2. Explain about repeated-value data binding with an example program.

Repeated-value data binding allows you to display an entire table. This type of data binding requires a special control that supports it. Typically, this is a list control such as CheckBoxList or ListBox, but it can also be a much more sophisticated control such as the GridView. You will know that a control supports repeated-value data binding if it provides a DataSource property. Repeated-value binding does not necessarily need to use data from a database, and it does not have to use the ADO.NET objects. For example, you can use repeatedvalue binding to bind data from a collection or an array.

ListBox, DropDownList, CheckBoxList, and RadioButtonList: These web controls provide a list for a single field of information. HtmlSelect: This server-side HTML control represents the HTML.

HtmlSelect: This server-side HTML control represents the HTML<select> element and works essentially the same way as ListBoxControl.

GridView, DetailsView, FormView, and ListView: These rich web controls allow you to provide repeating lists or grids that can display more than one field of information at a time.

With repeated-value data binding, you can write a data-binding expression in your .aspx file, or you can apply the data binding by setting control properties.

- List fruit = new List();
- fruit.Add("Kiwi");
- fruit.Add("Pear");
- fruit.Add("Mango");
- fruit.Add("Blueberry");
- fruit.Add("Apricot");
- fruit.Add("Banana");
- fruit.Add("Peach");
- fruit.Add("Plum");
- lstItems.DataSource=fruit;
- this.DataBind();
- List <string>fruit;
- fruit = GetFruitsInSeason("Summer");
- lstItems.DataSource = fruit;
- this.DataBind();
- lstItems.DataSource = GetFruitsInSeason("Summer");
- this.DataBind();

3. Explain about details view with an example program.

Details View

- DetailsView is a formatted data control like GridView Control in ASP.NET.
- The DetailsView control displays only a single data record at a time, even if its data source exposes multiple records.
- CRUD operations are possible in DetailsView.
- **asp:BoundField** is used to bind data in DetailsView.

The following are some important templates of DetailsView Control:

<HeaderTemplate>
<FooterTemplate>
<PagerTemplate>
<Fields>

Here's the basic DetailsView tag that makes this possible:

```
<asp:DetailsView ID="detailsProduct" runat="server"
  DataSourceID="sourceProductDetails" />
```

Other Types of Parameters:

The @ProductID parameter in the second SqlDataSource is configured based on the selection in a drop-down list. This type of parameter, which links to a property in another control, is called a control parameter. But parameter values are not necessarily drawn from other controls.

Source	Control Tag
Control property	<asp:ControlParameter>
Query string value	<asp:QueryStringParamete
Session state value	<asp:SessionParameter>
Cookie value	<asp:CookieParameter>
Profile value	<asp:ProfileParameter>
Routed URL value	<asp:RouteParameter>
A form variable	<asp:FormParameter>

```
<asp:SqlDataSource ID="sourceProducts" runat="server"
  ProviderName="System.Data.SqlClient"
  ConnectionString="<%$ ConnectionStrings:Northwind %>"
  SelectCommand="SELECT ProductName, ProductID FROM Products"
/>
<asp:DropDownList ID="lstProduct" runat="server" AutoPostBack="True
  DataSourceID="sourceProducts" DataTextField="ProductName"
  DataValueField="ProductID" />
```

```
protected void cmdGo_Click(object sender, EventArgs e)
{
    if (lstProduct.SelectedIndex != -1)
    {
        Response.Redirect(
          "QueryParameter2.aspx?prodID=" + lstProduct.SelectedValue);
    }
}
```

Finally, the second page can bind the DetailsView according to the ProductID value that's supplied in the query string:

```
<asp:SqlDataSource ID="sourceProductDetails" runat="server"
  ProviderName="System.Data.SqlClient"
  ConnectionString="<%$ ConnectionStrings:Northwind %>"
  SelectCommand="SELECT * FROM Products WHERE ProductID=@ProductID">
  <SelectParameters>
    <asp:QueryStringParameter Name="ProductID" QueryStringField="prodID" />
  </SelectParameters>
</asp:SqlDataSource>

<asp:DetailsView ID="detailsProduct" runat="server"
  DataSourceID="sourceProductDetails" />
```

4. Explain about ADO.NET disconnected model with an example.

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.

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

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.

Disconnected

It is dis_connection oriented.

DataSet

Disconnected get low in speed and performance.

disconnected can hold multiple tables of data.

disconnected you cannot.

Data Set can persist the data.

We can update data.

Example

```
Create Database —Student(
    [ID] [int] PRIMARY KEY IDENTITY(1,1) NOT NULL, [Name] [varchar](255) NULL,
    [Age] [int] NULL,
    [Address] [varchar](255) NULL
)
 INSERT INTO Student([Name],[Age],[Address])VALUES('NAME1','22','PUNE')
INSERT INTO Student([Name],[Age],[Address])VALUES('NAME 2','25','MUMBAI')
INSERT INTO Student([Name],[Age],[Address])VALUES('NAME 3','23','PUNE')
INSERT INTO Student([Name],[Age],[Address])VALUES('NAME 4','21','DELHI')
```

HTML Page:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>Untitled Pagetitle>
</head>
<body>
    <form id="form1" runat="server">
    <div>
<asp:GridView ID="GridView1" runat="server" BackColor="White" BorderColor="#CC9966"
 BorderStyle="None" BorderWidth="1px" CellPadding="4"> <FooterStyle BackColor="#FFFFCC" Fore
Color="#330099" /> <RowStyle BackColor="White" ForeColor="#330099" />
 <PagerStyle  BackColor="#FFFFCC" ForeColor="#330099" HorizontalAlign="Center"/>
 <SelectedRowStyle BackColor="#FFCC66" Font-Bold="True" ForeColor="#663399"                    />
<HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="#FFFFCC" />
 </asp:GridView><br />
            <asp:Button  ID="Connected"  runat="server"  OnClick="Connected_Click"
            Text="Connected" />
<asp:Button ID="Disconnected" runat="server" EnableTheming="False"OnClick="Disconnected_Click
" Text="Disconnected" />
 </div> </form>
</body> </html>
```

Code Behind:

```
String StrSQL = "", StrConnection = "";
        protected void Page_Load(object sender, EventArgs e){
            StrSQL = "SELECT * FROM Student";
            StrConnection = "DataSource=ServerName;Initial Catalog=Database;UserID=Username;
            Password=password";
        }
        protected void Disconnected_Click(object sender, EventArgs e){
            SqlDataAdapter objDa = new SqlDataAdapter();
             DataSet objDs = new DataSet();
```

```
        using (SqlConnection objConn = new SqlConnection(StrConnection)) {
        SqlCommand objCmd = new SqlCommand(StrSQL,objConn);
        objCmd.CommandType = CommandType.Text;
        objDa.SelectCommand = objCmd;
        objDa.Fill (objDs, "Student");
        GridView1.DataSource = objDs.Tables[0]; GridView1.DataBind();
    }
}
```

5. Explain about LINQ to objects with an example program.

LINQ defines keywords that you can use to select, filter, sort, group, and transform data. The minor miracle of LINQ is that different LINQ providers allow these keywords to work with different types of data. Here are some of the LINQ providers included with .NET 4.5:

LINQ to Objects: This is the simplest form of LINQ. It allows you to query collections of in-memory objects (such as an array, an ArrayList, a List, a Dictionary, and so on).

Queries in LINQ to Objects return variables of type usually IEnumerable<T> only. In short, LINQ to Objects offers a fresh approach to collections as earlier, it was vital to write long coding (foreach loops of much complexity) for retrieval of data from a collection which is now replaced by writing declarative code which clearly describes the desired data that is required to retrieve.

There are also many advantages of LINQ to Objects over traditional foreach loops like more readability, powerful filtering, capability of grouping, enhanced ordering with minimal application coding. Such LINQ queries are also more compact in nature and are portable to any other data sources without any modification or with just a little modification.

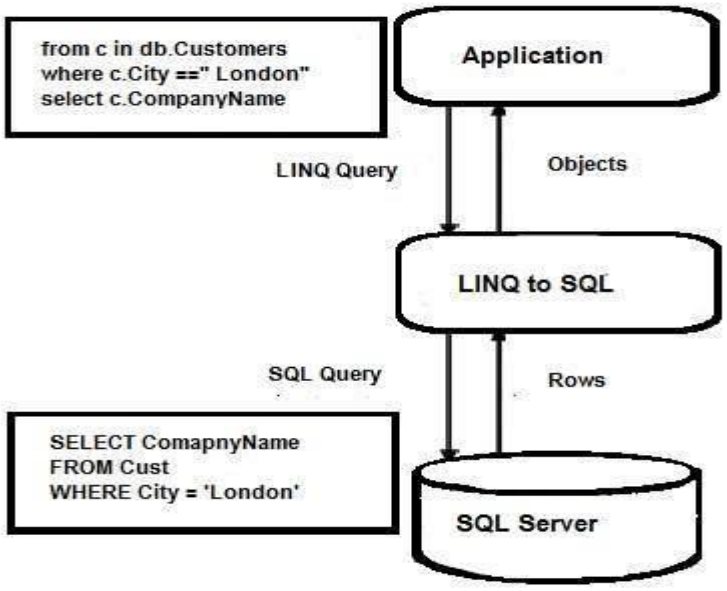
Example:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LINQtoObjects {
   class Department {
      public int DepartmentId { get; set; }
      public string Name { get; set; }
   }
   class LinqToObjects {
      static void Main(string[] args) {

         List<Department> departments = new List<Department>();

         departments.Add(new Department { DepartmentId = 1, Name
= "Account" });
```

```
            departments.Add(new Department { DepartmentId = 2, Name
= "Sales" });
            departments.Add(new Department { DepartmentId = 3, Name
= "Marketing" });

            var departmentList = from d in departments
                                 select d;

            foreach (var dept in departmentList) {
               Console.WriteLine("Department Id = {0} , Department
Name = {1}",
                    dept.DepartmentId, dept.Name);
            }

            Console.WriteLine("\nPress any key to continue.");
            Console.ReadKey();
        }
    }
}
```

6. Explain about LINQ to SQL with an example.

For most ASP.NET developers, LINQ to SQL (also known as DLINQ) is an electrifying part of Language Integrated Query as this allows querying data in SQL server database by using usual LINQ expressions. It also allows to update, delete, and insert data, but the only drawback from which it suffers is its limitation to the SQL server database. However, there are many benefits of LINQ to SQL over ADO.NET like reduced complexity, few lines of coding and many more.



Step 1 – Make a new “Data Connection” with database server. View &arrar; Server Explorer &arrar; Data Connections &arrar; Add Connection

Step 2 – Add LINQ To SQL class file

Step 3 – Select tables from database and drag and drop into the new LINQ to SQL class file.

Step 4 – Added tables to class file.

Querying with LINQ to SQL

The rules for executing a query with LINQ to SQL is similar to that of a standard LINQ query i.e. query is executed either deferred or immediate. There are various components that play a role in execution of a query with LINQ to SQL and these are the following ones.

- **LINQ to SQL API** – requests query execution on behalf of an application and sent it to LINQ to SQL Provider.
- **LINQ to SQL Provider** – converts query to Transact SQL(T-SQL) and sends the new query to the ADO Provider for execution.
- **ADO Provider** – After execution of the query, send the results in the form of a DataReader to LINQ to SQL Provider which in turn converts it into a form of user object.

It should be noted that before executing a LINQ to SQL query, it is vital to connect to the data source via DataContext class.

Example for Insert:

```
using System;
using System.Linq;

namespace LINQtoSQL {
   class LinqToSQLCRUD {
      static void Main(string[] args) {

         string connectString =
System.Configuration.ConfigurationManager.ConnectionStrings["Linq
ToSQLDBConnectionString"].ToString();

         LinqToSQLDataContext db = new
LinqToSQLDataContext(connectString);

         //Create new Employee

         Employee newEmployee = new Employee();
         newEmployee.Name = "Michael";
         newEmployee.Email = "yourname@companyname.com";
         newEmployee.ContactNo = "343434343";
         newEmployee.DepartmentId = 3;
         newEmployee.Address = "Michael - USA";
```

```
            //Add new Employee to database
            db.Employees.InsertOnSubmit(newEmployee);

            //Save changes to Database.
            db.SubmitChanges();

            //Get new Inserted Employee
            Employee insertedEmployee =
db.Employees.FirstOrDefault(e =>e.Name.Equals("Michael"));

            Console.WriteLine("Employee Id = {0} , Name = {1}, Email
= {2}, ContactNo = {3}, Address = {4}",
                              insertedEmployee.EmployeeId,
insertedEmployee.Name, insertedEmployee.Email,
                              insertedEmployee.ContactNo,
insertedEmployee.Address);

            Console.WriteLine("\nPress any key to continue.");
            Console.ReadKey();
        }
    }
}
```

7.Explain in detail about WCF.

WCF stands for Windows Communication Foundation. It is basically used to create a distributed and interoperable Application. WCF Applications came into the picture in .Net 3.0 Framework. This is a framework, which is used for creating Service oriented Applications. You can send the data asynchronously from one end point to another.

Creating a WCF service is a simple task using Microsoft Visual Studio 2012. Given below is the step-by-step method for creating a WCF service along with all the requisite coding, to understand the concept in a better way.

- Launch Visual Studio 2012.
- Click on new project, then in Visual C# tab, select WCF option.
- A WCF service is created that performs basic arithmetic operations like addition, subtraction, multiplication, and division. The main code is in two different files – one interface and one class.

A WCF contains one or more interfaces and its implemented classes.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServiceLibrary1 {
   // NOTE: You can use the "Rename" command on the "Refactor"
menu to
   // change the interface name "IService1" in both code and
config file
   // together.

   [ServiceContract]
   Public interface IService1 {
      [OperationContract]
      int sum(int num1, int num2);

      [OperationContract]
      int Subtract(int num1, int num2);

      [OperationContract]
      int Multiply(int num1, int num2);

      [OperationContract]
      int Divide(int num1, int num2);
   }

   // Use a data contract as illustrated in the sample below to
add
   // composite types to service operations.

   [DataContract]
   Public class CompositeType {
      Bool boolValue = true;
      String stringValue = "Hello ";

      [DataMember]
      Public bool BoolValue {
         get { return boolValue; }
         set { boolValue = value; }
      }

      [DataMember]
      Public string StringValue {
         get { return stringValue; }
         set { stringValue = value; }
      }
   }
}
```

The code behind its class is given below.

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Runtime.Serialization;
usingSystem.ServiceModel;
usingSystem.Text;

namespace WcfServiceLibrary1 {
   // NOTE: You can use the "Rename" command on the "Refactor"
menu to
   // change the class name "Service1" in both code and config
file
   // together.

   publicclassService1 :IService1 {
      // This Function Returns summation of two integer numbers

      publicint sum(int num1, int num2) {
         return num1 + num2;
      }

      // This function returns subtraction of two numbers.
      // If num1 is smaller than number two then this function
returns 0

      publicint Subtract(int num1, int num2) {
         if (num1 > num2) {
            return num1 - num2;
         }
         else {
            return 0;
         }
      }
      // This function returns multiplication of two integer
numbers.
      publicint Multiply(int num1, int num2) {
         return num1 * num2;
      }

      // This function returns integer value of two integer
number.
      // If num2 is 0 then this function returns 1.
      publicint Divide(int num1, int num2) {
         if (num2 != 0) {
            return (num1 / num2);
         } else {
            return 1;
         }
      }
   }
}
```