property should be persisted or not.
ShouldSerializeFillLoadOption()	It determines whether the FillLoadOption property should be persisted or not.
ShouldSerializeTableMappings()	It determines whether one or more DataTableMapping objects exist or not.
Update(DataSet)	It is used to call the respective INSERT, UPDATE, and DELETE statements.

Example

// DataSetDemo.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataSetDemo.aspx.cs" Inherits="DataSetExample.DataSetDemo" %>
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```

<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

        </div>
        <asp:GridView ID="GridView1" runat="server" CellPadding="3" BackColor="#DEBA
84"
            BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px" CellSpaci
ng="2">
            <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
            <HeaderStyle BackColor="#A55129" Font-
Bold="True" ForeColor="White" />
            <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />
            <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
            <SelectedRowStyle BackColor="#738A9C" Font-
Bold="True" ForeColor="White" />
            <SortedAscendingCellStyle BackColor="#FFF1D4" />
            <SortedAscendingHeaderStyle BackColor="#B95C30" />
            <SortedDescendingCellStyle BackColor="#F1E5CE" />
            <SortedDescendingHeaderStyle BackColor="#93451F" />
        </asp:GridView>
    </form>
</body>
</html>

```

CodeBehind

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace DataSetExample
{
    public partial class DataSetDemo : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            using (SqlConnection con = new SqlConnection("data source=.; database=studen
t; integrated security=SSPI"))
            {

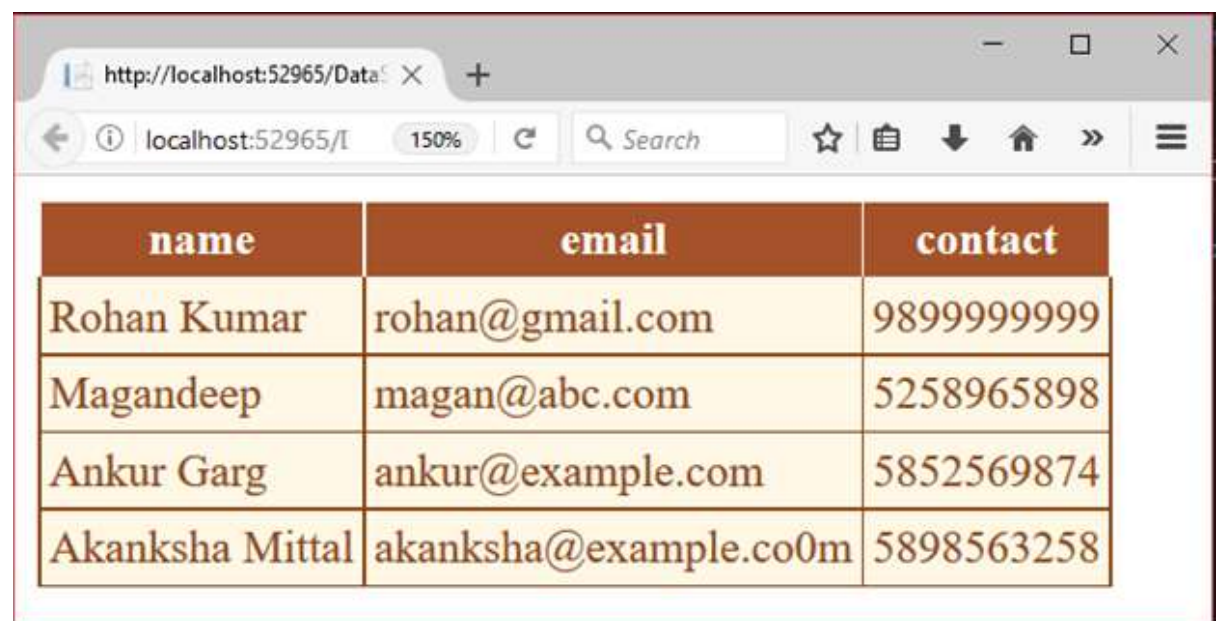
```

```

SqlDataAdapter sda = new SqlDataAdapter("Select * from student", con);
DataSet ds = new DataSet();
sda.Fill(ds);
GridView1.DataSource = ds;
GridView1.DataBind();
}
}
}
}
}

```

Output:



The screenshot shows a web browser window with the address bar displaying 'http://localhost:52965/Data'. The browser's address bar also shows 'localhost:52965/I' and a search bar. The main content area displays a table with three columns: 'name', 'email', and 'contact'. The table contains four rows of data.

name	email	contact
Rohan Kumar	rohan@gmail.com	9899999999
Magandeep	magan@abc.com	5258965898
Ankur Garg	ankur@example.com	5852569874
Akanksha Mittal	akanksha@example.co0m	5898563258

ADO.NET DataTable

DataTable represents relational data into tabular form. ADO.NET provides a DataTable class to create and use data table independently. It can also be used with DataSet also. Initially, when we create DataTable, it does not have table schema. We can create table schema by adding columns and constraints to the table. After defining table schema, we can add rows to the table.

We must include **System.Data** namespace before creating DataTable.

DataTable Class Signature

```

public class DataTable : System.ComponentModel.MarshalByValueComponent, System.ComponentModel.IListSource,
System.ComponentModel.ISupportInitializeNotification, System.Runtime.Serialization.ISerializable,
System.Xml.Serialization.IXmlSerializable

```

DataTable Constructors

The following table contains the DataTable class constructors.

Constructors	Description
DataTable()	It is used to initialize a new instance of the DataTable class with no arguments.
DataTable(String)	It is used to initialize a new instance of the DataTable class with the specified table name.
DataTable(SerializationInfo, StreamingContext)	It is used to initialize a new instance of the DataTable class with the SerializationInfo and the StreamingContext.
DataTable(String, String)	It is used to initialize a new instance of the DataTable class using the specified table name and namespace.

DataTable Properties

The following table contains the DataTable class properties.

Property	Description
Columns	It is used to get the collection of columns that belong to this table.
Constraints	It is used to get the collection of constraints maintained by this table.
DataSet	It is used to get the DataSet to which this table belongs.
DefaultView	It is used to get a customized view of the table that may include a filtered view.
HasErrors	It is used to get a value indicating whether there are errors in any of the rows in the table of the DataSet.
MinimumCapacity	It is used to get or set the initial starting size for this table.
PrimaryKey	It is used to get or set an array of columns that function as primary keys for the data table.

Rows	It is used to get the collection of rows that belong to this table.
TableName	It is used to get or set the name of the DataTable.

DataTable Methods

The following table contains the DataTable class methods.

Method	Description
AcceptChanges()	It is used to commit all the changes made to this table.
Clear()	It is used to clear the DataTable of all data.
Clone()	It is used to clone the structure of the DataTable.
Copy()	It is used to copy both the structure and data of the DataTable.
CreateDataReader()	It is used to returns a DataTableReader corresponding to the data within this DataTable.
CreateInstance()	It is used to create a new instance of DataTable.
GetRowType()	It is used to get the row type.
GetSchema()	It is used to get schema of the table.
ImportRow(DataRow)	It is used to copy a DataRow into a DataTable.
Load(IDataReader)	It is used to fill a DataTable with values from a data source using the supplied IDataReader.
Merge(DataTable, Boolean)	It is used to merge the specified DataTable with the current DataTable.
NewRow()	It is used to create a new DataRow with the same schema as the table.
Select()	It is used to get an array of all DataRow objects.
WriteXml(String)	It is used to write the current contents of the DataTable as XML using the specified file.

DataTable Example

Here, in the following example, we are creating a data table that populates data to the browser. This example contains the following files.

// DataTableForm.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="DataTableForm.aspx.cs"
Inherits="DataTableDemo.DataTableForm" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            </div>
        <asp:GridView ID="GridView1" runat="server">
            </asp:GridView>
        </form>
    </body>
</html>
```

CodeBehind

// DataTableForm.aspx.cs

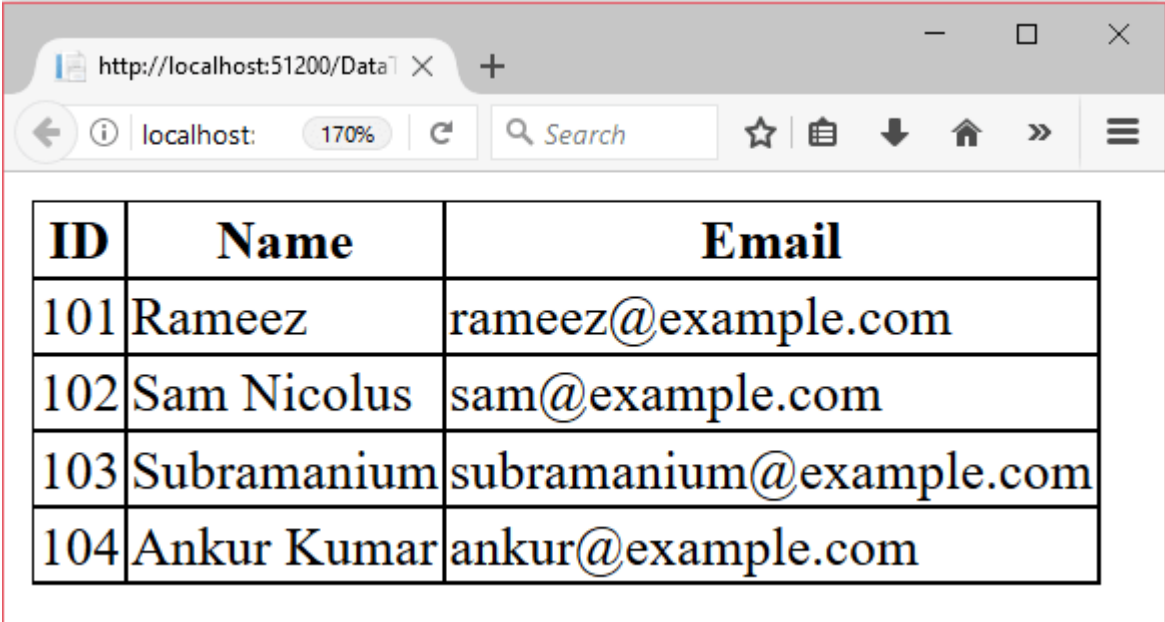
```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace DataTableDemo
{
    public partial class DataTableForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
```

```

DataTable table = new DataTable();
table.Columns.Add("ID");
table.Columns.Add("Name");
table.Columns.Add("Email");
table.Rows.Add("101", "Rameez", "rameez@example.com");
table.Rows.Add("102", "Sam Nicolus", "sam@example.com");
table.Rows.Add("103", "Subramanium", "subramanium@example.com");
table.Rows.Add("104", "Ankur Kumar", "ankur@example.com");
GridView1.DataSource = table;
GridView1.DataBind();
}
}
}

```

Output:



ID	Name	Email
101	Rameez	rameez@example.com
102	Sam Nicolus	sam@example.com
103	Subramanium	subramanium@example.com
104	Ankur Kumar	ankur@example.com

C# Public Access Specifier Example

```
using System;
```

```
namespace AccessSpecifiers
```

```

{
    class PublicTest
    {
        public string name = "Santosh Singh";
        public void Msg(string msg)
        {
            Console.WriteLine("Hello " + msg);
        }
    }
}

```

```
    }  
}  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        PublicTest publicTest = new PublicTest();  
        // Accessing public variable  
        Console.WriteLine("Hello "+publicTest.name);  
        // Accessing public method  
        publicTest.Msg("Peter Dicosta");  
    }  
}  
}
```

Output:

```
Hello Santosh Singh  
Hello Peter Dicosta
```