Introduction To .NET

In this lecture you will understand:

- Today's software scenario
- What Is .NET & why use it
- Roadmap for learning
What Is It?

.NET is a framework consisting of common language runtime and a set of .NET base classes which allow us to build distributed, secure, scaleable, robust, easily deployable and maintainable applications in a language independent, object-oriented, web enabled, managed way using rich level of integration amongst present technologies. Most of the books contain such a complicated definition of .NET. Somebody who begins his tryst with .NET will find this definition too heavy.

Software Scenario

Many applications are build today are Desktop applications. Some are Web applications. We also have Embedded applications. We have Distributed applications, Operating Systems, Compilers and Device Drivers. So anybody who is trying to do any kind of computing in today’s software world would be doing one of these different things. Desktop application can be an application like Internet Explorer, Antiviral Software like AVG, simple editor like Notepad, simple admission system where students come in and register for a particular course. These applications are typically build using VB and VC++ in the past and they are now build using .NET technologies. Web applications like NDTV website, MSN website. Earlier these websites were build using technologies like ASP or JSP. In today’s software scenario these web applications can be easily build using .NET technologies. ASP.NET is one of the subset of the .NET technologies created solely for building web applications and web services. Initially Embedded systems were build using C or Java programming. But increasingly there is migration towards using .NET for building even the Embedded systems. An Embedded system is a system where a program is running but the device on which it is running is not a computer. Mobile phone, washing machine and microwave are examples of Embedded system. The programs which run in Embedded systems can also be build using .NET technologies. A banking system, airline reservation system and railway reservation system are good examples of distributed computing. Earlier this distributed computing was done using C language to begin with and subsequently using a technology created by Microsoft known as Component Object Model (COM). These technologies are completely overtaken by .NET. In a distributed application scenario we have banking system and there are different branches. Those branches carry out business. They transact business. So machines which are present in different banks have to communicate with centralized server for the transactions to happen. In such distributed computing also .NET technologies have a big role to play. Operating system, compilers and device drivers are build using C and C++. .NET is all about building applications efficiently.

Why .NET?

- It is easy to develop applications in .NET because
  - We can actually check as you type like spell-check or grammar check which we make use in Word, PowerPoint or Excel.
  - It offers a technology created by Microsoft called as Intellisense.
  - We do not have to remember the function parameters, their type and their order. This becomes available to us on the fly as we start typing.
  - It provides very good examples/samples.
  - We get exhaustive MSDN help.
- Microsoft has taken lot of pains to ensure that development of any application becomes very rapid. Means turn around time for any project should not be very long. Everything should be made available to developer which helps into reduce the time of development.
  - Media player takes 10 minutes.
Embedded Web browser takes 12 minutes.
Embedded Excel browser takes 9 minutes.

- No matter what type of application we are building i.e. desktop application, embedded application or web application. The language does not change i.e. same language is used. Same tool is used for multiple applications.
- It is easy to manage large projects which has 1,00,000 lines code because it provides a source control mechanism, a good debugging platform and a good way of organizing all the files and resources of the project. These tools are needed to build applications efficiently.

.NET Technologies

There are three basic categories in which we tried to build .NET applications. They are Desktop applications, Mobile applications and ASP.NET applications.

Desktop Track

We expect desktop applications to have some sort of Internet connectivity. AVG antiviral software is an example of it. Everyday we put on the machine, the AVG application checks whether we are connected to the Internet and if we are, it indeed goes there and downloads the latest signatures. This way our antiviral software remains up-to-date. Any new viruses that have posed, they are taken into account by downloading suitable virus signatures every time we put on the machine and connect to the Internet. Desktop applications also allow to create pluggable interfaces. Google toolbar is an example of it. This Google toolbar is attached to the Internet Explorer. Yahoo also provides a toolbar which gets embedded in the Internet Explorer. Means Internet Explorer is a desktop application but people do want to provide plug-ins which will integrate with Internet Explorer. Desktop applications are becoming functionally much richer. For example, we expect that any proper desktop application would have any kind of help available. We also expect that search to become available within the application. Likewise updation of the existing desktop software is popularly being done today.

Mobile Track

In the mobile track, we have to be careful about the screen sizes. While building a standard application which runs on a PC, we can easily assume that the screen can be either 800 x 600 resolution screen or 1024 x 768 resolution screen whereas with mobiles, we get much smaller screens. The geometry of different mobile devices are different. Some are laid out horizontally, some are laid out vertically. Even if laid out vertically, size of each mobile phone’s display is also different. User interfaces are different. Some mobile phones make use of a pointer, a stick using which we can click on the screen which is touch sensitive such that we can use it as if we are marking numbers with a pen or tapping the calculator using fingers. What kinds of user-interfaces are there is depends upon the model of the mobile phone. When we are using mobile phones, connectivity is intermittent. When we are using mobile phones, connectivity is intermittent. We cannot assume that our phone is connected to Internet all times. We also even cannot assume that our phone is connected to carrier networks at all times. We have to account for all these possibilities whenever we are trying to build applications for the mobile phones. Synchronization of address book is an example of mobile phone.

ASP.NET Track

In the ASP.NET track, we might create different applications which act like portals. Portals are very comprehensive. For example, from Yahoo’s portal we will get so much useful information from different spheres right from what is the weather in different parts of the world, what kind of programs are being organized in the city which we are visiting, alarm systems, calendaring system, emailing. All this put together is a portal. In different websites, different themes are being followed. In the early days of web programming, different pages even though they had a similar look and feel, there wasn’t
a mechanism available where we declare a theme and use it in all the pages. With .NET that has become possible. This leads to faster development time. We also get different modules today. For example, if we want that there should be an emailing system available in our website then we just have to plug that module in our website. If we want that the calendaring system should be available or a shopping cart should be available, we have to plug that in our website. Those kinds of applications we might be required to create for a web application. .NET makes it easier for us.

.NET Based Software Development

For .NET based software development, we need .NET IDE which lets us build .NET applications quickly. We also need a compiler and a linker and put together we call it as a Software Development Kit (SDK). Unless we are able to build the program, debug the program, do the profiling of that program, these are very commonly used tools in any software development environment. We also need a .NET Runtime on which we run our applications. We get .NET runtime readymade when we install Windows OS whereas .NET IDE if we make use of then there are two choices which are available for us. Visual Studio 2008 and Visual Web Developer 2008 Express editions. We propose to make use of the Express Editions. This Visual Web Developer 2008 Express editions is a Rapid Application Development tool. Turn around time for any application is being much smaller using this tool. It has wizards which generates readymade skeletons. Very sophisticated debugging help is available. It has now become an Industry standard. It is free of cost legally. We can download it from www.microsoft.com/express/product/. With this link we will be taken to the Express Edition page of the Microsoft’s site. There we see Visual C# Express, Visual Web Developer Express and SQL Server 2005 Express. We should download these Express Editions. We need Visual C# if we are trying to build desktop applications.
Web Model Fundamentals

In this lecture you will understand:

- Desktop versus Web applications
- Web application fundamentals
- Web application development
Desktop Versus Web Applications

We begin with where is a web application located and where is a desktop application located. Then how do we invoke or activate a web application and how do we activate a desktop application. How many users should we expect a web application to cater to and how many users would be making use of a desktop application. Then we see how the access is done. How connectivity expectations are different for a web application and for a desktop application. What is the basic programming model that a desktop application uses and what is that model that a web application uses. Then what is the state which is different in desktop application and a web application as far as a state model is concerned. Then we will deal with the complexity that is involved in the desktop application and how does it contrast with the complexity involved in a web application. All these differences we are going to discuss.

Location

A desktop application typically runs on a machine and it is invoked from exactly the same machine on which it is running. We typically execute a desktop application by double clicking its icon. We can also execute the desktop application from the command line. Command line is accessed through Start | Run menu item and try to execute command line shell by typing cmd in the dialog that we get when we do Start | Run. Then we get a command window that’s the shell, text based shell. There we can execute the applications at the command line. For example, we can execute Internet Explorer by typing iexplorer.exe at the command prompt. We can start Microsoft Word by typing winword. Means for every single application, we may not want to do a double click, we can even execute those applications from command line. In a typical web application, application is anyway running on the web server. Whenever a web application is deployed on a web server, the moment we boot that machine i.e. the server machine, the application gets launched. Means whenever we are trying to access a website, its not that we are trying to start the website. Website is already running there on the web server. Unless we double-click the word icon, the word is not in the memory. We have to launch it into the memory whereas a web application is already running in memory but we have to access it through a web browser. If we want to visit Google, we just type out www.google.com in the browser. We invoke an application by sending a request through a web browser. Application is already running in memory of the web server. Means web model is basically a distributed model. The machine where the application is running is different than the machine from which we are trying to interact with that. Whereas typically in a desktop application, the machine where the application is running and where the interaction is happening is commonly the same machine. Most desktop applications would be activated from the same machine on which they are deployed whereas in a web application even the simplest of simple web application, application runs on a different machine; it’s invoked from a different machine. So physical location of somebody who is trying to interact with the application and the machine on which it is actually running, they are two different machines.

Invocation

A desktop application is invoked by double-clicking its icon or from the command-line whereas the web application is already running on the web server. We just have to send a request to the web server saying that let us carry out interaction with this already running application on the web server. For example, we may start Internet Explorer and then type out some website address here may be quest.ksetindia.com where we have hosted our store. So it is easily accessible to us. We have to send the request to this particular site. The quest.ksetindia.com really gets mapped onto the IPAddress of the machine where this particular site is hosted. quest.ksetindia.com gets converted to a 32-bit address wherein it identifies the machine where this particular site is hosted. From that particular machine, we are sending a request to some default page. The default page is returned to us. That default page is rendered or displayed within the browser.
Number Of Users

In desktop applications, typically there is one user at a time. Most desktop applications would have one user at a time unless we have created a desktop application to run in a network environment. For example, there might be an admission system wherein there are three desks at which admission information is being received and fees are being collected. So that’s a networked application. It's not Internet enabled application. It’s still a desktop application but all these PCs are connected in a network and may be they are communicating with a centralized server where all the data about the students is getting stored as it is being received at three different nodes or three different machines. Web applications are typically going to be used by thousands of users may be simultaneously. As a result of which the web server where this application is deployed has to be a powerful server necessarily. So any web application that we build, we always carry out a stress test on that application because by the nature of a web application, we expect that several hundred users would be connected to the site simultaneously. Is that scenario is being addressed in the website or not, that we need to figure out. Hence the basic parameters of testing itself changes when we talk about a web application as compared to a desktop application because by its very nature the number of users are just too many for a web application. Clients from different machines can physically interact with this application that is running on the centralized web server.

Connectivity

In case of a desktop application, the connectivity is needed rarely. For example, AVG antiviral connects to the Internet when we start the AVG application or when we put on the machine, the AVG software automatically starts and then it figures out whether Internet connection is alive at that moment and if possible, if it is alive then it connects to the Internet and tries to retrieve the latest virus signatures and downloads them automatically. If we want that those signatures should get downloaded periodically then we need limited connectivity. In case of any web application, it is understood that we are at all times connected to the Internet. Unless there is Internet connectivity, there is no web application running. Application might continue to run but unless we are connected to that application through Internet, that application really means nothing to us. Connectivitywise expectations of desktop platform and web platform are completely different.

Programming Model

In a typical Windows based desktop application, there is a Event Driven programming model means in a window if we have some text boxes, some buttons, some list boxes, some combo boxes, some tree controls then as and when we interact with these controls an event is raised. Then there are functions present in the application which react to these occurrence of different events. For example, if we click on a mouse button then some method within the application would be called. We have to arrange for this, when we build desktop applications. But basically it’s an Event driven model. As events occurred, suitable functions get called. Web application model is basically a request-response model means application is already running, we have send a HTTP request saying that give us a particular page. It returns a response and that response gets rendered within the browser. There has to be some rule according to which these requests are made. Those set of rules is known as a protocol. The commonly used protocol for interacting with a web application is HTTP protocol. According to this protocol, only we have to send a request and according to this protocol, only the response comes back to us. So writer of every web application has to ensure that whenever he sends a response for the request made by the user that response is uniform. Then only our browser would be able to understand it properly and display that response to us within the window of the browser. So the way it is imperative, it is necessary that the request has to be done in a uniform and consistent manner, same way it is the responsibility of the web developer to send the response to its client also in a uniform manner and to ensure this uniformity, we have protocols like HTTP through which we make a request and through which we make a response.
State Model

In case of a desktop application, it’s a stateful model. For example, if we have two buttons present on a desktop application. When we click on one button, some event happens. As a result of which, some function gets called and then some calculation is done within that function which is kept somewhere in memory. Then we click on the other button, some other event will occur, some other function would be called but this function can have an access to the answer that we got when we clicked on the 1st button. Means whatever is the state that is something that is available for subsequent actions. Whatever calculations that we did when we click the 1st button, result of that calculation can be made easily available to another event handler which gets called when the 2nd button is clicked. In case of web application this is not so at all. HTTP protocol which is used for making a request and sending back a response to the user, it’s a stateless protocol. Stateless protocol means what happens during the 1st request is never remembered. So those results that occurred when the 1st request was made are not available when the 2nd request is made. So that is the way the HTTP protocol has been created. So the web model is a stateless model. The 1st request and the 2nd request are quite unrelated. 2nd request does not know that some 1st request has already been made by the user. This leads to problems and those problems were addressed by technologies like ASP.NET by storing the state of the 1st request for the user. HTTP does not support that but ASP.NET model allows to build that kind of statefulness into the application. Statefulness is built into the application not by HTTP but by ASP.NET which lets us remember what is it that the user has done during the previous request.

Complexity

Desktop applications are simple whereas web applications are complex. The devices through which we access a website are different. For example, if we want to access the website present on server, the application remains on the same server but the devices on the client side change completely. We can access a website through the desktop, we can also access it through laptop, we can access it through a PDA or even a mobile phone. Geometry of these devices are very different and that has an impact on building the web application. We have to build the application keeping in mind that a user is going to connect through any of these four different methods and he should still be happy to see what he gets on a screen. So we have to render the response in such a manner or we have to send the response in such a manner that based on what kind of device the user had made a request from, it still gets properly displayed on his device. In desktop applications it is going to deployed on a machine which is going to have at the most 2-3 different resolutions. Different users may use different browsers to send request to our application. Means what kind of request is going to come is same but the capabilities of these different browsers would be different. Settings that we do within the browser might also be different.

Web Application Fundamentals

Web applications have no user interface. Application is running on the server that application does not have any kind of user interface to offer to us. The web application organizes itself into different pages and classes. Most of the web application development today is being done in an Object Oriented manner. Pages contain some directives and HTML tags. The client would send a request for a particular web page. The pages based on the directives that are present in that and the classes that we have written in the background, those would interact with one another to create a response. This response is always in the form of HTML. Response does not contain any directives. Directives within the page and the classes that we have written they work with one another and try to generate a response. This response is in HTML. This response is the one that is send to the client. This response may contain text, images, video, flash, etc. Combination of all these together construct a response. This response which is send using the HTTP protocol is interpreted by the browser and based on the kind of HTML tags present within that response, appropriately the user interface is then rendered for us in the browser.
Web Application Development

Development is done on a local machine. Once the development is over we deploy the application on the remote machine on which the web server is present. When we carry out the development there are different options available to us for building websites. We might build websites using ASP.NET or using Java technologies or using PHP technologies. When we carry out deployment it would be either in Internet Information Server (IIS) i.e. if we build application using ASP.NET then we need IIS web server as our deployment choice. If we have built the application using PHP then we may want to go for the Apache web server or if we have built the website in Java then we may need application servers like WebLogic, WebSphere or TomCat application server.

Why Local Development

We always have to build locally but deploy remotely. If we are trying to create an application, we are not going to finish that application in one day. So we have to have time to build that application. An incomplete application, we cannot expose to the users. Hence we must do the development locally, do the testing locally and then deploy the application on a production server or a web server. Because once it is out on the web server, it’s available to every body. So we cannot do a half job and make it available everybody to the world. It brings efficiency to the development because if the site is deployed remotely in tat case carrying out debugging, carrying out access to the pages is going to take time. Instead if we have environment on the PC then it is definitely going to improve the speed. Then the user should not be able to tamper with the live data. So in a web application scenario this becomes important that we always create, build, test locally. Once we are sure that everything is fine then only deploy the application on a web server.
Creating First Website

In this lecture you will understand:

- Evolution of Web applications
- Web development tools
- Why ASP.NET
- Creation of first web site
Web Evolution

Four different types of applications in this evolution phase could be a collection of static inter-linked pages. Means in the early days of web people created 10-15 pages for their website, manage to put them on the web and from one page we could quickly navigate to the other page. Links were provided in each page and through those links we are able to navigate from one page to another. They began with the home page and within home page they provided different links through which we could really move onto the other pages present in the website. Different flight scheduling information which is present on different websites of Airlines is an example of this. This phase lasted for quite some time. After that came an era wherein people realized that why are only static contents on the website? Why can’t we put dynamic contents on the website? If we take the same example of an airline then in addition to the flight schedule people also started posting on their website the statuses of different flights. This is a big step forward because this is when people started moving from putting static content on the website to putting dynamic content on the website. Then it become more demanding and so demanding that they said that why don’t we create full-fledged applications which are hosted on the web and today that’s the reality. We don’t really have to go to the airline office to book the tickets. We can book them online. All that we get a etiquette and that etiquette if we show on the counter we will issue a boarding pass. Even there are provisions with certain airlines wherein we don’t even have to go to the counter to get the boarding pass. We can print the boarding pass right before 2 days of the flight. We can even choose a proper sitting which is appropriate to us. The moment we choose a suitable sitting, it immediately block for us. Means this is a full-fledged application. There is another set of applications that people are creating and exposing them as webservices. This services model is very appealing. For example, if we want to know the current conversion rate from US Dollar to Japanese Yen or Japanese Yen to Indian Rupees there are webservices available on the net. We have to make function calls within the service and we make these function calls from within our web application. Up-to-date information about the current conversion rate from one currency to another is available to us. This web service exposes lot of other things as well. Web service does not have a user interface. Services are available in the form of functions and we simply have to make call to those functions. The difference between the full-fledged web applications and the webservice is webservices never have any UI. So if we want to do any web programming we should look at two possibilities; one is the web applications and 2nd is the webservices.

Web Development Model

From the browser, we have to make a request to the web server saying that give us this particular page hosted on the website and on the server side the web server host series of pages that are there in the website and then there is application logic in the form of different classes. Each web page will have HTML + a few directives. Directives are some what similar to what we learnt in C programming as Preprocessor commands. These pages having HTML and the appropriate directives then interact with the classes and then the final outcome of this interaction is some kind of a HTML response. This HTML response is send to the client and then it gets rendered within the browser of the client. The content of the response is no longer static. It is dynamically generated. It is up-to-date and it is timely means it’s not stale. If we take example of some news channel NDTV then www.ndtv.com if they are to create a site then the content that is there on the site has to be updated every 15 minutes or at least every half an hour. As the news happens all over the world this content will have to be updated. Means they can’t really have static content. They have to have up-to-date content. It should be dynamically changeable and it should be timely. In the sense it should not become stale. Newer and newer contents should continuously keep getting updated on their particular news site. So content of the response is going to change dynamically. The HTML response that is generated, since it is going to have dynamic content it is impossible to create this by hand. Hand-crafted HTML content those days are over. We have to have programmatic elements within the
application which will dynamically update the contents. Unless and until we have that in place, it
would just a too much time for us to be able to generate a HTML response. So, one thing is content
goes to be dynamic. 2nd thing is these dynamic contents will have to be generated
programmatically and 3rd problem is that HTML response if we start hand-crafting ourselves, it is
going to take time. Realize that over the years as the web evolved, HTML has also evolved. So
whatever are the capabilities of HTML lets say 5 years ago, today the capabilities are very different.
HTML has evolved and made it possible to integrate videos with the web pages in a seamless
manner. Means technology is in place to achieve all this but then if we start learning all the HTML
involved in doing this, in that case focus is going to get lost. HTML allows to present the information
in a proper manner. We should be able to programmatically generate HTML because the moment
that we make this content generation of HTML programmatic obviously we can change it dynamically.
We are going to use programs to generate HTML. That programming ability will allow the content
dynamic and we will rely on wizards to be able to generate most of the boiler plate HTML,
readymade or routine HTML.

Web Development Tools

We can use either ASP.NET or JSP or JSF or PHP to build a web application. JSP is Java Server
Pages and JSF are Java Server Faces. So these are Java technologies which allow to generate any web
application. It might be ASP.NET. We have chosen ASP.NET as our tool for building a web
application. We have chosen ASP.NET as our tool because ASP.NET provides excellent
development tools. ASP.NET is a good tool for beginner. It embraces the beginner. It makes him
feasible to generate the website quickly. ASP.NET has a good debugger and has an ability to let us
create the applications and try them, test them and execute them on our own machine. With
ASP.NET we also get an integrated web server on which our site gets hosted. It provides the
intellisense which is in short a context sensitive help. If we are trying to generate the HTML
text as we are typing HTML, it will provide us with help. So intellisense not only saves the typing
effort but it also saves a lot of time on our part going back and forth between documentation and the
program. It becomes convenient once we start using intellisense with ASP.NET. Microsoft says that
its development tools are available to us free. We can download Visual Web Developer 2008 to build
all the applications that we propose to create in this course. This tool is legally free.

First Web Site

If we want to create a website then we have to launch Visual Web Developer 2008 Express Edition.
Once this is done, we need to go to the File menu. From the File menu, select New Website. It is
shown in the slide. The moment we select New Website, we will have four different options; one is
ASP.NET Web Site, 2nd is ASP.NET Web Service, 3rd is Empty Web Site and 4th is WCF Service.
We would select ASP.NET Web Site. Then we need to indicate that where do we want this website to
get created. To begin with we should always choose File System indicating that the website should be
created on my hard drive. Rather than putting it straightway on some HTTP web server or on a FTP
server, we would first want to test it, try it out on our own machine before we can think of deploy it
on any other machine. So File System is chosen as the location where the website should get created.
3rd option is the language in which we want to do the development. It would either be C# or VB.Net.
Then we can give the name to the website. Here it is website1 and then click on OK.

Wizard Generated HTML

When we do so, the wizard has created a Default.aspx file. .aspx file would always contain directives
and the HTML. It has been properly highlighted indicating what are the different tags present in this
page. This wizard created skeleton HTML would always been Default.aspx file. We can add any
HTML that we want to this particular Default.aspx file. If we want the content to be dynamic then we
can also manage to generate this content dynamically through a program. The moment Default.aspx
is created, one more file also gets created called Default.aspx.cs or Default.aspx.vb depending on
whether we are created the website using C# or VB.Net language. If we double-click on Default.aspx.cs or Default.aspx.vb, we will be taken to the C# file or the VB.Net file. In the HTML, in the 2nd line from the top, this Default.aspx.cs is mentioned. This file is known as code behind file. So we have a aspx file and a code behind file. If we double-click on this code file from the Solution Explorer then we will be taken to the actual code that has been generated in Default.aspx.cs or Default.aspx.vb.

Writing your code to generate HTML

Here we would see that there is a class that has been declared whose name is _Default and then : System.Web.UI.Page clearly indicating that this _Default class has been inherited from the Page class. Page class is the class present in the Framework Class Library present in the name space System.Web.UI. Then we have a method Page_Load( ) which receives two arguments; object sender and EventArgs e. Page_Load( ) is similar to main( ) in our C program. Within Page_Load( ), we need to add our own code which we want to get executed any time a request is send to this Default.aspx of our website1.

Actual Code

We would put a for loop here running from i = 0 to i < 10 i.e. 10 times and each time through this loop, we are going to display the latest value of i. To do so we would make a call to the method Write( ) present in a Response object. So when we say Response.Write( ), Response is an object and Write( ) is a method inside that object. For that to execute we need a namespace which is included at the top i.e. System.Web. All the using statements shown in the slide are inserted by the wizard. We had written <br /> in Response.Write( ) at the end. This is equivalent to a 'n'. The moment we put this here automatically the number that gets printed through Response.Write( ) would start coming in different lines. After that we would try to build the website by doing CTRL + F5.

Output

When we do CTRL + F5, we get the output shown in the slide. 10 different values of i being displayed. In the browser’s address, we can see http://localhost:1296/WebSite1/Default.aspx. Here 1296 is known as a port number. When you execute this program, this port number may not be same as shown here. The request for Default.aspx page is made and the moment we do so, the Page_Load( ) method is called and within that we had a for loop which wrote the response, which created the HTML and that HTML was sent to the browser and browser rendered it by displaying different values of i starting at 0 and ending up at 9.

Visual Web Developer IDE

As shown in the slide, at the top we get different tabs. Through these tabs we can move from one file to another. We can see the code behind file as well as the aspx file. On whichever file we click, that file’s content will be shown. Means we can simultaneously work with multiple files through ASP.NET. Then we have a Solution Explorer where inside Default.aspx, whatever different classes are there those are shown. Then there is a file called as web.config. We get to know all the details of the project in the Solution Explorer.
In this lecture you will understand:

- Genesis of HTML
- Contents of HTML document
- How to add HTML page
- How to manage tables
- How to generate dynamic content
Genesis of HTML

There are at least three things common to all websites. One of them is the location. Every single website will have some or the other Uniform Resource Locator. For yahoo, it is www.yahoo.com. For Google, its www.google.com. Each website will have some or the other Uniform Resource Locator means a unique way through which we would be able to reach that particular website. It’s known as the URL. Each one of them would be using some or the other transfer protocol. The one which is predominant is HTTP. When we send a request or even if we get back the response which is rendered finally in the browser, there will have to be some or the other rules that will have to be followed. Every response will have to be a HTTP response means the response that has been created as per the rules that have been laid out in the HTTP protocol. Likewise, when we make a request for a particular page on a website, this request also has to go in a standard form and that special way of making a request is the HTTP request. Means using the protocol or the principles that have been laid out in the HTTP protocol which for that matter stands for HyperText Transfer Protocol, we have to make the request. So the request-response model of the websites works on this HTTP protocol. Third is the publishing. All the content that is published is always in the form of HyperText and the language which is used for publishing the content is the HTML language. That is common amongst all websites. Information in today’s world is quite complex. Hence we have to have some ways or means to identify this information or put this information using the markup language like HTML through which we would want to publish this information. HTML can publish online documents; documents which are not only headings and text but also tables or lists or photographs. All these should be retrievable at the click of a button. We can design forms for searching some information on the web or we can do online reservations through a HTML page. We can put a spreadsheet in a HTML page, displays video within a HTML page or play some sound clips.

A Simple HTML Document

Typically at the beginning there would be a tag which identifies what is the type of the document. Following that there would name of the HTML standard that is being used to create this particular HTML file. Then there would be a tag known as HTML, inside which another tag is embedded called <head>. The tag is always written within <> symbol. Beginning of the tag and end of the tag has to be clearly identified. For example, as shown in the slide <html> and <head> are the opening tags and closing of these tags are indicated by </html> and </head> respectively. <head> tag is embedded inside the <html> tag. The header section is supposed to contain the title for every document. We would put a title within the header section and in this case the title is Untitled Page. To identify that this is a title, we again have to mark it up using a tag <title>. </title> is used to mark the end of the title. Following the header there would be a <body> tag and inside this <body> tag we want to put the content which we want to appear within the HTML page. We are putting simple text like This is my first html document and that is we expect should get rendered within the browser if we try to open the html file in the browser.

HTML – Important Concepts

- HTML even though it’s a markup language, it’s not a programming language. Means there are no loop, procedures and functions in HTML. HTML language is used for presentation.
- HTML documents typically have extensions of either .htm or .html.
- If HTML documents are opened in Notepad then we cannot see the actual presentation of header. We would see only the HTML document i.e. different tags, text inside those tags, etc. HTML document actually get rendered when we open them in browser. So HTML documents are displayed by web browsers like Internet Explorer or Mozilla Firefox browser.
• All documents must have a short title.
• All tags must be within `<html> </html>`.
• Other tags are optional.
• Must know some of the important tags that are used in HTML.

**Add New HTML Page**

We will try to add a new HTML page to our existing Website1. For that we start Visual Studio 2008 Express Edition and then open the 1st website i.e. Website1. When we do so, the Default.aspx page will be shown to us which contains some directives and some boiler plate HTML which is finally send to the client and rendered in the browser. We would try to add a new HTML page to our Website1 project. Right-click on the Website1 project in the Solution Explorer. The moment we do so, we will be asked from the different menu-items which one we would want to pick up. We say Add New Item. A dialog would appear. We want to add a new HTML page. The dialog asks for the name of the page. We want to add the page as Hello.htm and then click on Add.

**Make Changes**

When we do so, a new HTML page would get created for us. Now we want to make changes in that HTML page. In each HTML page along with the tag there are attributes. For example, in the `<html>` tag shown in the slide there is `<html xmlns="http://www.w3.org/1999/xhtml">` means the attributes are always organized in the form of name, value pairs. `xmlns` is the name and the value is identified by putting that value within "". There can be multiple name, value pairs for a single tag. Means multiple attributes can also be applied to a particular tag. Here each page has an attribute and it’s organized in name, value pairs; so its syntax is name = "value". We can think of adding an attribute to the body by saying `<body bgcolor = "green" >`. `bgcolor` is the background color. If we apply this attribute to the body tag then whatever is within the body tag when displayed in the browser would appear green in color. We can change the title from Untitled Page to **My first HTML Page**. Within the body tag we can add a single line **This is simple text**.

**Compile**

After having done this, we would do CTRL + F5 to compile the website.

**Result**

The result is shown in the slide. We can see that the changes that we have suggested in the previous slide have been made within the dialog that is shown in the slide. Title has now changed to **My first HTML Page** and in the body we have text saying This is simple text. So This is simple text is displayed within the web page. Request is made for the page Hello.htm and the address is http://localhost:1489/Website1/Hello.htm. So once the request goes for Hello.htm, its contents would be interpreted by the browser and suitably rendered within the window of the browser.

**Using Tables**

Tables come inside the `<body>` tag. Beyond **This is simple Text**, we are trying to add a table and that has to be inserted within the `<table>` tag. Inside the table we have to identify the row for that table. We are trying to identify the topmost row. In the `<tr>` tag, there is a Table Header containing Name and Telephone Number. Means in the very first row of the table, we are going to get the Name and Telephone Number. Below this Name and Telephone Number columns, we are trying to add some names and telephone numbers. For that in the next row the Table Data for the very 1st cell is Name of the person is Sanjay Jain and data for the next cell i.e. Telephone Number in the same row is +91 98450 23491. So one row contains the header, another row contains the table data. Both are marked
using the `<tr>` tag. We may add one more Name and Telephone Number. So we add Vinod Wakhare with telephone number +91 94221 12491. If we now try to make a request for this particular HTML page then the result obtained is shown in the next slide.

**Result**

We can clearly see that Name and Telephone Number are clearly identified as headers. So they are in bold. Then Sanjay Jain and Vinod Wakhare are the two rows of the table containing name and against each name suitable telephone number has also been shown.

**Static vs. Dynamic Pages**

- In 1990s
  - Most of the information was static.
  - Means changed rarely.
- In the 21st century
  - Most information on web is dynamic rather than static.
- Dynamic content
  - Will have to be generated by websites like BBC, CNN, Times of India.
  - Will have to be generated by websites like Yahoo, MSN, Rediff.
  - We should aspire to create dynamic website ourselves.
- Web programmer must be able to
  - Generate page content on-the-fly programmatically.
  - ASP.NET allows for that.

**Add Web Form – Hello.aspx**

Go to the Solution Explorer. Right click on that and then say Add New Item. Select Web Form from the dialog that appears. Give the name of the form as Hello.aspx and click on Add. Now we would want to programmatically create the HTML table and add rows to it.

**Create Table, Add row**

In the `Page_Load( )` method, we go on writing `Response.Write( )` statements. In each `Response.Write( )`, we are outputting a string which is a real html. So `<table>`, `<tr>` and `<th>` is given having header as Name and Telephone Number. In the next line we are giving the table data with tag `<td>` in the form of Name along with Telephone Number. Programmatically we are creating this table tags to represent a table which will finally be send to the browser. Then we compile and execute this website by doing CTRL + F5.

**Check Output…**

When we do CTRL + F5, we get in the output as shown in the slide, Name and Telephone Number as header. We get Sanjay Jain and +91 98450 23491 as the name and telephone number i.e. table data in the cells. We can appreciate that this is a dynamic content. But there is still some problem associated with this. 1st problem is that we have completely hard-coded this information within the program. 2nd is that there is no provision to work with the list of names and telephone numbers. We cannot modify them, delete them, cannot sort them, add, list them.
Building Dynamic Websites

In this lecture you will understand:

- How to manage a telephone contacts book
- What steps are involved in it
- How to perform various operations on it
Managing Contacts Book - Steps

In the 1st step, we must first lay out the requirements of this dynamically managed website. Unless those requirements are clear, do not begin with creating programs for the application. Otherwise people start building the application and then there is some confusion about the requirements. As a result of which people are required to back to the design board and make changes. Sometimes minor changes but sometimes very major changes might be required. So knowing the requirements of the problem that we are going to solve is a very important step. Once we understand the requirements clearly then we can think of designing classes to implement the requirement. In this application where we are trying to manage the contact book, in the 3rd step we are trying to add persistence to our application. Adding persistence means whatever contacts are there, we should be able to save them on the disk and they would remain there even if we close the application. Once that is done then we should really attempt to build the user interface.

Step 1: System Requirements

In every contact, we propose to maintain three items of information; name, telephone number and mobile number we are proposed to store in a contact. In addition to that there can be variety of other details that we may wish to store. But here we are using only three details. Then we would like to perform variety of operations on these different contacts. For example, we may want to display the list of contacts that are there in the contact book or we may decide to add a new contact to the address book. We can delete one of the existing contacts or modify the information present about a particular contact. So Add, List, Delete and Change are the common operations on any kind of data. Same operations we want to perform on telephone contact book.

Step 2: Designing classes

Contacts book contains multiple contacts. Information about variety of contacts can be stored in the same contact book. We have already outlined that what property each contact has. There are properties like Name, Telephone Number and Mobile Number. Whenever we attempt to create a class for maintaining different contacts this Name, Telephone Number and Mobile Number would expose in the form of a property. If we have the get and set clauses for these properties then we would be able to set a new value into this name or telephone number or mobile number or using the get clause of the property we would be able to retrieve the current value of name, telephone number or mobile number. We are propose to create a class Contact inside which we will have data and the properties that we just have said. The properties and operations that we propose to carry out, all of them would be managed by the Contact class. This Contact class must be in our website.

Create New Website - Contacts

We will create a website using File | New Web Site. The dialog will appear, select ASP.NET Web Site. The location where this website is created is the File System means on the local hard-disk this website is going to get created. Choose the language as Visual C#. Indicate the location where we want to create this website in the File System. Give the name as Contacts. Then click on OK.

Add New Class - Contact

Now we can try to add a Contact class to this particular website. No sooner we create the website, two files get created in it; one is Default.aspx and 2nd is Default.aspx.cs. To be able to add a new class to this website, we need to go to the Solution Explorer. There right click on website project name i.e. Contacts. The moment we do so, a new menu would appear. Select Add New Item.
The moment we select Add New Item, a new dialog would appear. In this dialog many items that we can add to a website are listed out. Out of these, we select Class. Then give the name for the file as Contact.cs. Then click on Add. So a new class would now get added. Before that addition happens, it will ask that Do you really wish to add this new class to the website and it will also suggest you where this class is going to get added. The name of the folder is App_Code. Inside this a new file is going to get created. So say that we want to create this in App_Code directory.

Result – Contact.cs

When we click on Yes, we will see that a Contact.cs file will go into the App_Code directory as shown in the Solution Explorer. The Contact class created is shown in the slide. Now we need to add some code to it.

Add Code To Contact.cs

Now we would add the code in the Contact.cs file. Inside the Contact class, we would have three protected members i.e. protected string m_name, protected string m_telno and protected string m_mobile. Then we have a 0-argument constructor called Contact( ) which will initialize all these strings to empty strings. In addition to 0-argument constructor, we would also have a 2-argument constructor which when invoked would receive name and the telephone number i.e. telno. The values that we receive are setup in m_name and m_telno i.e. name is set in m_name and telno is set in m_telno. m_mobile is set to empty string.

Slide 11

Now we would create properties like name, telno and mobile. Inside each property we have a get and set clause. In the name property, get will allow to return the current value of m_name whereas set will accept a value and set it up in m_name. Similarly in the telno property, get will allow to return the current value of m_telno whereas set will accept a value and set it up in m_telno. Same is for mobile property. get will allow to return the current value of m_mobile whereas set will accept a value and set it up in m_mobile. So, each of these three properties has a get clause and a set clause.

Step 3: Add Persistence

Now we need to add persistence to this project. Means whatever contact information we have, we should be able to save it to the disk and that saving can happen after the application terminates means once the application execution is over, the contact information should remain in the file in which we have managed to save it. While doing this persistence we have to realize that too much overhead should not be there as far as storing of the contacts or loading of the contacts are concerned. When we decide to store the contacts persistently, we have to do that on the disk. But how do we organize this data on the disk? There are three possible choices that are available at your disposal. One of them to store the data or contacts information in some simple text file. Another could be store this in XML format and the third possibility in which we can store all this contact information is in a Database. To begin with, we want to keep this contact information in a simple text file.

How contacts are stored in text file

In one single line, we would have only information about one contact. Means name, telephone number and mobile number all these three entities for one contact would be present in one line. Each field would be separated from the other using a comma (,) which will help us to separate the name from telephone number and telephone number from the mobile number. For example, as shown in the slide it gives the clear idea i.e. Rajesh Diwan,2246347,9853103449 and Mansoor Sheikh,45678902,9422145678. Each field is separated from the other using a comma. After first line, there should be the information of next entry in the next line. So we would devote one line per person in the text line. This way of organizing the data is also called as CSV format standing for Comma
Separated Values. It is very easy to read / write files in C# and can be opened in any text editor like notepad.

**Add New Class - FileContact**

We will create a new class FileContact. To create a FileContact class, go to the Solution Explorer, right click the project name i.e. Contact and select Add New Item. Then from the dialog that appears select Class and give the name of the class as FileContact. Then we propose to inherit this class from the Contact class. So all the properties that were in the base class will be available even to the derived class. Moreover we propose to provide the persistence support within the FileContact class. To carry out reading or writing of file, we use two readymade classes present in ASP.NET; one is StreamReader and another is StreamWriter. StreamReader class reads from a file whereas StreamWriter class writes to a file. Stream is a sequence of bytes that are either going into a file or coming out of a file.

**Add Code – FileContact.cs**

Within FileContact.cs, we have some namespaces at the top i.e. using System and using System.IO. System.IO is needed to be able to make use of the StreamReader and StreamWriter class. As shown in the slide, it clearly indicates that FileContact class is derived from the base class Contact. We have a 0-argument constructor which is empty. To achieve persistence, we should be able to write the contact information to the disk and to do so we will have a methods called WriteToFile( ) implemented within FileContact class. We have a 0-argument constructor which is empty. To achieve persistence, we should be able to write the contact information to the disk and to do so we will have a methods called WriteToFile( ) implemented within FileContact class. Whenever this WriteToFile( ) method is invoked, a StreamWriter object is passed to it which is collected in the reference sw. Then we will use this StreamWriter object and using it call a method called WriteLine( ) and into that write name, telno and mobile. These strings are separated using commas. All strings are concatenated and whatever is the resultant string that would be written to the text file in a comma separated fashion. When WriteToFile( ) is invoked, a particular record would get written into the stream whose reference is passed and collected in sw in the WriteToFile( ) function.

**Slide 16**

Now we will write the LoadFromFile( ) function. This function receives a StreamReader reference indicating from where we are proposing to carry out reading. We will use that StreamReader to make a call to a ReadLine( ) function. Whatever string that it returns, we will collect that in a variable line. We need to check whether line is null or not using if ( line != null ). ReadLine( ) would read only one line. If there are no more lines to read from a file then ReadLine( ) would return null. If line is != null means something remains to be read from the file which is associated with the StreamReader sr. Once we have read a line then we can split it across the commas. To do so, we will call a function line.Split ( ',', ). There are three fields per line. They would be split up into three parts. Those three parts are strings. Those strings are collected in list array which is an array of strings. Then we can split the line that we have read across the different commas that are there in the line. list[ 0 ], list[ 1 ] and list[ 2 ] after having split would contain name, telno and mobile. We would collect them in individual variables name, telno and mobile. Then return true saying that we have been able to successfully extract the name, telephone number and mobile number from a line that has been read from a file. If we are unable to carry out the reading means if we reach the end of file in that case return false.
Managing Contacts

In this lecture you will understand:

- How to manage contacts
- Design of classes to do so
- ArrayList collection
Managing contacts

When we decide to manage multiple contacts, we have to have the ability to store multiple contacts in a file. Last time we used a comma separated value file means a simple text file rather than a database or an xml file. Instead of getting into those complications, we said the CSV format is a straight format which is popularly used today. So we would try to store our address book in a text file using CSV. While managing contacts we would be required to add new contacts to that text file or we may be required to delete an existing contact from the text file. We can also search a contact within the contact book. We should also be able to save all the contacts into a file. We should also be able to load all the contacts from a file. We should be able to handle unlimited number of contacts. We should be able to maintain as many contacts as we want within the contact book.

Design

We have already seen the Contact class containing protected strings like name, telephone number and mobile number. We also had created a 0-argument constructor, 2-argument constructor and properties within the Contact class. Properties within the Contact class can either get or set the name, telephone number or mobile number. From the Contact class, we had derived a FileContact class inside which we had a 0-argument constructor and WriteToFile( ) function and LoadFromFile( ) function. WriteToFile( ) function write contacts into the file and then we are also able to load those contacts back from the file. This was done for one contact. Now its time to manage multiple contacts simultaneously. To manage those multiple contacts we need to create a list of contacts and to do so we will create a class called ContactList. In this ContactList class, to handle several contacts simultaneously we need to create an ability to handle multiple contacts and for that we propose to create an ArrayList. In the ArrayList, we would have references to several FileContact objects and each FileContact object is going to contain a name, telephone number and mobile number. Within the ContactList class we are also going to have a 0-argument constructor, a Search( ) function, a LoadFromFile( ) function and a SaveToFile( ) function. ArrayList is a collection that .NET makes available readymade to us. So rather than hand-crafting our own array to maintain several FileContact objects, we would try to make use of standard ArrayList collection to do so.

Add class ContactList - ContactList.cs

Create a class ContactList. To begin with we would have using System, using System.Collections and using System.IO. Inside the ContactList class, first create an ArrayList object whose reference we propose to store in m_list. Then we have a protected string m_path which is used to store the path of our text file. Then we have a 0-argument constructor which is empty. Then we implement the LoadFromFile( ) function. The purpose of this function is to be able to load contacts from the file into memory. For that first we create an ArrayList object by saying m_list = new ArrayList( ). Whatever path is passed onto this LoadFromFile( ) function we would store it in a member variable m_path. Then we would create a new StreamReader object and pass to it the path of the file which we wish to make use of. StreamReader clearly indicates that we want to read from this stream using the StreamReader object. Once the filename is passed onto the object, it would open the file then we would be able to read from that file. To carry out reading we will use an infinite loop saying while ( true ) and whenever we reach end of the file, we would break outside this loop. Inside the loop, we will create a new FileContact object and then using the reference of this FileContact object, call its method LoadFromFile( ). There are two LoadFromFile( ) functions; one which we are defining in the ContactList class and one is already present in the FileContact class. Then we would check if ( ! con.LoadFromFile ( sr ) ) then we would break outside the loop. Otherwise we would add the contact information to the ArrayList using Add( ) method i.e. we say m_list.Add ( con ). Once the control is outside the while means we had done all the contacts then we close the StreamReader by saying sr.Close( ).
Slide 6

In the SaveToFile() function, we would try to check out whether the file is existing or not. If it is existing then we propose to delete it. When we say File.Exists(), Exists() must be a static method inside the File class. Similarly Delete() must be a static method inside the File class. Then we would try to create a new StreamWriter object and to do so we will use m_path in which we have already stored the path of the file in which our contact information is present. File class is defined in the namespace System.IO. Once a StreamWriter object is created, we must start writing contact information using the StreamWriter into the file whose path had already been stored in the StreamWriter object. Number of contacts can be found using the Count property of the ArrayList class. So when we do m_list.Count, we would come to know how many FileContact objects are present in the ArrayList. Then we run a for loop and inside the for loop, we would first try to obtain the FileContact object. ArrayList is a generic class which can hold on array of any objects. So when we say m_list[i], we get the i th object present in the ArrayList. But since ArrayList is framework class, when it returns the reference through m_list[i], it will return the reference of the FileContact object. We need to specifically cast it before we collect it in a reference called con of the type FileContact. Then we would say con.WriteToFile(sw). Once control comes outside the for loop means we are done with writing all the contacts into the file using StreamWriter then we can close it by saying sw.Close(). Then we would create a property List and it returns the ArrayList collection. Inside this property we have a get clause which returns m_list. It is needed when we want to add a new contact or delete an existing contact or make changes in the existing contact or display list of all the contacts.

Search( )

We should carry out the search using the name. Moreover to be able to search, we should not be required to give the entire name, even if part of the name is given then also we should be able to carry out the search. What is to be searched is made available to the Search() function in the form of a string called needle. Now we can run through the ArrayList collection. We can find out the number of contact information present in the ArrayList using m_list.Count. Inside the loop, obtain the FileContact information using m_list[i] and cast it into Contact and collect the reference of that object in con. Then make use of two functions; one is Contains() and another is ToLower(). Whatever string is given to ToLower(), it converts it into a lower case alphabet. Contains() helps to check any part of the name. Once we have obtained from the ArrayList, the Contact reference then we can check if (con.name.ToLower().Contains(needle.ToLower()) ) and if that name is present in the current contact then we say return con. Otherwise we would return null.

More About "ArrayList"

ArrayList is a collection class. We should use it because it supports all operations like any Array. Arrays are fixed in size. ArrayList expands when it is needed. Our contacts book can contain any number of contacts. There is no upper limit on that. We do not have to bother about how many contacts are there ahead of time. Means while writing the program we need not know how many contacts we are going to deal with. ArrayList would be mature enough to tackle that situation as and when it arises. To be able to make use of this ArrayList collection, we need to add a statement using System.Collections.

After Adding Classes...

We have added three classes to the website i.e. Contact, FileContact and ContactList and they can be seen in the Solution Explorer as shown in the slide. Solution Explorer contains Contact.cs which is present in the Contacts directory which is present in the ASP.NET directory which is present on C drive. FileContact class is present in the FileContact.cs file again in the same directory. Lastly we
have ContactList.cs containing ContactList class definition which has the ability to maintain multiple contacts again in the same directory.
Telephone Contacts User Interface

In this lecture you will understand:

- Building user-interface
- Steps involved in building UI
- Testing UI
User Interface

When we decide to build the user interface we have to understand what operations are we expecting the user to perform while working with this website. We wish to carry out operations like display, add, delete and modify on our contact list. Out of all these, we would first concentrate on the Display. Whatever contacts we have, we should be able to display all of them on the screen. To do so, we will have some contacts with us. For that we will first create some text file in which we would have some minimum contacts and there onwards we would start displaying them. We would then add a new contact to that particular file, so on. Whatever information we display on the webpage, it should be possible for the user to read easily. Moreover the information that is present, we should be able to carry out updation of that whenever a new contact is added or an existing contact is removed or an existing contact is changed.

Design

The backend classes are Contact, FileContact and ContactList. We have created the Contact class having protected strings like name, telephone number and mobile number. We also have constructors and properties within the Contact class. From the Contact class, we have derived the FileContact class which has the ability to add persistence to this contact information. We were able to write the contacts into a file or load them back from a file. Then we had created a ContactList class which could maintain multiple contacts simultaneously in memory using an Array List collection. Inside the ContactList class, we had 0-argument constructor, the Search( ) function, LoadFromFile( ) and SaveToFile( ) functions. The LoadFromFile( ) function in turn call the LoadFromFile( ) of the FileContact class. Likewise SaveToFile( ) function of the ContactList class in turn had called WriteToFile( ) function of the FileContact class. Now to be able to display the information present in the contact book right now, we need to add a page called ContactsManage.aspx. So the user interface attempts to display all the contacts present in the contact book would begin with a creation of a new webpage which is known as a Web Form. Whenever we create this Web Form, two files get created; one is an aspx file containing html and the page directives and a code behind file which in this case is a C# file. So the file created is ContactsManage.aspx.cs. So displaying of information, we would manage through this Web Form.

Steps Involved

First we need to create and populate a data file called ContactsList.txt. Then we would add a new web page in the form of a Web Form and we will call it as ContactsManage. Then we will have to insert a table inside this web form and we need to do so by hand by editing the aspx file. Then we will try to populate this table with the contents that we read from the text file i.e. ContactsList.txt file. Population of the html table we propose to do through a program. Finally we are going to test what we have done by building the website and then making a request for a page i.e. the ContactsManage.aspx page such that we are able to figure out whether all that we had done is it working or not. If it is working then we should be able to see list of contacts displayed on a web page.

Step I – Create Data File

Go to the Contacts website. In the Solution Explorer, locate the App_Data folder. The App_Code folder contains all the cs files that we have created. Now we need to create a data file. So we right click on the App_Data folder of the Contacts website within the Solution Explorer. The moment we do so, menu shown in the slide will pop up. Select Add New Item from the menu that pops up. Then a dialog will appear from which we have to select Text File item. Give the name of the file as ContactsList.txt. Once we have done that, we will type out information about at least three contacts within it. Each contact having information within it like name, telephone number and the mobile number. So we would store information shown in the slide in this text file.
Result

When we do so, now we would find that in the App_Data folder, the ContactsList.txt is ready with the three contacts information. As shown in the slide, the result is going to look like once this ContactsList.txt is ready.

Step II - Add New Web Form ContactsManage.aspx

Now we need to add a new Web Form to our website. For that, once again go to the Solution Explorer, right click on Contacts website and from the menu that pops up select Add New Item. When we do so, a dialog will appear and select Web Form from that dialog. Give the name of the web form as ContactsManage.aspx. Then click on Add.

Result I (aspx)

The moment we do so two files get created; one is ContactsManage.aspx file and another is ContactsManage.aspx.cs file. This Result I shows the ContactsManage.aspx file. This ContactsManage.aspx file is added outside the App_Data and App_Code folders. It is clearly seen in the Solution Explorer.

Result II (Code-behind)

One more file i.e. the code behind file is get created. In the code behind file we have the using clauses at the beginning and a new class has been created called ContactsManage and it has been derived from the System.Web.UI.Page. Every single page that we have is always going to get derived from the Page class. Whenever we create a website, it always creates a Default page. That Default page is also derived from the Page class. So we have added a new Web Form that also stands derived from the Page class. ContactsManage is the code behind class for the webpage that we have just now added called ContactsManage. The idea behind having this code behind file is to be able to deal with different events that occur in a webpage. To be able to handle those events we are proposing to create a separate web page called ContactsManage derived from the generic Page class.

Step III – Insert Table

Locate the file ContactsManage.aspx. Inside this aspx file, we need to insert a table. First we would try to change the title of the web page. The default title is Untitled Page. So we will change that title to Manage Contacts. Then try to insert the table inside the body of the aspx file. Locate the body and then inside the <div> tag, try to insert the <table> tag. When we insert this <table> tag, this <table> tag has two attributes associated with it. One attribute is the ID of this table and another is runat. That runat attribute makes this particular table a control which will work on a server side. First give the ID for it. We have given the ID for the table as ListTable. Then we are going to ensure that there is not any duplication of IDs per page. Then we have marked it as a server control. The moment we marked it as a server control, it becomes possible for us to access it on the server side as an object. We propose to use this object to be able to change the contents of the table dynamically i.e. through our program. So we need to add runat="server" attribute.

Step IV – Populate Table

In the ContactsManage class, by default the Page_Load( ) function is provided. Whenever the page is going to get loaded, this Page_Load( ) function is going to get called. Within the Page_Load( ) function, we propose to call two more functions; one is AddHeader( ) function and LoadFromFile( ) function. AddHeader( ) function is going to add header to our table. We call a function LoadFromFile( ) which will actually reads the contents from the text file and as we read the contents of the file line by line, we will start populating the table by calling AddContactRow( ).
So we need to call AddHeader(), LoadFromTextFile() and AddContactRow() functions in Page_Load() function.

Page_Load()

In the Page_Load() function, we say ListTable.Rows/Add (AddHeader()). Header is the 1st row. We want the contact information as shown in the slide on our webpage. The header is Name, Telephone and Mobile. The headings are obtained by the AddHeader() function. AddHeader() function will return us the information which we will then add by calling the Add() function. To be able to call the Add() function, we will first say ListTable.Rows. Rows is a property in HTMLTable class. So ListTable is going to be an object of HTMLTable class. Then we call LoadFromTextFile(). This function helps us in reading the text file line by line and as we read each line, we will add its information into the table.

AddHeader()

The return type of AddHeader() is HtmlTableRow. So finally when we add a new row, it will be through the Add() function of the HtmlTableRow. Inside the AddHeader() function, we will first create an object of HtmlTableRow class. Once this is done, we will also create an object of HtmlTableCell and then setup the value inside this cell by using c.InnerHtml = "Name". Then we would try to add it to the HtmlTableRow by saying r.Cells.Add(c). Cells is the property of HtmlTableCell class. The way we have added the name, we can also add the telephone number. For that we need to first create another of HtmlTableCell and then populate it with the string Telephone and then we can add it to the HtmlTableRow. Exactly the same way, we may add one more column containing the string Mobile. So now the header is ready. This header is returned by saying return r. This will get added to the ListTable.

LoadFromTextFile()

First we create the ContactList object which is one of the backend classes. Then we call a function LoadFromFile() of the ContactList class. The file which we want to load, we have to clarify it. We have created ContactsList.txt file in which we have stored information about three contacts. The path of this ContactsList.txt file is made available to the LoadFromFile() function. To be able to get the path of ContactsList.txt, we have to call the MapPath() method. Server is the property of the Page class. Using that property we access the MapPath() function. This function retrieves the absolute path of the ContactsList.txt on the web server. We are saying "~/App_Data/ContactsList.txt" where ~ indicates the root of the website. We created the Contacts website in C:\ASP.NET\Contacts. But we actually physically deploy this file on the web server; this may not be the path. Whatever is the path that is assigned for our website, from there onwards we know that inside that particular path there is going to be App_Data and inside App_Data there is ContactsList.txt. But since we do not know what is the exact physical path, we just mentioned ~ which indicates root of the website. In this case ~ means C:\ASP.NET\Contacts. So we will pass the path of ContactsList.txt to LoadFromFile() function. Then it would be able to load that file successfully. Then we want to go through this file one after the other contact. Hence we run a loop for that. List is a property that we had exposed. This List property is used to return ArrayList in which all the contact information is present. When LoadFromFile() stands called, all the contacts that are there in file have been populated inside the ArrayList. So when we do clist.List we get an access to the ArrayList collection. Using the ArrayList collection, we use the Count property to come to know as how many contacts right now are present in the ArrayList. We will run the loop those many times. Each time through this loop, we should add a new row in the HTML table. To do so, we propose to create a function AddContactRow(). We pass (Contact) clist.List[i] to this AddContactRow() function.

AddContactRow()
In the `AddContactRow()` function, we are going to receive the Contact information in `con`. We want to add it to the HTML table. Create a `HtmlTableRow` object and a `HtmlTableCell` object. Now we can populate the information inside this cell. We say `con.name` to actually get to the name. Now we try to add the name of the person to the `HtmlTableCell` and that cell we are going to add to the `HtmlTableRow`. Whatever we do for name, exactly same thing we do for the telephone number and on the similar lines we will do same thing for the mobile number. Once all has been done, we should now try to add the row to the HTML table. We can do so by saying `ListTable.Rows.Add(r)`.

**Step V – Test**

For testing, in the Solution Explorer, right click on `ContactsManage.aspx`. The moment we do so, a menu shown in the slide would appear and from that select Set As Start Page menu item. Now we can do Build and run by doing CTRL + F5.

**Fruits of Labour!**

As soon as we do so, the output that is obtained is shown in the slide. Such a website would appear. Name, Telephone and Mobile are the headers that we had created. Then the table has been populated with 3 rows data as shown in the slide.
Contacts – Professional Style

In this lecture you will understand:

- Importance of presentation
- Different ways to control presentation
- Cascaded Style sheets
- How to use them for Contacts website
What Makes A Great Site

Now you must be visiting several sites and even when we take a casual look at any site, at times we have a very good feeling that yes, this is really very well done and at times we feel that no, this should have been much better presented. The information present there should have been much better. In fact, it is important to now get into the what makes it a bad site and what makes it a good site. If we look at www.orkut.com, we would find that the presentation of information there is fantastic. They have organized it very lucidly. Problem is that the content is very poor. Most of the people go on orkut to spend some time. These are all casual users. That does not make a good site. If we wish to have a long lasting appeal for the website then we have to have a good content and we also have to have a great presentation style of presenting that content then people get attracted to our site. At times we go to the other extreme where content is fantastic but it is not presented in a nice manner. Not that it could not have been presented but people think that it’s not important. But that is also a wrong idea. Its example is www.w3c.org. All the specifications of HTML, cascaded stylesheets and everything that we wish to do on the Internet, all of them are present at this w3c.org where w3 is World Wide Web. So anything and everything that we want to do on the Internet are available at www.w3c.org. They use fonts like Times New Roman which do not look good on the website. Even though these sites have very useful information, not very many people find it so appealing. Contrast these two; orkut.com and w3c.org, with youtube.com. We would find that good content and good presentation style that youtube uses, makes a great site. We should not get carried away by either of them. We should not go after creating too great content and we should not pay too much attention to the style in which it is presented. It should be reasonably good content and it should be a reasonable good presentation of that content which is found useful by the users of those sites. We should try to achieve a balance between good content and good presentation of that content.

Eye For Detail

When we try to present something, it is important that we should have a good eye for detail. Success of the website lies in the detail. Unless and until enough attention is paid to the details, the idea will never fly. When we are talking about the details of a presentation on a website, we have to look at what different fonts are being used at what places, what are the colors that are being used, how is the text being placed, what kind of alignment is being used, how the margins for the table are being build. These details are extremely important in deciding whether the user is finally like it or not like it. Good websites will always pay enough attention to these different details. That is what makes it a great presentation. If we wish to make a great presentation then it is suggested that we should never mix the content with the presentation. The way ASP.NET came up with a theme saying that HTML should not be mixed with the code. That’s why they let allow to create a .aspx file and .aspx.cs file. The way they manage to separate out the content from the business logic likewise content itself should be separated from the presentation style. Content is always going to be managed by HTML whereas presentation will have to be done using a technique known as cascaded stylesheet. Style is the way that we want to present. Sheet is imagined as a page which contains several styles listed out. Our need is to be able to separate the content from the way that content is laid out means how do we present that content to the final user. If we can separate out these two then we would definitely have a much better control over each one of them.

Example

Look at the table that we have shown in the slide. This table contains text like this is line 1, this is line 2 and this is line 3. This text is displayed in three different lines. In each of these rows, we are indicating that background color of the text is going to be according to the color number FFCC00. This is a hexadecimal number. Out of these 6 digits in the hexadecimal number, 14 two digits stand for the red color, next two digits i.e. CC stands for the green component and 00 is the blue. So red, green and blue are mixed up to get the final effect of that color. So we are specifying some
background color in which this text has to be displayed. Then we are also mentioning that the text be displayed in Arial font of size 2. Then we are saying that this text should be displayed in red color. So we also have a foreground color. Then we have this is line 1 displayed in bold face. Some similar attributes are also being associated with line 2, so also for line 3. Even though it’s a workable HTML piece, it will show the table according to the desired fonts, desired background color, desired foreground color, so on. This is not the best way of organizing it because we are trying to mix the content with the presentation. If we look at that table, we cannot identify what is being displayed because the text this is line 1, this is line 2 and this is line 3 somehow got lost into that. That happened because we try to mix the presentation with the content. That we should never let happen. So we need to separate that. When we separate the content then see the table as shown in the slide that we follow. It’s very clear that the text that we displayed is this is line 1, this is line 2, this is line 3 and then what style is being used for this text is defined in a subtext class in a CSS file. So the moment we are able to separate the .htm with .css in that case it’s much easier to present. At the same time pay attention to the content and pay attention to the details of the style. Because when we are looking at the cascaded stylesheet, we are concentrating only on the style. When we are looking at the .htm then we are concentrating only on the content. That way both become more manageable.

Our Requirements

We want the table header to be displayed in different color as compared to the body of the contacts. Means Name, Telephone number and Mobile number are our headings. They should appear in different color and the data for that should be in different color. Font of these three titles to appear in bold. So we want the output as shown in the slide. Page font should be easy to read. To do so, we are proposing to make use of a Font family called Tahoma, Arial and Verdana. Alignment of the text should be done in a justified manner.

Steps Involved

First we will create a Style Sheet with name Contacts.css. Then within the .css file, we will have to define the styles and we are proposing to use two styles head and body. body is used for page and head is used for header i.e. name, telephone number and mobile. Then we try to apply the styles that we define in the .css file to the content that we have on this webpage. Then we try to use style attributes defined in a css file in a programmatic fashion. Finally we would try to compile and test whether what we intended to achieve has indeed been achieved or not.

Step I - Add A Style Sheet

To do this creation, go to the Solution Explorer and right click on the Contacts project. The moment we do so, a menu shown in the slide would appear. Select Add New Item from the menu. When we do so, a dialog will appear and select Style Sheet from the dialog. Give the name as Contacts.css and click on Add.

Result

When we do so, the Contacts.css file will get created as shown in the slide. A body tag or a body style stands inserted within the cascaded stylesheet. The menu has also changed. One more option Styles is also available as shown in the slide. Using this Styles menu, we can add more styles other than the body style which we have right now. Then we have Contacts.css inserted within our project as shown in the Solution Explorer.

Step II – Define Styles

While defining the styles in a .css file the general syntax is

```
selector{
}
```
Within braces we have to specify what properties these selector enjoys. We indicate that using property : value pairs. In this case name of the selector is selector and it enjoys three properties; property1, property2 and property3 having values like value1, value2 and value3. Means within a selector, we can have multiple properties and their values, separate each pair from other using ;. As shown in the slide, we wish to display the Contacts table. Within the body style, we indicate the font-family property with values like Tahoma, Arial, Verdana and font-size property with value small. Then we have a .head style which is used for header of the Contacts table where we want to display the text in bold. So we use property font-weight with value bold. Then we have to indicate the color property with value #000099 i.e. blue color. Then we have a background-color property with value #ccffcc. Finally we have a text-align property with value justify. These two styles we want within our cascaded style sheet.

**Table Header Style**

When we create cascaded style sheet, body style was already there. So we can type the contents of body style. head style is not available by default. We need to insert it. To insert that .head style besides the body style, go to the styles menu and select Add Style Rule. When we do so, a dialog shown in the slide would appear. Give the name of the class as head. After typing the class name click on the > sign. When we do so, it will now go to the right hand side box. After that click on OK.

**Result**

When we do so, the head style would get inserted in the form of .head as shown in the slide. Inside this .head style, we need to add background color, foreground color, etc.

**Step III – Apply Style Sheet**

Go to the ContactsManage.aspx page where we want to apply the style. Locate its heading section and drag the Contacts.css file into the heading section.

**Result**

When we do so, in the aspx file we can see that besides title there is also a link tag get introduced. The link tag has certain attributes. For example, one attribute is href. Another is rel and third id type. These get added in header of our aspx file. &lt;link&gt; tag would be inserted any time, this html that we have right now is going to be connected with some other external source. In this case the external source is css file called Contacts.css. Appropriately href value has to be changed. Then we are indicating the relationship. We are saying that relation is stylesheet and inside the stylesheet we have text/css which indicates the type of the content. This is also commonly known as MIME type. The moment we drag and drop the css file on the ContactsManage.aspx, in its header section this link tag would get introduced and inside it there are attributes like href, rel and type each having a suitable value.

**Step IV – Add style attribute**

Go to the code of AddHeader( ) function and say r.Attributes.Add ("class", "head"). Css file contains styles. They are known as either elements or classes. head that we created was a class. Hence we need to add it to the table row. This will add the head class to the HtmlTableRow such that whenever that row is rendered, it knows what style should we actually use.

**Step V – Compile & Test**
When we compile and execute this application, the list of contacts that are displayed are more impressive than what we were able to manage last time around. The moment we were separated HTML content from the style or layout, it has definitely become more manageable. Not only that we can easily or effectively apply different styles to different parts of text that are being displayed on this webpage.
Cascaded Style Sheets

In this lecture you will understand:
- Types of CSS
- CSS Syntax
- What is cascading
- Benefits of CSS
Who Uses CSS

We have to be convinced that cascaded style sheets are in vogue and many popular sites are using them. If we want to figure this out then we can visit some sites and check whether they are using cascaded style sheets or not. Some of the visited sites are www.yahoo.com, www.timesofindia.com. Both of them are using style sheets. Then we also visited www.cricinfo.com which is a most popular cricket information site and we also visited www.orkut.com. All these popular sites make use of cascaded style sheets. If we want to find out whether they are using cascaded style sheets or not, we should right click on any page of any website. From the menu that pops up, select View Source. When we select View Source, it will start showing the HTML content of that particular page. Towards the beginning, we can find that there is a <link> tag or <head> tag inside which this css information is present. Whether an external style sheet is being used or an internal style sheet is being used will vary from one site to another. That will also dictate whether information about the css is there in the <link> tag or in the <head> section of the html page. We also visit the site www.e-chishiki.com is a Japanese site. chishiki in Japanese means knowledge. When we do right click on this site’s Home page then we found that they are indeed making use of css. Independent of the language, independent of the Geography, CSS seems to be a very popular way to give style to the content.

Types of CSS

CSS can be applied in three different ways. One of them through an external style sheet which we had done in Contacts website. We can also apply the CSS internally. For this we have to make an entry in the <head> section of the html document. The third method of applying the CSS is Inline tag method.

External CSS

External CSS we have used for Contacts website. The location of the style rules are done in .css file. We had created Contacts.css file for this purpose and then through the <link> tag we had applied the style saying that whatever styles are defined in the .css file they be used for rendering this particular html document. For example, we can say in the <head> section a <link> tag saying href = "mystyle.css" where mystyle is the name of the file. Relationship i.e. rel = "stylesheet" indicates that it stands for a style sheet and MIME type is indicated as type = "text/css". Once this is done then all the html content in this particular page would be rendered as per the style rules that have been defined in mystyle.css.

Internal CSS

In the Internal CSS, the style location is still the aspx file or a html file. Application of this is done in the <head> section and the way to do so is to say within the <head> section <style type = "text/css">. Then for a horizontal rule hr, we can say that its color is navy i.e. hr { color: navy; }. In its body, we want that the left margin would be 20 pixels i.e. body { margin-left: 20px; }. So we define the style right within the <head> section rather than saying within the <head> section that we are proposing to make use of an External CSS, right here we define the styles. When we do so, this will be useful only when a single page requires a unique style sheet because once we declare an external style sheet then all the pages within that website is going to be rendered using that particular external CSS file. So if amongst all the pages if we want that one particular page is such that we do not want external style sheets to be applied to it then for bringing uniqueness to that page we may define styles right within that html page or the aspx file in the <head> section as shown in the slide using the <style> tag.

Inline CSS
When we use inline CSS with each element we can really apply the style. For example, we may say `<p style = "color: yellow; font-family: verdana">` where p stands for paragraph. If we do this then in that case this particular paragraph would appear in yellow color and in a font verdana. This is the least flexible way of applying styles because this works specifically for the paragraph tag wherever it is being used. Rather than applying this to all the paragraphs, we are applying it for one particular paragraph within that page. So for every single paragraph, we will apply the same style again and again. If we really intend to do that then we had lost the advantage of using the css. This is so because for every single tag if we start applying the style inline in that case the very purpose of using the css gets defeated. So we should do this, only if we want to carry out overriding of external or internal styles.

**CSS Syntax**

In this we have to understand five things i.e. selector, class, id, div and span.

**Selectors**

Basic syntax for declaring a selector is to say `Selector { property: value }`. In the example shown in the slide, we are giving `ColHdg` as the selector name and `text-align: left` and `color: red` are two pairs of properties and values. This marks the basic syntax of css. `Property: value` together is also often known as declaration. We can also think of using the same declaration for multiple selectors. For example `h1`, `h2`, `h3`, `h4`, `h5`, `h6` are the standard tags for applying headings. If we so desire, we can apply `color: yellow` as the property: value pair for all these different headings. So when we start using it, we are saying `<h1> This is a level 1 heading </h1>` and `<h2> This is a level 2 heading </h2>`. For both of them, we are going to make use of the yellow color. The sizes of this are come out to be different but they are all going to appear in yellow color. We can also make use of descendant selectors. If we want to use descendant selectors we have to declare them by saying `p b { color: red }`. We are saying that wherever within a paragraph tag bold is used, it should appear red in color. But if we just use paragraph then it should not appear red in color. If we want to get this effect then in that case we can declare descendant selectors by saying `p b`.

**Selector Class**

Here the syntax is, for the paragraph tag or paragraph html element, we are proposing to apply different styles. For example, we are declaring `p.right` saying that text alignment should be right whereas when we say `p.center` then text should be aligned in center. While declaring the paragraph tag, we are saying make use of a class called right for this particular paragraph. Hence `This para will be right aligned would indeed be right aligned when it gets displayed in the browser`. If we say `<p class = "center">` then the text would be centrally aligned means for the same paragraph tag we are applying different styles. So right and center are known as selector classes. Now if we so desire that whichever tag makes use of center attribute, for all of them a particular style as defined by `.center` should get used. When we say `p.center`, it applies only to the paragraph tag but if there are other tags which have a center attribute then for all of them if we want the same style to be used then in that case we need not declare it as `p.center`, we just say `.center`. Here by saying `.center`, we are saying all the tags which has center as the attribute for all of them, this particular style i.e. center aligned style be used whenever the text is displayed using those tags. In the example, we are saying that `h1` and `p` i.e. two tags are used and for both of them we are saying class = "center". Hence the text that appears in `h1` and the text that appears in paragraph come out to be center aligned because we have defined `.center` rather than saying `h1.center` or `p.center`.

**Selector id**

Here we are saying `p#firstpara { text-align: center; color: red }`. If we are so desire that a particular paragraph within the html document, it should be applied the firstpara style whereas all other
paragraphs need not have the firstpara style. If we say p.firstpara then every single firstpara or every single paragraph would have been forced to make use of this firstpara style. If we do not want that and we want that only limited paragraphs should get this particular style then we can make use of this selector id. Then we have another issue wherein if we so desire, we can make our style sheet more descriptive by adding suitable comments to it. These are C style or C++ style comments wherein within /* and */ combination, we describe any note or remarkable about a particular style. In this case we are saying /* use this style only for emphasis */.

```html
<div>
  <div> allows to create customized block-level elements. The way to declare this is to say
  
  DIV.introduction
  {
    font-size: 14pt ;
    margin: 24 pt ;
  }

  While using it, we can say
  
  <DIV CLASS = "introduction">
    This is a simple introduction to the document
  </DIV>

  Wherever this division i.e. DIV occurs, this introduction style is get applied to it and whatever are the properties or styles we defined in the introduction those would become effective for that entire division.

  <span>
  <span> allows to create customized inline elements. Suppose we say
  
  SPAN.logo
  {
    color: white ; background-color: black ;
  }

  then it’s a style declaration. When we use inline CSS for example, in the paragraph tag we say
  
  <P>Welcome to the <SPAN CLASS = "logo"> KSET </SPAN> Web site. We want that the word KEST should appear white in color on a black background. So we are applying SPAN.logo to KEST alone. When we do so, we are doing this inline right where we are using a paragraph tag.

  Conflict

  We may create a style which is there in external CSS, it’s also there in the internal CSS. In that case, how this conflict is resolved is we need to appreciate. We also have to remember multiple CSS can also be included in one html document. If a particular style is there in the external CSS and is also declared internally in the head section then see how this conflict is resolved. For example, in the external CSS we might be declaring h3 { color: red ; text-align: right } whereas in the internal CSS in the head section we are saying h3 { text-align: center; font-size: 12pt }. There is no conflict between color and font-size because in the internal CSS we did not mention the color and in the external CSS we did not mention the font-size. In the text-alignment there is a conflict. In the external CSS, we are saying that text should be right aligned whereas in the internal CSS we are saying that text should be centrally aligned. Whenever we do this, the resultant style is going to be h3 { color: red; text-align: center; font-size: 12pt }. Here text-align: center got a higher priority. Now the fact that the two styles are being combined to create a third virtual style is cascading all about. We cascade several styles into one virtual style so it is known as a cascaded style sheet. While doing the cascading effect, it always give the higher priority to the internal CSS over the external CSS. The order in which this
precedence is applied is the lowest precedence is the default that the browser has. Still higher precedence is to the external CSS, still higher to the internal CSS and then highest precedence is to the inline style. Means right there at the point of using a particular tag, there if we want to make certain changes in the style that has the highest priority. Priority moves from left to right.

**Benefits Of CSS**

CSS is completely rule based. Since it is rule based, it is less error prone. It is very easy to write descent CSS. It leads to shorter development time. The advantage with external style sheet is that once we visit a page and the CSS gets downloaded there onwards every time the style elements do not have to be downloaded. If we apply the styles inline then every single time we go to a page that inline style will also have to be downloaded. Instead if we download the external style sheet once then all other webpages if they have been rendered using the same style sheet then downloading of the style elements get completely eliminated. That is going to lessen the network traffic and improve the performance. So download once and use often is good with external style sheets. We get much better control over the placement of elements because we can plan things in advance. We can reuse the same styles again and again throughout the website.

**Summary**

CSS has its own simple language. Best way to learn CSS is to start typing to change various properties and then try to make use of the intellisense that Visual Web Developer offers and try to build the style sheets. Do not create too many .css files. We should think before we act.
In this lecture you will understand:

- **Requirements** - Add new contacts
- **Steps involved**
- **Interaction of all classes involved**
Add Contacts – Requirements

Whenever we try to add a new contact, we should make a provision to let the user to make entry for name, telephone number and mobile number of a new contact. While doing so, we should see to it that the user interface remains simple and easy to use. We must also ensure that the fields that the user has to compulsory type have to be marked clearly when this webpage is displayed.

Steps Involved

First we have to make some changes in the ContactList class. Then in the ContactsManage.aspx page where we had listed all the contact information that is there in the Contacts text file, there we would try to insert one more table such that the user is indeed able to enter the new name, new telephone number and new mobile number for the contact that he wishes to add to the text file. Then we will also provide a Add button. Once the entries of name, telephone number and mobile number are done, to confirm that this is the contact information that the user intends to add, we would provide a button. The moment a button is clicked, its corresponding associated function should get called. That function is commonly known as a handler and clicking of the button is known as an event. In short the function that gets called whenever we click on the Add button is commonly known as an event handler. Then we will have to add the code to this function which facilitates addition of the new contact into the text file. Once we are done with all these then we want to compile and test the program.

Step I – Update ContactList

We have the Contact class which has the name, telephone number and mobile as the string variables. Then we have exposed these string variables through properties where each property has a get and put clause using which we can either retrieve the values present in these strings or store new values into these strings. We also have 0-argument and 2-argument constructors within the Contact class. Then we had derived a FileContact class from the Contact class wherein we provided functions like WriteToFile( ) and LoadFromFile( ). WriteToFile( ) helps to write a contact information into a file whereas LoadFromFile( ) helps to load the information that is there in the associated text file back into memory. Then we wish to manage multiple contacts and to manage multiple contacts we have the ContactList class. The ContactList class manages to maintain multiple contacts in memory through a collection class called ArrayList. So we have an ArrayList where we store references of different FileContact objects. For every contact, we will create a FileContact object and reference of this FileContact object, we store within the ArrayList. In this class we had exposed two functions; LoadFromFile( ) and SaveToFile( ) which inturn call WriteToFile( ) and LoadFromFile( ) of FileContact class. We also had a Search( ) function which can search for a particular contact in this ArrayList. We want to add two methods i.e. 1-argument constructor and the AddContact( ) function to this ContactList class.

ContactList – New Functions

In the constructor, it receives a string indicating the path of the file. Within the constructor, we are trying to create a new ArrayList object and then this address of the ArrayList object is collected in m_list which is a variable of the type ArrayList. Then we call LoadFromFile ( path ). Then we would create a AddContact( ) function which receives FileContact reference. Means any new contact is to be added to the text file, first we will call AddContact( ) function. Within this function we had, m_list.Add ( con ) which is going to add a new contact i.e. FileContact con into the ArrayList. Once this is done, this updated ArrayList we would now want to save into the file by calling the SaveToFile( ) function of the ContactList class. 1-argument constructor loads all the contacts information into the ArrayList whereas AddContact( ) function updates the ArrayList and then saves this updated ArrayList into the disk in a Contacts text file.
Step II – Create Table

Now we want to add a table to the ContactsManage.aspx file. Within the ContactsManage.aspx file, we already have some html along with the Page directives and below that we should create a line saying Please enter contact details.: As shown in the figure in the slide, it shows where this line would appear on the page. Subsequent to that we Name, Telephone and Mobile number along with three text boxes using which the user can actually type out the Name, Telephone and Mobile number. Once this is done then the user clicks on the Add button. As soon as the user clicks on the Add button, the handler for this Add button will get called. We will first create a table into which we have 6 cells, in these 6 cells we will put Name, Telephone and Mobile as the text strings and then the three empty text boxes. All these will now get inserted into a table of 3 rows and 2 columns. First we will add a table tag saying <table id = "AddTable" runat = "server" >. When we say runat = "server", it indicates that this particular table will get converted into a suitable object on the server side. In the first row, we have Name* where * indicates that it’s a compulsory field that the user needs to enter whenever he fills in the form. Similarly we are making Name and Telephone as compulsory fields. We want a input textbox besides Name*. For that we will add a tag <asp:TextBox ID="nameInput" runat="server"></asp:TextBox> where asp:TextBox indicates that it’s a ASP.NET control rather than a HTML control. This particular table row is over. In the next row, we should put the Telephone* as the table data and then in the next cell we should once again create a textbox <asp:TextBox ID="telInput" runat="server"></asp:TextBox>. On similar lines we will add one more row with text as Mobile and the cell <asp:TextBox ID="mobInput" runat="server"></asp:TextBox>. Now table is ready. Then we say <asp:Button ID="AddButton" runat="server" Text="Add"></asp:Button> i.e. the Add button is inserted.

Step III – Add Event Handler

Now we need to add an event handler for the Add button. Go to the Design view as shown in the slide. In the Design view we can actually see the table visually. In the Design view, we have to double-click on the Add button. As soon as we double-click on the Add button, a suitable handler will get created in a class.

Step IV – Event Handler Code

The class in which the handler for the Add button will get created is a ContactsManage class which is our code behind class. ContactsManage.aspx is the one which had the html tags, ASP.NET tags and the Page control. In the ContactsManage.aspx.cs the ContactsManage partial class already stands created which is derived from the Page class and listing of all the contacts that are there in the text file that logic is written in the Page_Load( ) method. So when we click on the Add button the AddButton_Click( ) event will get added to the ContactsManage class. Page_Load( ) and AddButton_Click( ) both are event handlers. So their parameters are similar. Both receives object sender and EventArgs e. Every event handler is going to receive the same parameters; sender object and EventArgs e indicating more information about the event. In AddButton_Click( ), we will first check out whether the name that is given and the telephone number that is given has it been typed in the form of some name and telephone number. Those are compulsory fields. We can check their length in the textboxes i.e. we check if ( nameInput.Text.Length > 0 && telInput.Text.Length > 0 ). If length is > 0 then it indicates that something is typed in those two textboxes and if it is then we call WriteToTextFile( ) function where WriteToTextFile( ) is a member of same ContactsManage class and this function is going to append a new contact information to the existing text file. Once this is done, we call a function UpdateListTable( ) which is going to display the fresh list of contacts in the ContactList.

Write Contacts
In the WriteToTextFile() function, we will first create a new FileContact object. Once this FileContact object is created, we will invoke the 1-argument constructor of the ContactList class. When we invoke this constructor, we need to provide the path of the ContactsList.txt file. To provide the path, we have to use Server.MapPath("~/App_Data/ContactsList.txt"). ~ will act as root. It will find out actual full physical path. Now the ArrayList collection is populated with current Contact list. Then we would try to pick up the values from three text boxes and once this text is picked then if within that text if there is , then replace it with space. This is important because finally our text file is going to be a CSV file where we have values separated by commas. Once this is done then we try to store that replaced name, telephone number and mobile number into con.name, con.telno and con.mobile. So we say con.name = nameInput.Text.Replace( ',', '  ' ), con.telno = telInput.Text.Replace( ',', '  ' ) and con.mobile = mobInput.Text.Replace( ',', '  ' ). Then we say clist.AddContact( con ). So this new FileContact we passed to AddContact( ), it is going to add to the ArrayList and the updated ArrayList, it is going to save into the text file.

Refresh List Of Contacts

Now we want to do that this new text file that we have written, we should read it back and display that new and updated list of contacts into the webpage. For that we had written a UpdateListTable( ) function. In this function first thing we do that we will clear out everything that is there in the ListTable. So we say ListTable.Rows.Clear(). Once this is done, we will add a new row to that ListTable and in that row we are going to add column headers using ListTable.Rows.Add( AddHeader( ) ). Then we call LoadFromTextFile( ) function which loads the ContactsList.txt file and display the updated list of contacts.

How It Works

The slide shows the ContactsManage webpage. When we click on this Add button, inside the ContactsManage class which is our partial class AddButton_Click( ) handler would get inserted. From this AddButton_Click( ), we call two functions; one is WriteToTextFile( ) and another is UpdateListTable( ). Both these functions are part of ContactsManage class. WriteToTextFile( ) 1st going to create new FileContact object and then it proceeds to create a new ContactList object. When it creates the new ContactList object, to the 1-argument constructor we had passed the path of the text file. That path would be used to load the ContactsList.txt and ArrayList is populated with all the contact information. Subsequent to this we had called the AddContact( ) function to add the new contact into the ArrayList and this AddContact( ) function having added a new FileContact into the ArrayList, it had called the ContactList's SaveToFile( ) function such that the updated ArrayList should get saved into the ContactsList.txt file. Then we had called UpdateListTable( ) function of the ContactsManage class wherein we had just prepared the table and then manage to load all the latest contact information from the text file and displayed that within the table that was already there.

Step V – Compile & Test

Finally we will do CTRL + F5 and compile and test the web page. When we do so, the webpage will look like as shown in the slide.
Get And Post Requests

In this lecture you will understand:

- Different working environments
- Request – Response model
- GET & POST Requests
- Anatomy of an URL
- Other types of HTTP requests
- What does a page header contain
Difference In Environments

The different environments in which we work are one is the development environment and another is the production environment. Development environment means we are trying to do the development of the website on the local machine and production environment means once the website is created, we deploy it on the web server remotely and then make request to that web server through the client machine. But to begin with, the server as well as the client both are on the same machine and that is the development environment. When we have listed the contacts or added new contacts, we did CTRL + F5. The moment we do CTRL + F5, our ASP.NET application gets build means compilation and linking takes place and once this is over, automatically the Visual Web Developer launches the browser i.e. Internet Explorer. Once this browser is launched, it sends request to the startup page of the local web server. During development phase this is important that the web server is present right on our machine such that debugging and testing becomes easier for us and a request is made by default to the page that we have marked as the startup page. So request would be made for this page, whatever response comes that would be rendered in the browser. But in the production environment, we do not do CTRL + F5 because CTRL + F5 we will do on our machine but website is not there on our machine. The website is deployed remotely on the web server. Moreover that website is already running on the web server. Hence we start our browser and then from our machine i.e. the client machine we send a request to the ContactsManage.aspx page or any other page that we wish within the website can be invoked by giving the aspx filename for that particular page. So send the request to ContactsManage.aspx because we had set ContactsManage.aspx as the startup page. Hence in the development environment that’s the one that would get requested from the web server whereas in the production environment, we can make a request to any of the pages. We would usually want to make a request to ContactsManage.aspx page. There is a Request – Response model is used. We send the request and then the response comes back from the web server and that response is always in the form of HTML. That response is properly displayed within our browser.

Request – Response Model

We always send a HTTP request from the client and a HTTP response comes back to us. HTTP stands for HyperText Transfer Protocol. HTTP protocol was invented by Tim Berners Lee. HTTP 1.0 had come into existence. The documentation of this protocol on which our websites work is available at www.w3c.org/protocols/RFC2616. RFC stands for Request For Comments. 2616 is a document number which contains the details of HTTP 1.1 protocol. HTTP protocol is a text-based protocol means the request goes in the form of text and response also comes back also in the form of text. This protocols talk about how a web server would communicate with a web client. How this communication happens is the HTTP protocol all about. When a request comes from the client, a connection needs to be established. Once the connection is established then we can request a data from the server. When that request comes, the server responds to that request by returning suitable HTML content and once that is over, the connection is immediately closed. Means when we make a 2nd request, there is no trace of the 1st request. That is over because the moment response is send, the connection is closed. So we say that HTTP protocol proposes a stateless model clearly indicating that there is no state for that particular client kept on the server across two different requests. Means once the request is made, before the client makes the 2nd request, everything about the 1st request is gone. That is a stateless implementation of the protocol.

Difference In Requests

When we make these requests, for listing we just make a request; we are not going to send any data to the server for that. As against that, imagine if we are going to do addition, searching or deletion, we will have to pass data to the server. For example, while adding we will have to say that add this particular data to the existing contact list. When we want to search a contact, again we will have to send some information saying that this contact we want to search or when we wish to delete, again
we will have to pass the contact information saying that this particular contact is being deleted from the contact list that is maintained on the server. Means in addition, deletion and searching, data has to be passed to the server whereas when we are listing, we are not sending any data to the server. So for listing we will make use of a HTTP Get request whereas for addition, searching and deletion, we will make use of a Post request. These are two very commonly used HTTP requests.

**Which To Use When**

GET request is always the very first request. Whenever we visit the webpage, Home Page that time, always GET request is done. GET request does not mean that we are going to only get data from the web server. For the 1st request we can appreciate that only data is to received from the server. Hence a GET request is made. But that should not make a feel that GET can be used only to receive the data and not sending it. That would be an incorrect belief. GET can also send small amounts of data back to the server. It can also send bigger amounts of data but if we try to do that then the request that we make that becomes clumsy to look at and secondly for sending more amount of information, there is already a POST request also available to us for sending data, whether it’s small or whether it’s large. Typically in Form based applications, we may have small amount of data or large amount of data to be sent to the server. In the Contacts website there is small amount of data to be sent. So we can send this data using GET request and it can also be send using the POST request. But typically for sending data, the POST request is used. Hence for addition, deletion, searching, for all of them we are propose to make use of a POST request. 2nd utility of the POST request is whenever we wish to carry out multiple interactions with the user on the web page, that time also POST requests are quite handy. That is the reason why Visual Web Developer or Visual Studio uses POST request as the default request. Rather than GET as the default, POST is used as default. Suppose we want to delete a particular contact from ContactsList.txt. To be able to do so, 1st we search for the contact and then we should delete it. Hence, on our webpage there will have to be two buttons now; one for searching a particular contact and 2nd for carrying out the deletion if the search in the previous case was successful. Whenever such multiple interactions are required on the webpage that time a POST HTTP request is popularly used.

**GET / POST Request**

When we make the GET or POST request, the request is typed within the browser. For example, as shown in the slide there is http://localhost:1827/Contacts/ContactsManage.aspx. This is known as a URL i.e. Uniform Resource Locator. When we mention this URL and hit the Enter key, this URL gets converted into a GET request or a POST request. Here we are saying GET/Contacts/ContactsManage.aspx HTTP/1.1 Host:localhost:1827. The URL that we mentioned that is not the one which is send to the server. The URL is first converted into either a GET request or a POST request. For listing out contacts, we said that there would be a GET request. So here we are saying GET/Contacts/ContactsManage.aspx. But this may as well be a POST request. If we click on the Add button or the Search button or the Delete button that we intend to put on the webpage, we will have to send data; that time the request that we make, the URL that we type in the browser will get converted suitably into the POST request. So GET and the POST requests that we pass, ultimately get passed on either to the local server, if it’s a development environment or it would get served to the web server on the remote machine, if it’s a production environment. Means if it’s a ongoing site to which we are making a request then it will be passed on to the web server of the remote machine. So either the local machine’s web server or the remote machine’s web server depending upon whether it’s a development environment or a production environment. All the request we are going to go only to the web server either local or remote. Web server checks the extension of the request that we have made. If it’s a .aspx extension, in that case our request is going to be passed on to the ASP.NET runtime for this request to get suitably processed. So whether it’s a GET request or a POST request, our URL is first get converted into that request and in the request there a GET or POST is written right at the beginning. When we create the List webpage or when we create Add that time how is it...
differentiated whether it’s going to be a GET request that is going to happen or a POST request that is going to happen. When we make the request, in the aspx page the method is written. If its method = GET then a GET request is made. If its method = POST then a POST request is made.

**URL Anatomy**

In the URL http://localhost:1827/Contacts/ContactsManage.aspx, http:// stands for the protocol that is being used for making the request. It can be ftp:// or file://. It all depends upon which protocol we are using. After http://, we are giving the name of the server. localhost means the local machine or development machine. In place of localhost there might be a name of the machine which is the remote machine where the websites stand deployed. Then we have the port number which in this case is 1827. Then we have /Contacts/ContactsManage.aspx which indicates the location of that webpage on the web server.

**Significance of URL Parts**

Whenever the URL is created, at the beginning we had http://. http stands for Hyper Text Transfer Protocol. Following http://, we had localhost which is the name of the local machine. This is the unique name for the server machine because ultimately localhost gets converted into the IP (Internet Protocol) address of the machine. Every machine on the Internet is identified using a unique address which is the IP address of the machine. We also have to specify the port number which in this case is 1827. Port number can be any number. It is not necessarily as 1827. Server is not only going to receive request from our browser, its going to get request from variety of browsers used by variety of people. So server handles multiple connection requests. Using the port number, which web application we are talking about is identified. If that is the one that we want to use then we have to indicate that through the port number because there might be several applications hosted on the same machine. Different applications work on different port numbers. To identify one application from the other, the port number is used. Subsequent to that we had /Contacts/ContactsManage.aspx. It indicates the location of the page that we are requesting within the website. We are requesting for the page ContactsManage.aspx which is inside the Contacts directory. We had C:\ASP.NET\Contacts, Contacts become the root of the website. Then we have a webpage and the code behind file i.e. ContactsManage.aspx and ContactsManage.aspx.cs. Means the URL mainly consists of four parts; 1st is the type of the protocol then the name of the machine where we are trying to make a request, followed that the port number and lastly the location of the file that we are trying to make a request for.

**Deletion Steps**

When we carry out deletion, it’s a collection of requests and not only one request. For example, we would type out the URL in the browser. When we do so, it will be a GET request. The page would get loaded. What is going to be loaded into the page is going to come as a HTML response for the GET request that we made. So when we are carrying out the deletion 1st we are going to show a list of existing contacts. To get that list of existing contacts, we just have to type out the URL of ContactsManage.aspx. That would become a GET request and its response will be rendered in the browser. Once that response comes, if we decide to delete a particular response, we must first search whether it is there or not. To do that, we will have to make a POST request. When we make a POST request, a response would come back which would indicate the result of the search that we had tried to initiate. Means we are trying to delete a particular contact, if its name is present or if its telephone number or mobile number is present, it would be displayed to us before we go ahead with the deletion. But if the contact that we are trying to delete is not there in that case a suitable response will have to be send to the client saying that such a contact does not exist. Means for searching also we have to make a POST request and then get back a POST response. Once that is over, we are going to click on the Delete button. That time also a POST request would be send because here too we want to send data that this particular item be deleted from the contacts list. Hence for that request a POST
response is come back to us. So we are talking about one GET request and two POST requests. This is a scenario where there is a multiple request-response interactions are happening on the same webpage. There must be an ability to correlate these different multiple request-response interactions. To do so, ASP.NET provides a concept of sessions. HTTP is a stateless protocol. If there are multiple requests that are happening then there is no way for us to ensure that our information is remembered on the server when we make the next request i.e. when we make the 2nd request, information about the 1st request is never going to be maintained by HTTP protocol. But that is a requirement. In our example, there are multiple POST requests coming from the webpage and we want that when we make the 2nd request, something about the 1st request should be memorized on the server. To provide this facility, ASP.NET provides a concept of sessions. This session is able to remember things across multiple POST requests that we make.

Types Of Requests
Apart from the GET and POST request, we also have HEAD, OPTIONS, DELETE, PUT and TRACE requests. GET request’s job is to fetch a document from the website whereas POST is used for sending the data. HEAD is used to get only the header information and not the body of the response. When the response comes back, header as well as the content comes. If we want to get only access to the header then we have to make use of the HEAD request. Then there is a OPTIONS request. When we use OPTIONS request, we can come to know what different types of requests are being supported for a particular application. DELETE deletes a particular resource or a document from the given website whereas PUT puts a resource or a document on the given URL. TRACE is used to figure out the request that we are sending to the server, if is not going to the server, if it is passed through several proxy servers or gateways, in that case is our request going to get modified before it finally lands with the web server or not, if that we want to figure out that time we want to make use of a TRACE request. It is kind of a loop back. TRACE request is useful in debugging if something is not working for the website. These many requests can really be made.

Understanding Page Header
We have something is written within % and %. Whatever is written within <% and %> is useful for the server. Then we have a Page directive or Page headers. Within this Page header, we have different Page properties i.e. Language, AutoEventWireup, CodeFile and Inherits. These properties are written in the form of name, value pairs. So ContactsManage.aspx does definitely have apart from the HTML, this particular page directive. In this Page directive, the properties that we mention are going to be used by ASP.NET runtime, commonly known as ASP.NET engine. Language="C#" indicates what language has been used in the code behind of this particular webpage. We can use only one language per page. We cannot use multiple languages in the code behind page. Then we have AutoEventWireup="true", when we set it to true, automatically the Page events would be bound to the appropriate handlers. By setting it to true, only limitation is the name of the Page handlers become same. So wiring of the event and the handler or connection of the event and the handler happens automatically when we have AutoEventWireup="true". Then we have to clearly indicate the code behind file. In our case the code behind file means ContactsManage.aspx.cs for the webpage ContactsManage.aspx. ContactsManage.aspx.cs contains a partial class. This is responsible for the event handling mechanism of ASP.NET webpage. ContactsManage.aspx.cs contains code for handling variety of events. There are many more attribute possible in Page header.
Delete Contacts

In this lecture you will understand:

- Delete existing contacts
- Steps involved
- What are sessions
- How are they managed
Delete Contact - Requirements

While deleting a contact, we should be able to search for a particular contact using the name of the person. We would provide two buttons for actions. The action involved is either the search or the delete. When we do search, we will find out the contact using the string which is given as input by the user. Deletion will carry out elimination of the first matching contact. If there are two Ramesh present within the contact book then at least to begin with, we will try to delete only the first Ramesh present within the Contact book. For carrying out search, we would have to provide the input and for doing that input, we would try to make use of the same table which we had used for entering data for a new contact. The name of that table we had given as AddTable. Then we should also be able to display the information about the contact that is to be deleted because unless the user is able to read the entire contact information, he should not be forced to take a decision whether he wishes to delete that contact or not. For that we will have to make a provision within our program. Once the information about the contact that has been searched is displayed properly then the user clicks a Delete button then we should go ahead and eliminate that contact from our contacts book.

Steps Involved

First we will have to update backend class i.e. the ContactList class. Then we will try to insert Search and Delete Buttons within the webpage i.e. ContactsManage.aspx. Once the buttons are created, we will try to provide event handlers for these buttons. Once these event handlers are created, we will have to add code into these event handlers. Finally we will build and test whether this webpage is working all right according to our expectations or not.

Step I – Update ContactList

We are working with 4 classes, out of which 3 are the backend classes. One of them is Contact class from which we have derived the FileContact class. Now we will try to concentrate on the ContactList class. While carrying out the addition, we had added 1-argument constructor as well as the AddContact( ) function to the existing ContactList class. Now we are going to add one more function called DeleteContact( ).

ContactList – New Function

The DeleteContact( ) function is going to receive a Contact object because DeleteContact( ) must obviously be able to decide what contact to delete and that can be figure out from the parameter that it receives in the form of Contact c. To carry out the actual deletion, we will first search for this contact according to the name of this contact. c.name will give the name of contact. We would invoke the Search( ) function present within the ContactList class which would return that particular contact’s details which we would collect in Contact con. So we say Contact con = ( Contact ) Search ( c.name ). Once this is done, we would try to eliminate that contact from the ArrayList by saying m_list.Remove ( con ). Once this contact has been eliminated from the ArrayList, we will try to save the balance ArrayList into the ContactsList.txt file. So we call SaveToFile( ).

Step II – Insert Buttons

To insert the buttons for Search and Delete, we need to go to the ContactsManage.aspx file. Into this aspx file beyond the directives that is already there, we should now try to insert the Search button. The way we had inserted the Add button, just besides that the Search button would get inserted. For that we use asp:Button tag where ID we are giving as SearchButton, text on the top of button is going to be Search and runat = "server" indicates that this would become a control when the request comes from the client. On the server side this Search button is going to work like a ASP.NET control. Then we say <br/> Writing <br/> is same as writing <br/></br>. We are trying to insert two blank lines in between the Search button and the text line which says Contact to be deleted:. After the text
Contact to be deleted:, we once again put two blank lines using <br /> <br />. Now we would put a label by saying asp:Label. The way there is a button control, similarly there is a Label control. We will give an ID as DelContact and make it a server side control using runat = "server". Whenever the search is over, if the search is successful, the information about the contact that is to be deleted should be done on this particular label. Subsequent to that, we want the Delete button. To get that Delete button, we once again leave two blank lines. Then we have asp:Button tag where ID is DeleteButton, text is Delete and set runat = "server". We will also set Enabled = "false" because to begin with, we do not want this Delete button to be available for selection and we can ensure that by saying Enabled = "false". Later on we will enable it. By enabling, it will not appear gray in color. So it will be available for selection. As shown in the slide, the Delete button is disabled.

Step III – Add Event Handlers

Once the aspx file is ready now we need to add event handlers for the Search and Delete buttons. For that go to the Design view of the aspx page and then double-click on the Search and Delete buttons. Now two handlers would get inserted within the ContactsManage.aspx.cs file. Inside the ContactsManage class, the Search button handler and Delete button handler would get inserted.

Step IV – Search Handler

In SearchButton_Click( ), we have two parameters, object sender and EventArgs e. Inside the SearchButton_Click( ), we have to find out whether the name has been indeed be typed or not means we are validating the data before the search proceeds. So we say if (nameInput.Text.Length > 0) and if it is then we first create a new ContactList object. Once this ContactList object is created, we would use it to call a method called LoadFromFile( ) and while loading from the file, we will have to provide the path of the text file. The text file ContactsList.txt is present in the App_Data folder. LoadFromFile( ) would load the contents of the ContactsList.txt and populate the ArrayList. Once the ArrayList is ready, now we are able to search in that ArrayList. So we call Contact con = (Contact)clist.Search(nameInput.Text.ToLower( )). We are carrying out search on the basis of the name that has been entered. This method will check that the name that we are passing in lower case is present in the ArrayList or not. If it is there then that contact information would be returned and collect that in con. Once we have that information with us then we check whether its null or not i.e. if (con != null) and if it is then we need to display the information about that contact on the webpage. For that we say DelContact.Text = con.name + "," + con.telno + "," + con.mobile. Once this text is displayed, we would say Session["contact"] = con. When we do the search, search does not happen on the client machine. Search happens on the server machine. Once the request goes to the server and the contact is indeed searched then we saw that whenever the HTTP response comes back the connection is closed. The state for the client request is never stored on the server. Means by the time we actually try to delete information about what we search is gone. If we want to memorize that information then we have to make use of a session. Session is a reference to an object of HttpSessionState class. It stands declared in the base class. When we say Session["contact"], we are using that contact as an index into the session array and into that we are trying to memorize the contact that has been searched. That particular contact which has been searched, its information would be stored in the session. Once stored in the session object, next time when we make a request i.e. when we click on the Delete button, we would be able to use this information on the server side. Means it has been made possible through session to remember about the last request. Means state of the client has been preserved in the server. The mechanism to do so is HttpSessionState class’s object i.e. Session. Then we say AddTable.Disabled = true because once the search is over the AddTable would be disabled. In addition to that we should also say DeleteButton.Enabled = true i.e. Delete button is now enabled.

Step IV – Delete Handler
When the user clicks on the Delete button, this DeleteButton_Click() handler would get called. The same parameters are received in the DeleteButton_Click() button handler. Inside this we now try to make use of the information that is saved in the Session object and we can retrieve that by saying Contact oldCon = (Contact)Session["contact"]. Session we can treat it as if it's an array and contact becomes an index to that array. Whatever is stored in Session object is collected in oldCon. Then we create a new ContactList object then try to load the existing contacts that are there in ContactsList.txt by calling the LoadFromFile() method. Once the ArrayList has been populated with existing contact information, now we can call DeleteContact() method in the ContactList class by saying clist.DeleteContact(oldCon). The change that happened in ArrayList has to be reflected in our webpage. To do so, call UpdateListTable() method. Then we say AddTable.Disabled = false i.e. the table is enabled. User should be able to add new information or should be able to carry out new search. Then make the Delete button disabled i.e. DeleteButton.Enabled = false. Then the text which is written on the label is to be emptied using DelContact.Text = "".

**Step V – Build & Test**

Build the application using CTRL + F5 and test it. When we build the application, we would see the webpage ContactsManage.aspx along with the existing lists of contacts present in the ContactsList.txt file as shown in the slide. Now we need to search for a particular contact. To do so, we type name of contact that we wish to search i.e. Rajesh and then click on the Search button.

**Search Results**

When we click on the Search button, Search button handler would get called. This handler would search for Rajesh contact and then try to display that information below Contact to be deleted:. In the label the information about Rajesh now would appear. Now the Delete button is enabled. Search table is now disabled. To delete this contact, we will now click on the Delete button.

**Updated contacts**

When we click on the Delete button, the Delete button handler would get called. We had deleted this particular contact whose information is kept in the session. Now as shown in the slide, the Rajesh contact has been deleted from the contact list and there are only two contacts remaining in the list.
Change Contacts

In this lecture you will understand:

- Change existing contacts - requirements
- Change existing contacts - procedure
Requirements - Edit Contact

While editing a contact or while changing a contact or while modifying a contact the user must be able to select the contact that we wish to edit. The user should be able to change all the fields i.e. name, telephone number and mobile number. Editing should be easy to perform and the contacts list should be properly refreshed the moment editing is over.

Steps Involved

First we will update the backend class called ContactList. Then we would try to insert a new table along with a button. We will call this table as Edit table wherein the user should be able to make changes in name or telephone number or a mobile number and once the editing is done the user should click on the Save button. All these insertions of the suitable tags for the ASP.NET control is done within ContactsManage.aspx file. Then we will also have to suitably enable / disable the Edit table after the searching or the deletion. Then we would try to add save event handler. Once the event handler is ready, we will put code into it to carrying out actual saving of the contact list. Finally build and test whether the functionality is being implemented as per our expectation.

Step I – Update ContactList

As shown in the slide, in the ContactList class there is a UpdateContact( ) function. We had a 1-argument constructor and the AddContact( ) functionality when we wanted to add a new contact. DeleteContact( ) was added when we wanted to delete the existing contact. Now we wish to add UpdateContact( ) because we wish to modify an existing information.

ContactList – New Function

The UpdateContact( ) function receives name, telephone number and mobile number. Along with that it also receives the original name. So the parameters are string original_n, string n, string t, string m. Inside this function, we would first search for this name. For that we will call a Search( ) function. So we say Contact con = ( Contact ) Search ( original_n ). This Search( ) function is going to traverse the ArrayList and find out where is this particular name. Search( ) would return the contact information. Now the name, telephone number and the mobile number that we had received, we set that up in Contact con. Once they have been setup, we will have to call a function SaveToFile( ). SaveToFile( ) function saves the updated ArrayList into the ContactsList.txt file.

Step II – Insert Edit Table

Now we will try to insert the Edit table as well as the Save button through the aspx file. We would try to lay Name, Telephone number and Mobile horizontally. We would try to put a textbox below each one of them and a save button below the textboxes. We insert a new table using a <table> tag. We will give the id as EditTable. Then we are saying visible = "false" indicating that to begin with, we do not want this table to be visible. Then runat = "server" indicating that this is going to be converted into an object on the server side. Then we will describe the rows of this table. We have two rows. In the 1st row, we are going to have the table data as Name*, Telephone* and Mobile. These are table data tags. In the next row we have three ASP.NET textbox controls. So 1st declare the table row and inside the table row, in each table data, put the ASP.NET textbox control. ID for the 1st is editName, for the 2nd is editTel and for the 3rd is editMob and for all the three we set runat = "server" i.e. making them a server side control. Now the EditTable is ready. Now we want Save button below that. For button we have the ASP.NET Button control by saying asp:Button. Its ID is SaveButton. Give the text on the button as Save and make it also an object on the sever side i.e. runat = "server". Visible = "false" indicates that the way we are disabling the EditTextable to begin with, we are also disabling the Save button to begin with.
Step III – Enable / Disable Edit Table

Go to the SearchButton_Click( ) handler. Inside this existing code is already there. Locate DeleteButton.Enabled = true and then make a call to ShowEditTable ( con ). Into the EditTable, which contact information is to be displayed that is passed to ShowEditTable( ). Then in the DeleteButton_Click( ) handler, existing code remains as it is. Below that we locate DelContact.Text = "", EditTable.Visible = false and SaveButton.Visible = false means we are disabling the EditTable and the Save button. The ShowEditTable( ) method receives a Contact information. It sets up the value present in the con object into the three cells of our table by setting editName.Value = con.name, editTel.Value = con.telno and editMob.Value = con.mobile. Then makes the EditTable visible by EditTable.Visible = true and the Save button is also made visible by SaveButton.Visible = true.

Step IV – Add Save Handler

Go to the Design view of the aspx page and then double click on the Save button.

Step V – Save Handler

When we do that, the Save button handler would be created. The handler for the Save would be created in ContactsManage.aspx.cs file. This is done by the wizard for us. Name of the handler is SaveButton_Click( ) and it receives object sender and EventArgs e. While carrying out saving 1st we will try to get the old contact information that has been saved in the session object and we extract it by saying Session [ "contact" ]. Session object can be accessed in an index fashion. The extracted information is casted into Contact and store it in Contact oldCon. Then create a new ContactList object. Once this object is created load all the information that is there in the ContactsList.txt file into the ArrayList in memory. To do so, we say Server.MapPath ( "~/App_Data/ContactsList.txt" ). This gives the physical path of ContactsList.txt file and from that file LoadFromFile( ) would read the contact information and populate the ArrayList. Now we would call a backend function called UpdateContact( ). We pass old contact’s name to it, new name, new telephone number and new mobile number to it. The modified contact is the one that would be passed to the UpdateContact( ). So old contact’s name as well as the modified contact information is passed to the UpdateContact( ). Once the ArrayList is updated that updated ArrayList should get reflected in the webpage and to do so we call UpdateListTable( ) function such that the table is now properly populated with the current values of the ArrayList. Then we say AddTable.Disabled = false. Then say EditTable.Visible = false i.e. make EditTable invisible. SaveButton.Visible = false i.e. make the Save button invisible along with the EditTable. Then we make DeleteButton.Enabled = false and the label which displays the name, telephone number and mobile number is emptied using DelContact.Text = "".

Step VI – Build & Test

First we make a request for ContactsManage.aspx using CTRL + F5 i.e. we build it. Once that is done, we can type out the name that is to be searched and then click on the Search button.

Edit & Save a contact

When we click on the Search button, the EditTable should now be visible. We had typed out name as Sachin. Hence the EditTable is now populated with Sachin Tendulkar, his telephone number and his mobile number. The Save button is enabled now whereas the AddTable stands disabled. Now we have to decide out of the name, telephone and mobile about Sachin Tendulkar, what changes to make. So we have written Sachin Tendulkar’s middle name i.e. Ramesh as shown in the slide and then click on the Save button.

All Done!
As soon as we click on the Save button, the Save button’s handler is going to get called. Within the Save button handler, we had updated the ArrayList and we had also made changes in the ContactsManage.aspx page such that the modified name should appear in the recent contact list. So as shown in the slide, the changes are reflected in the contact list.
XML Contacts - I

In this lecture you will understand:

- Understanding XML
- Important XML terminologies
- Using XML file to store contacts
- Important XML classes
  - XmlReader
  - XmlWriter
What is XML?
XML is an eXtensible Markup Language. In XML, we can create our own tags whereas whenever we are using HTML, we are restricted to using the standard tags that are made available as per the HTML standard. In XML, the format in which we layout these tags is a text format. It contains different tags the way HTML document has. Only thing is that we can decide the tag names. XML provides a very easy way to exchange data between different modules of the applications or from one application to another. It’s a standard which has been created by World Wide Web Consortium. Hence it’s a universally acceptable standard. Then we make use of readymade classes that are present in the .NET Framework Library to read/write XML files. It’s extremely easy to learn and use. Entire World has embraced XML to a great extent whenever there is need to exchange data.

Our Very Own XML Document
Any XML document would usually begin with a XML declaration containing XML version number and the character encoding that has been used in the XML document that follows i.e. <?xml version = "1.0" encoding = "utf-8" ?>. This XML declaration is optional but if present then the 1st value within that declaration will have to be the version number of the XML. In this case it is 1.0. encoding indicates what kind of format we are using for representing characters. So encoding is an attribute. Attribute is also optional. If we need then we can mention as utf-8 or utf-16 where utf stands for Unicode Transformation Format. Basically it indicates whether each character in the document is going to be represented using 8-bits or using 16-bits. Here we are using utf-8 indicating that every character in the document that follows would be represented using 8-bits. If XML declaration is present in the document then it will have to be the very 1st line in that document. We will begin with <contacts> and end it with </contacts>. Inside this we have another tag called <contact>. This <contacts> and <contact> in XML terminology is known as an element. <contact> ends with </contact>. Inside the <contact> tag we have <name> tag which ends with </name>. Name may be Mansoor Shaikh. Then we have his telephone number followed by mobile number. Since mobile number is optional, so we say <mobile></mobile>. We can have one more <contact> tag within the same XML document which will also have some name, telephone number and mobile number. So Mansoor Shaikh, Anil J Fernandes, etc, are the values. So <contacts> is the root element within which we <contact> and inside that we have three elements <name>, <telephone> and <mobile>.

XML Based Contact Book
When we created the Contacts website in which we had comma separated values of contact information stored in a ContactsList.txt file that time we had created Contact as the base class from which we had derived the FileContact class. Then we had a ContactList class which tried to communicate with this Contact or FileContact class to be able to store contacts, read contacts, add contacts, delete contacts and modify contacts. Then all the information that was persistently stored used to go in ContactsList.txt file. All this was the backend logic. Then we had the front-end logic stored in a ContactsManage class through which we are trying to invoke member functions of ContactList class which inturn communicated with FileContact member functions or Contact member functions. This was the scenario for the text based contact that we had created. Similar we want to do for XML based contact book. In place of FileContact class, now we will have the XmlContact class which is also derived from the Contact class. Then we will have to suitably update the ContactList class. In place of ContactsList.txt, this time we will have ContactsList.xml. All this backend logic, we will once again try to manage through the webpage ContactsManage. We will have to newly create XmlContact class. ContactList and ContactsManage, we will have to appropriately change. ContactsList.xml we will have to create.

Steps Involved
Since many things are common from the Contacts project that we had created, 1st this we need to copy Contacts project into a folder Contacts_xml. Then try to create an empty XML file. Do not populate it by hand. Then we create XML reader/writer class and its name is XmlContact. Then suitably update the ContactList class to be able to load a XML file, save an XML file i.e. we need to add XML load/save functions to ContactList. Then make changes in ContactsManage.aspx.cs. Within the webpage, we will have to make changes in all the handlers that we had written i.e. Add, Save and Delete contact handlers. Then we will have to replace the FileContact class with XmlContact class. Then loading and saving calls for a text based file, those calls also we will have to change for corresponding XML related file.

Step 1 – Adding XML File

Go to Contacts_xml website and locate App_Data directory in the Solution Explorer. The moment we are able to locate that folder, right click on it. When we right click, we will get a menu as shown in the slide. From this menu select Add New Item. As soon as we select Add New Item, a dialog would appear which contains four different options. Select XML file from the dialog that appears. Give the name of the file as ContactsList.xml and click on Add.

ContactsList.xml

When we do this, an empty XML file will now get created which is shown in the slide. In this file, there is a declaration at the top which contains XML version number and the encoding that this document makes use of. By default this line has been inserted by the wizard in ContactsList.xml. We need to add the root element here i.e. <contacts></contacts>. This we have to type by hand. Inside this, what is to be populated, we have to do that in a programmatic fashion.

Step 2 – XmlContact class

We have to create a new class called XmlContact. Add this class to App_Code folder by right clicking App_Code folder and select Add New Item. Then select Class from the dialog that appears and give the name as XmlContact. Then a new class will get inserted. This is going to act similar to FileContact class except that it would not make use of the .txt file. Instead we expect this XmlContact class to make use of ContactsList.xml file. We should be able to read and write data into this XML document and to do so we propose to make use of classes XmlReader and XmlWriter. Within XmlContact class, we will try to make use of XmlReader and XmlWriter classes to carrying out reading or writing of our XML document. We are going to have a Contact class and we derive XmlContact from it. In the XmlContact class, we propose to have 0-argument constructor, WriteToXml( ) and LoadFromXml( ) functions. All the contact information would be written using WriteToXml( ) and would be read back using LoadFromXml( ) function.

XmlContact

At the top we say using System and using System.Xml. XmlReader and XmlWriter classes would become available to us only when we import System.Xml namespace. This XmlContact class is going to be derived from the Contact class. Inside this class we try to declare constant strings i.e. const string contactStr = "contact", const string nameStr = "name", const string telStr = "telephone" and const string mobStr = "mobile". We want to create tags and we are trying to give names of these tags contact, name, telephone and mobile which we are storing in four constant strings. const indicates that these strings are unchangeable. These correspond to the XML element that we intend to create within the XML file. Then we have a 0-argument constructor which right now is kept as empty.

Slide 11
LoadFromXml() function will read a contact information from the XML file. To be able to carry out that reading, XmlReader object will be passed to it. Whichever XmlReader object will be passed to it, using that it will try to read a contact from the XML file. Our contact is organized in an XML file in a manner as shown in the slide. Within <contact>, we are going to have <name>, <telephone> and <mobile> elements. We want to read the contact information of only one contact and then go back from this LoadFromXml(). We will begin with bool ret = false. This indicates the return value of LoadFromXml( ) function. Then using the XmlReader object that this LoadFromXml( ) function receives, we will start reading the XML file. So long as xr.Read() returns true till that time this loop should continue i.e. the loop is while ( xr.Read() ). Inside the loop we would check that the node at which we are present right now is it an element or not i.e. if ( xr.NodeType == XmlNodeType.Element ). NodeType is a property of the XmlReader class. XmlNodeType.Element is an enumeration. If it is an element then we should try to access name of it. Then we check if ( xr.Name.ToLower() == nameStr ). Name is a property of the XmlReader class which returns the string present in that node. If it matches with nameStr then we say m_name = xr.ReadString( ) i.e. read the value present inside that name element and ret = true. m_name is a property which is available to the XmlContact class. m_name property was declared in the Contact class which we have inherited in this class. If it is not name then check whether we have come across a telephone element or not i.e. else if ( xr.Name.ToLower() == telStr ) and if it is then say m_telno = xr.ReadString( ). If it is a mobile number i.e. else if ( xr.Name.ToLower() == mobStr ) then say m_mobile = xr.ReadString( ). Then we check else if ( xr.NodeType == XmlNodeType.EndElement && xr.Name.ToLower() == contactStr ) i.e. we are finding out that the node at which we are present is it an element or not. If it is an element and if it happens to be the contactStr means have we reach the end or not. If we have reach the end of that contact then break. Finally return the value present in ret using return ret. So we are going to read one contact information any time LoadFromXml( ) is called.

Slide 12

To create an XML document, we will have a WriteToXml( ) method and it expects XmlWriter object to be passed to it. Firstly we say, xw.WriteStartElement( contactStr ) which will create the element <contact> within the XML file, the file which is associated with the XmlWriter object and whose reference has been passed to WriteToXml( ). Then we would write the <name> element by saying xw.WriteStartElement( nameStr ). Inside that <name> we propose to have Mansoor Shaikh which can be written using xw.WriteValue ( this.name ). Then we would write the <name> element by using xw.WriteEndElement( ). To write <telephone> tag we say xw.WriteStartElement ( telStr ), write value of telephone using xw.WriteValue ( this.telno ) and write </telephone> tag using xw.WriteEndElement( ). Finally we want to write the <mobile> tag which is written using xw.WriteStartElement ( mobStr ), value of mobile is written using xw.WriteValue ( this.mobile ) and the </mobile> tag is written using xw.WriteEndElement( ). Lastly to write the </contact> tag, we once again say xw.WriteEndElement( ).
In this lecture you will understand:

- Updating ContactList class
- Updating ContactsManage class
- Build and Test
**XML Based Contact Book**

When we wish to create XML based contact book, we had XmlContact class which is derived from the Contact class. ContactList and ContactsManage are we are going to change in this lecture. Contact and XmlContact are done.

**Step 3 – ContactList Class**

The figure shows that the ContactList is going to manage ArrayList and when we created the text based contact book that time in the ArrayList references of FileContact were present. We had functions like 0-argument constructor, 1-argument constructor, LoadFromFile( ), SaveToFile( ), Search( ), AddContact( ), DeleteContact( ), UpdateContact( ) within ContactList class. Parallel to these, we are going to have an ArrayList here which is going to hold references of XmlContact objects rather than FileContact objects. We have functions like 0-argument constructor, 1-argument constructor, LoadFromXmlFile( ), SaveToXmlFile( ), Search( ), AddContact( ), DeleteContact( ) and UpdateContact( ) within the ContactList class. 0-argument constructor remains same and Search( ) function also remains same.

**ContactList – 1-arg. Ctor**

In the 1-argument constructor, we receive the path of the XML file. Whatever is the path that is stored in a private variable m_path. When this ContactList constructor is invoked, we will also try to create a new ArrayList. Then we call LoadFromXmlFile( ) function.

**ContactList – LoadFromXmlFile( )**

Inside the LoadFromXmlFile( ) function, we have a XmlReader reference xr which is set to a value null. Within a try block, we will try to create a XmlReader object by saying xr = XmlReader.Create ( m_path ). Then in an infinite while loop, we should create a new XmlContact object. Once the XmlContact object is created, store its reference in XmlContact con. Using con, call the LoadFromXml( ) function which is present inside the XmlContact class. We check if ( ! con.LoadFromXml( xr ) ) and if it is then break. The contact information that is read using con.LoadFromXml( ), we will try to add that to our ArrayList by saying m_list.Add( con ). In ArrayList, we are going to maintain several contact objects. For the try block, we may or may not have a catch block. Now we will have a finally block. Finally goes to work whether an exception occurs or does not occur within the try block. In the finally block, we check if ( xr != null ) and if it is then close the reader by saying xr.Close( ). m_list.Add( ) is going to maintain the ArrayList of XmlContact objects.

**ContactList – Add, Delete, Update**

The AddContact( ) function receives the XmlContact con object. It adds the XmlContact object to the ArrayList by saying m_list.Add( con ). Then we will have to save the updated ArrayList to the XML file, for that we will call SaveToXmlFile( ) function. Then we have DeleteContact( ) function which receive the contact to be deleted. First it has to search for this particular contact. For that we will use ContactList’s function Search( ). Contact con = ( Contact ) Search ( c.name ). We have to remove this contact that has been searched from the ContactList using m_list.Remove( con ). Once the contact has been deleted from the ArrayList, we now want to save this updated ArrayList into the XML file by calling SaveToXmlFile( ) function. We also have the UpdateContact( ) function which receives 4 strings; string original_n, string n, string t, string m. 1st string is the original name, then name, telephone number and mobile number respectively. Within this function we 1st search for the original name using Contact con = ( Contact ) Search ( original_n ). Once this done then we store in this con object the values of name, telephone and mobile numbers that we received. Once that is done we
have to save this updated contact information present in the ArrayList in the XML file by calling 
SaveToXmlFile( ) function.

**ContactList – SaveToXmlFile( )**

Into SaveToXmlFile( ) function, we have to write the contact information that includes the contact 
within which we have name, telephone number and mobile number. First we would create an object 
of XmlWriterSettings and once this object is created its reference is collected in variable settings. 
Then we create a reference xw of XmlWriter which is set to null. Inside the try block, we would set 
the Indent property of the XmlWriterSettings object to true. Unless we set this Indent property to 
true, we find that if we write the XML tags into the XML file, all of them would come in same line. 
But we do not want that. We want that the contact information should get stored in the file in a 
readable fashion to ensure that we have to mark Indent property to true. Then we will create a 
XmlWriter object using xw = XmlWriter.Create ( m_path, settings ). Then we say 
xw.WriteStartElement ( "contacts" ). So <contacts> get written into the XML file. Then we will 
iterate through the ArrayList by finding the number of elements in it, those many times. Number of 
contact information in the ArrayList can be found using m_list.Count property. Inside the for loop, 
we would first extract the information about the XML contact from m_list i.e. we say XmlContact 
con = ( XmlContact ) m_list [ i ]. Then we call con.WriteToXml ( xw ) which will write the contact 
information into the XML file. Then we will have to put </contact>. So we say 
xw.WriteEndElement( ). Then we can flush out the writer by saying xw.Flush( ). In the finally block, 
we check if ( xw != null ) and if it is then we say xw.Close( ).

**Step 4 – ContactsManage Class**

In the XML based ContactsManage class, we are going to more or less similar functions as compared 
to the text based ContactsManage class. AddContactRow( ) and AddHeader( ) these functions are 
same. So also the ShowEditTable( ) method. Then other methods like Page_Load( ), 
AddButton_Click( ), WriteToXmlFile( ), UpdateListTable( ), LoadFromXmlFile( ), 
SearchButton_Click( ), DeleteButton_Click( ) and SaveButton_Click( ), we will have to rewrite these 
functions.

**ContactsManage – Page_Load( )**

In this method, we are going to add three columns by calling AddHeader( ) function. So we say 
ListTable.Rows.Add ( AddHeader( ) ) which adds a header but addition actually happens through the 
AddHeader( ) method. Then we call a function LoadFromXmlFile( ). In the ContactsManage.aspx.cs 
file, we have to import System.Xml. The LoadFromXmlFile( ) function creates a new ContactList 
object. We pass the path of the XML file to that ContactList object. Then it will iterate through the 
list and each time through that it will call AddContactRow( ) method i.e. AddContactRow ( ( Contact 
) clist.List[ i ] ). So as many items are there those many times a new row will get added through the 
AddContactRow( ).

**ContactsManage – Add Contact**

To carry out addition of new contact, we have to have the AddButton_Click( ) handler. Within this 
handler, we have to first validate the name and telephone numbers length. So we say if ( 
nameInput.Text.Length > 0 && telInput.Text.Length > 0 ) and if it is then call WriteToXmlFile( ) 
function. Subsequent to that call UpdateListTable( ) function using that whatever are the current 
contents of the XML file, it would appropriately update the contact information shown in the 
webpage as per the current contents of the XML file.

**Add Contact Continued...**
In the `WriteToXmlFile()` method, we first create a new `XmlContact`. Then create a new `ContactList` object and pass path of the XML file to that. Whenever we give / Windows convert it to \. If we write the path using \\ then we will have to provide \ otherwise it will create a problem. Once that is done, we will try to pick the values from the three fields i.e. textboxes and if within those textboxes, if there are commas then replace them with spaces. Now the contact is ready. We should add this contact first to the `ArrayList` by calling `AddContact()` method which updates the `ArrayList` and the updated `ArrayList` is also going to save into the XML file. The `UpdateListTable()` first clear out all the information that is there in the `ListTable`. Once the existing contact information is cleared from the `ListTable`, now we are going to populate it. For that first we need to add the header and after that we have to load the XML file by calling `LoadFromXmlFile()` method. `LoadFromXmlFile()` method read the XML file and populate this `ListTable`.

**ContactsManage – Search Handler**

Inside the `SearchButton_Click()` handler, we have to first validate the length of the name. If the length is more than 0 then first we create the `ContactList` object by passing the path of the `ContactsList.xml` file. Then call the `Search()` method and pass the name that has been typed to it. While doing so, we have to make use of the `AddTable`. Once the object is located, it is collected in `con`. Then check whether con is null or not using if (con != null) and if this condition is true then we create `DelContact.Text` and store name, telephone number and mobile number in it. Then we say `Session["contact"] = con` which put the con into the session. Once it goes into the session then we have to manage the front-end. So we say `AddTable.Disabled = true` and we again say `DeleteButton.Enabled = true`. Then we display the `EditTable` by saying `ShowEditTable (con)`.

**ContactsManage – Delete Handler**

In this handler, we will begin with first extracting the value that is stored in Session against the contact index which is collected in `oldCon`. Then create a new `ContactList` object and then call `clist.DeleteContact(oldCon)` method and then update the `ListTable` by calling `UpdateListTable()`.

**ContactsManage – Save Handler**

In this handler, we once again obtain the contact information stored in the Session, get it in `oldCon`. Then create a `ContactList` object by calling its constructor. Then call the `Search()` function. Once this is done, now `EditTable` would come into picture. Pick the values present in the `EditTable`, store them in the `Contact` object and then call `SaveToXmlFile()` because the changed information of the contact we will have to be persisted with. For that `SaveToXmlFile()` will have to be called. That changed information will also get reflected into our webpage and to guarantee that make a call to `UpdateListTable()`. Then set `AddTable.Disabled = false`, `EditTable.Visible = false`, `SaveButton.Visible = false`, `DeleteButton.Enabled = false`, and `DelContact.Text = ""`.

**All Done!**

When we test this, we will get a table shown in the slide. One table shows the existing contacts, another the `AddTable` and the `EditTable` but the `EditTable` is not visible because it will become visible only when search is over.
Contacts Revisited

In this lecture you will understand:

- Working of ArrayList
- How is it different
- Recap of Contacts and Contacts_Xml
- Steps involved in List, Add, Search, Delete, Modify
Operations

The operations that we carried out are Listing, Addition, Searching, Deletion and Modification. All of these make use of the ArrayList. ArrayList is used to maintain the contact information in memory but whatever is present in memory is not enough because ultimately any information unless it is stored in a consistent and proper state on a disk, we would never be able to retrieve it back. Means we got to persist the information present in the ArrayList onto the disk. In the websites that we created we did manage to write that information into the txt file or the xml file. When we created Contacts website, that time ArrayList was same but this time instead of writing the information about the contacts in a text file in a comma separated pattern, we decided to make use of the data standard i.e. Xml standard to write information into an XML file. We did manage to read this information back from the XML file and we also populate the ArrayList with this information and carried out variety of operations on that particular ArrayList.

ArrayList

ArrayList is a collection of either a FileContact objects or XmlContact objects. The ArrayList is wrongly construed as a collection of objects whereas if we look carefully the ArrayList really is not a collection of objects. It is collection of references of these different objects. So in ArrayList, in each individual slot, reference of an object is stored. In Contacts website, each reference was a reference to a FileContact object whereas in Contacts_Xml website each reference was a reference to a XmlContact object. Each element is of the type reference to an object. We created the Contact class in both websites. From that we derived FileContact object or an XmlContact object. Each element is of the type reference to an object. We created the Contact class in both websites. From that we derived FileContact object or an XmlContact object. But the Contact class itself is derived from another class and name of that class i.e. the base class is Object rather than a class. All datatypes in .NET are derived from the Object class. So from Object, Contact is derived and from Contact either FileContact or XmlContact is derived. So the ArrayList collection class is able to hold sometimes the FileContact objects and sometimes the XmlContact objects because in a base class reference we are always allowed to store address of a derived class objects. This process is known as upcasting of that reference. So ArrayList is a collection of upcasted reference. We can store references to any objects within the ArrayList. That is the reason why whenever we tried to extract from the ArrayList, we are always required to cast what we have extracted suitably. ArrayList is somewhat different than the array means we can access its element using an index. ArrayList has a special property whereby it can increase in size or decrease in size as per the requirement which is not possible with a conventional array. In conventional array, we have to always mention the size of the array upfront and once this size is mentioned, there onwards during course of execution of program there is no question of increasing the size of the array or decreasing the size of the array. So expansion and shrinking is not possible in normal array, its possible in case of ArrayList which makes it special.

Operations On ArrayList

The two operations that we performed in building of these two websites, Contacts and Contacts_Xml website were addition of a new contact into the ArrayList or removal of an existing contact from the ArrayList. Add( ) and Remove( ) functions receive an argument of the type Object o. Means we might pass it a FileContact object’s reference or a XmlContact object’s reference but Add( ) and Remove( ) are going to access that reference into a pointer to the base class or a reference to a base class. Suppose we have an ArrayList where we have already addresses or references of different XmlContact or FileContact objects. If we decide to add one more XmlContact or FileContact object, firstly that XmlContact or FileContact object needs to be created and reference of this particular object will have to be passed to the Add( ) function. ArrayList class has one of the member variable which keeps track of how many elements have been filled in this ArrayList. No sooner we had passed
to the Add( ) function the reference to a FileContact or XmlContact object, it will increment that count and then in the next empty slot present in the ArrayList, it would try to store the reference to this new XmlContact or FileContact object. Every single time we add a new element to this ArrayList, that time the ArrayList does not grow because that would be quite inefficient. For example the ArrayList grows by multiple of 5 slots. Means this is commonly known as a grow size. Means suppose that any time the grow size is 5 then the moment we create the 1st contact object and try to store it in the ArrayList, the ArrayList will create 5 slots. Out of that we will consume only the 1st. Till the time we make next 4 requests, there is no need to increase the size of the ArrayList. But the moment all the 5 slots have been consumed, if we try to add one more i.e. the 6th XmlContact or FileContact object to the ArrayList, at that time the ArrayList size would expand by 5 elements or it will expand by whatever is the value of grow size. This is all about the addition of new XmlContact or FileContact object to the ArrayList is concerned. Now we want to perform elimination of an existing contact from the ArrayList and for that the Remove( ) function is used. We may ask it to remove a XmlContact or FileContact object but Remove( ) function is always going to receive these reference to XmlContact or FileContact objects into a Object reference i.e. o which is once again going to be an upcasted reference. Suppose we decide to carry out deletion of the 3rd contact present in the ArrayList. The moment this 3rd contact is to be removed by calling the Remove( ) function, the Remove( ) function makes sure that 3rd reference in the ArrayList becomes null. The moment it becomes empty, the XmlContact object or FileContact object will not be pointed to by any reference. So next time the Garbage Collector present in the ASP.NET runtime will immediately eliminate this XmlContact or FileContact object because nobody is referring to that object. So that object is eliminated from memory. It is known as Garbage collection. If the 3rd contact gets deleted then that slot will fall vacant in the ArrayList. Hence the ArrayList implementation is done in such a manner that anytime a slot falls free, elements to the right of that slot automatically pushed to the left. Means left shifting takes place anytime the element goes away from the list.

**Purpose Of Each Class**

Contact is the base class from which we have derived the FileContact and XmlContact class. Then we had a ContactList class which used to interact with the FileContact or XmlContact class. ContactList also used to read or write information from either ContactsList.txt or ContactsList.xml. All of this was the backend logic. To interact with this backend logic, we had built the front-end logic in the form of different helper functions and different button handlers and we had put all of them into ContactsManage class. ContactsManage class is derived from the Page class where Page class is the .NET Framework class. Contact class represents information about any individual contact in memory whereas FileContact or XmlContact were given the responsibility of either reading one contact information or writing one contact information. From where we read and where we write that depends upon whether it’s a Contacts website or a Contacts_Xml website. For the Contacts website, reading or writing happen from a txt file and the values that were written into this file were always comma separated values. In Contacts_Xml website, the functions that we use from the XmlContact class were capable of writing one contact in the form of XML into the ContactsList.xml file or we had another function within the XmlContact class which would read information about one contact from the XML file. FileContact and XmlContact reading / writing functions were able to deal with only one contact at a time. For that we had to construct the ContactList class which could deal with multiple contacts rather than one contact. If we want to write multiple contacts in a XML file then the writing function inturn will rely on the XmlContact’s writing function because that knows how to write one contact. In a loop, we have to keep calling that repeatedly within the ContactList class. If we do so then same writing function that we was used for writing one contact can repeatedly get called and go on writing newer and newer contact information into the XML file. Whatever applies to writing equally applies to reading of the information from the ContactsList.xml file. ContactsManage class had different button handlers. For example, we had button handler for Add, Save, Delete, Search. All those handlers along with certain helper functions are available.
Addition

When we carried out addition of a new contact, we had three text boxes in which we could type the information and once that information about name, telephone number and mobile number was typed, we had to click on the Add button. When we clicked on the Add button, the ContactsManage class’s AddButton_Click( ) handler get called. This function in turn call two more functions; one was WriteToTextFile( ) and another was UpdateListTable( ). WriteToTextFile( ) became WriteToXmlFile( ) in the XML based contacts book. WriteToTextFile( ) in turn created a new FileContact object. In case of WriteToXmlFile( ), it must have created a new XmlContact object. Once the FileContact or XmlContact object is created, we also manage to create a new ContactList object. To create a new ContactList object, we must have invoked the constructor. The constructor manages to load ContactsList.txt or ContactsList.xml whatever the case may be. Having done that the WriteToTextFile( ) having read the ContactsList.txt or ContactsList.xml, it tried to replace commas with spaces in the new contact that we must have typed within the three textboxes in the webpage. Once that is done the ContactList’s AddContact( ) function got called. AddContact( ) added a new contact to the ArrayList which get appended to the existing contact in the ArrayList. After updating the ArrayList then we manage to call the ContactList class’s SaveToFile( ) or SaveToXmlFile( ) function to save the data that is in the ArrayList now in either the txt file or in the XML file. The information that is been written to the XML file, it should also try to update that information on the webpage. To carry out that updation, we had called UpdateListTable( ). UpdateListTable( ) had prepared the table and then it manage to call LoadFromTextFile( ) or LoadFromXmlFile( ) to load the contacts from the disk into the ArrayList and then manage to populate all of that within this ListTable.

Search

We have typed out the name of the person to be searched in the 1st textbox out of the three textboxes. No sooner we click on the Search button, the SearchButton_Click( ) of the ContactsManage class i.e. button handler got called. When this button handler got called, it created a new ContactList object. As soon as the new ContactList object is created, the 1-argument constructor of the ContactList class is used to invoke and then we have to pass the name if the txt file or name of the XML file to this constructor. It used to load the contents of the txt or XML file into the memory and then populate the ArrayList. Once the ArrayList is populated with existing contact information, we call the ContactList class’s Search( ) function. This Search( ) function might either be successful or unsuccessful. Assuming that it is successful, the contact information that is searched used to get displayed in a label. Then we manage to store the search contact reference in Session object. Once that is done, we manage to call the ContactsManage’s ShowEditTable( ) function such that the information that was shown in the label also got displayed in this EditTable. ShowEditTable( ) displayed the searched contact information in the EditTable. Means what is searched is displayed in the label, it also got displayed in the EditTable. So EditTable is populated only with a motive to edit the information.

Delete

As shown in the slide, we have searched for a name Rajesh and in the label, we have displayed Rajesh Diwan, his telephone number and mobile number. Subsequent to this, once the information is shown in the label and the user actually decides to delete this particular contact, he would click on the Delete button. As soon as he clicks on the Delete button, the Delete button’s handler i.e. DeleteButton_Click( ) got called. It tried to create a new ContactList object. The constructor manages to load the ContactsList.txt or ContactsList.xml file. Whatever information we had stored in the Session about the contact that is to be deleted, that object would be retrieved from the Session object. Once that information is retrieved from the Session, we manage to call ContactList class’s DeleteContact( ) function. DeleteContact( ) function search for the contact that was retrieved from the Session in the ArrayList and we called it as old contact. The searched contact is eliminated from the ArrayList. The updated ArrayList then got finally saved into the disk by calling either SaveToFile( )
or SaveToXmlFile( ) function. Once this is done, it should also get deleted in the webpage. Hence we called UpdateListTable( ) which updated the ListTable by first putting the columns in place and then calling the LoadFromTextFile( ) or LoadFromXmlFile( ) to read the current contact information and display it in the ListTable.

**Modify**

If we have searched for a particular contact, it will get displayed in the label. It will also get displayed in the EditTable. Once the EditTable is shown, the user may click on the Save button. No sooner does he do that the SaveButton_Click( ) handler in the ContactsManage class would get called. This handler created a new ContactList object. Then it retrieved the old contact information from the Session and then managed to call the UpdateContact( ) function. Loading or populating the ArrayList was done by the constructor which manages to load the txt file or XML file. Now the ArrayList stands populated. Information about the existing or old contact that has been retrieved from the Session. Then UpdateContact( ) manages to search and update and then save the updated list into the disk by calling SaveToFile( ) or SaveToXmlFile( ) function. Lastly UpdateListTable( ) prepares the column headers and loaded the information from the text file or the XML file and displayed it in the ListTable.

**How Sessions Work - I**

When we carry out the search request at that time, in the web server’s memory there is already a Session object present. This Session object is being referred to by ASP.NET runtime. That’s why even if we do not connect to the web server; the Session object still continues to survive. The moment we make a request for a particular webpage for example, when we clicked on the Search button, the Search( ) function or handler got called and then the webpage was loaded. At that time a Page object got created. Suppose this is Page object for client1. The same website can be accessed by multiple people. Hence there might be other Page objects as well. All these page objects have a reference to the same Session object. Means common Session object is there for all different page objects. Means within the Page object there is a reference to this Session object. ContactList object is created whenever we try to carry out the search firstly the ContactList object is created. Creation of newer objects is happening in web server’s memory. Once the ContactList object got created, the ArrayList object got created. Then we populate the ArrayList object by calling functions like LoadFromTextFile( ) or LoadFromXmlFile( ). It got populated with the different FileContact or XmlContact objects. Then we try to carry out the search. When search is done, it will try to go through these different objects that are there one after the other. Once the search is done successfully, the searched contact object is saved in a Session. Means to whichever object we wish to saved in a Session, a reference is pointed from the Session object. Once that is over the Page object, ContactList object and the ArrayList object, all of them would die because control would go back from the search. Once all these jobs are over and control returns the Page object, ContactList object and the ArrayList object are bound to die. However, even though they die, the Session-based contact object is continue to survive. Since it is surviving, now if we try to carry out modification or deletion, we are able to retrieve this from the session.

**How Sessions Work - II**

When we carry out the deletion or modification, at that time in the web server’s memory in the Session object, we have reference to the contact object that was searched during the Search( ) function. Now the Page object would get created for this client. All of them are going to share the same Session object. So the reference would also being pointed to this Session object. Then the ContactList object is created. Subsequent to that the ArrayList object is created and subsequent to that different FileContact or XmlContact objects get created. So the ArrayList gets suitably populated. If we are trying to modify then we must be able to go to the particular contact which we wish to modify or if we want to delete a particular contact, we should be able to locate that contact
from the ArrayList. To do that a search is carried out. But when we carry out the search, we are searching for what that has to be clear and that can be retrieved from the Session because Session object is indeed pointing to the contact object which we had searched successfully. This contact object is compared for with every single object that is there in the ArrayList. Once this retrieval has been done, the searched contact is deleted or modified. Then the updated ArrayList is saved to file. Subsequent to that the page object would die, the ArrayList and ContactList object would also die. The Session object is left in memory being referred to by page objects of other clients.

**Nature of Get & Post**

We can treat Get as a humble request rather than authoritative command which is a Post request. Post is known as a request but it’s an authoritative command. Whenever a Get request is made no matter for which client we are making this request, the response for each one of them would be same. For example, no matter which client makes a request from which browser, he is always going to get the same list of contacts. However, in case of Post, what would be the response that depends upon what did you Post to the client, what data did you send to the client. For example, one client may search for Sachin’s name in the contacts whereas another client may search for Rahul’s name. So what would be returned as a response to these two different clients is bound to be different. Unlike that for Get, response is same for everybody. When we say that response is same for everybody, we are believing that Get is not being used to send the data. Commonly data is send using Post. So assuming that we are not going to send any data using Get then we can say that yes, we are going to get same response for every client when we do Get whereas we are going to get different responses in case of Post depending upon what data did we send to the server.
Three Tier Architecture

In this lecture you will understand:

▶ Web Application Architecture
▶ Our development environment
▶ Importance of databases
▶ Working with databases
Tiers Of Web Applications

We have a Client machine, then Web Server machine and then a Database Server machine. The client machine is supposed to interact only with the web server whereas the web server, whatever request it gets, it has to pass them on for either storing data or retrieving data from a database server. In the previous two websites i.e. Contacts book website which is text based and XML based, in both these two websites, we did not have the database server. We had only the client and the web server. Whenever the web server is used to get request to lets say add a new contact, that new contact we did not store in a database. We try to store them in a Comma Separated Value format text file or further to that we try to store the same contact information in an XML file. We had no database server component in those two websites. Its time we graduate into a database server because that is found in real life. So these different components of a web application would reside in different machines. So any web application is a distributed application. The moment an application becomes a distributed application, the way the application is build changes, the concern changes, the issues change. In a typical desktop application all the three part, the front end, the middle tier and the backend, all of them are usually on the same machine. Whereas in a distributed application, since different components of the application reside on physically different machines, hence architecturing such an application becomes more difficult. We can never bypass any of these tiers or any of these layers. The client can never communicate to a database server directly. Client will always have to first make a request to the web server. It is the web server which will communicate with the database server. Likewise database server can never send information directly to the client. It will be able to communicate with the web server; web server will extract the information from the data server and then pass that information to the client’s browser where it will get suitably rendered. In distributed application, number of users of that application are quite large. Means several requests are coming to the web server and the same web server, once it receives the request tries to pass on some of the request to database server to retrieve data or to store data into the web server. There can be multiple web servers which communicate with one database server. There might be multiple database servers as well. So in that sense it becomes many to many kind of relationship. Whenever we have multiple web servers that is known as web farm. Each layer would be able to communicate to only its immediately next layer. Client cannot directly communicate with database server and database server also cannot directly connect to the client. So communication between client and database server is really not feasible. All the requests always have to be routed through the web server only. Typically the client is Internet Explorer or Firefox whereas web server is IIS 6.0 or 7.0. If it is Windows XP then it is IIS 6.0. If it is Windows server 2003 Operating System then IIS 7.0 is used. Client is an entity through which we can access the functionality of web application. The database server can be SQL Server 2000 / 2005.

3 Tier Architecture Details

First is the presentation layer. This presentation tier is either an Internet Explorer or Firefox i.e. it is a web browser. Through this web browser, we just have an access to the functionality of the web application. The middle tier is commonly known as a business logic tier wherein on the web server there is the web application present. Web server could be Microsoft IIS server or an Apache server or a Tomcat server. Tomcat is the open source implementation of a web server. Most of the time, in building a web application is spent in building the business logic layer. That is where the focus of all the application developers is. For example, in the Java world, we not only have the web server, we have an application server. Application servers like WebSphere or WebLogic or even Tomcat, these are servers that are used in Java world. Those servers are not merely Web Servers where we make a request and get a response. Those application servers have web server as one of their components. There are many other components also present in application servers. For example, there is a component which allows to balance the load that comes on the web server. So to manage the business logic, lot of innovative ideas have come up. We are going to use here Microsoft IIS server. The third
layer is the persistence layer or a backend layer or a data layer. This data layer makes use of a relational database i.e. RDBMS. There are different implementations of this relational database. One of which is carried out by Oracle. Another has been carried out by Microsoft’s SQL Server. Then there is open source implementation in the form of MySQL. Out of all these, Oracle implementation is possibly the best. So, different implementations of this RDBMS is done by different companies. Even though the implementations are different, the mechanism to use these different implementations remains same. We as programmers do not have to bother about how Oracle actually manages to store the data internally or how MS SQL manages to store the data internally. We have to look at the access mechanisms and these fortunately are standard. No matter what exactly is the implementation of the database. Communication that is carried out between different tiers, that we can never bypass a tier. Separation of layers ensures a good design for the application. If we separate our logic into these different layers, it’s much easier to manage the overall application.

Our Development Setup

We have Internet Explorer in the presentation tier. We have IIS server software in the middle tier or business logic tier. We propose to have SQL server software in the data tier. All these tiers for development setup are going to be present in exactly the same machine. It makes sense in doing so because if we actually distribute this across three different machines then development becomes harder. If all of them are present on the same machine in that case development becomes much faster because we do not have to deal with network traffic, network congestion. When we say IIS server and SQL server are present on the same machine this server means the server software and not the physical machine. So IIS server software, SQL server software and Internet Explorer for our development are present on the same machine which reduces our development time. Most of the times development is done in this manner. Later on the business logic layer is transferred to IIS server which might be physically a different machine and the data is then transferred to SQL server software which might be installed on a different machine. So deployment scenario and development scenario is going to be different but while doing the development all the three tiers are going to be present on the same machine. Separation of layers makes it simple to deploy the application because the mechanisms that we use to carrying this out are independent of where the server software is actually present. So mechanisms of access from presentation layer to business logic layer or from business logic layer to the data tier are going to remain same, no matter where these three layers are physically present; are they on the same machine or are they on the different machine that do not change the mechanisms. So once the development is over deployment becomes much simpler. Between the Internet Explorer and the SQL server software, there cannot be a communication even when all the three softwares are setting on the same machine.

Database Management System

Database Management System commonly known as DBMS. It is known as relational because it contains information about related data. For example, in our database there might be a list of customers. Then there might be a list of products which the customers may be interested in purchasing. Which customer places an order for which product, we will have to again store somewhere else. Means somehow the customer is related with product through the order that we have placed for the product. This relationship can vary from simple relationships to complex relationships. RDBMS is all about creating the meaningful relationships between the data of the application. This RDBMS is very convenient and efficient to use. In the Contacts website, every time we used to load the contact information that is in the file into the ArrayList. Then we used to modify or delete one of the elements of that ArrayList. We used to rewrite the whole information back into the txt or XML file. So frequent reads and frequent writes is going to hamper the performance of the application. This efficiency concern is no longer our concern. The moment we start making use of the DBMS technology, we have to just see how to access the data, maintaining it efficiently, storing it properly, letting you retrieve it in a meaningful way is taken care by database server software. So there is a
separation of concerns is happening. The moment we have separated the concerns and assigned the efficiency to each of these software pieces, the overall web application performance is bound to be good. Typically in database applications we can build variety of things. For example, a bank carries out the financial transactions using database applications. In airlines website, we carry out online reservations or we have access to schedules of airlines. We can also keep track of which flight has originated from which place at what time. All this information is available on websites. eCommerce websites where online stores are there where we can go and purchase different products. In manufacturing databases have large role to play. In typically an automobile company, there would be 100s and 1000s of components that go into manufacturing of an automobile. There might be spares of different machine that are involved in the manufacturing process. So inventory of products that are being produced, the stores information has to what different parts are there in the stores, issuing of that to the Production or manufacturing department, taking it back, issuing tools, issuing different materials which are necessary for helping in the production of the end product, all that information ultimately has to be stored in a meaningful manner and for that database management system is used. In HR departments of companies where people keep track of the employee records or their salaries or their skill sets, when did they join the company, what is their experience level, how much they have gained in this company, all this information can easily be managed by using a database management system. These are some examples of the DBMS. There are other 100s of application also present in real life.

Does It Solve My Problems?

Our problem is to maintain a contact book. Can we do it using a DBMS? Of course yes because not only from as small as a contact book to building much bigger applications, all these can really be served by the DBMS software. We can build telephone contacts. To do so, we will use MS SQL Server database. Advantages of doing so are we used to frequently load the XML file, update it and again store it. Same is applied for the text file. That sacrifices the efficiency. This is not going to happen with the databases. Once we store the data into databases, it is there. We just have to retrieve it, update it. Store only the piece of information rather than entire set of contacts. Almost every single website worth is solved which has dynamic WebPages would certainly be making use of a database management system to handle their data. A professional level RDBMS like MS SQL can handle large amounts of data. When this data handling is happening, it might so happen that some occurrence of an unexpected event takes place. For example, we are trying to do ordering of a product through Internet based banking account. In the middle of the payment, suppose it so happens that we try to make a payment, our account gets debited and the person who is going to sell us something his account is to get credited. In between these two operations; debiting our account and crediting the sellers account, suppose a crash happens. Then our account stands debited whereas his account will not be suitably credited. Means the transaction will not be complete. Database management technology has features wherein it can put both these operations together in a transaction wherein if both the operations do not happen in that case, it will simply role back to the earlier state. Means if our account got debited and sellers account did not get credited and something went wrong, the debit that happened to our account will be rolled back to its earlier state. Means we can do transactions aware processing of our data. This is a big advantage with databases. RDBMS is very easy to use; moreover it has become a worldwide standard for handling the data. It is very robust and a proven technology.

Logical View Of A Database

How the data is physical organized by the database server is none of our concern. That efficiency is we are not going to worried about. It is guaranteed to us by the RDBMS software. The logical view of the database is important. Every database is going to have a name and one or more tables. For example, we may have a Contacts Database. So Contacts is the name of the database. Inside the Contacts database, there can be multiple tables. Tables also have a name and in that table we have
columns and rows. For example, we may have a Contacts table containing three columns, Name, Telephone and Mobile. Then there would be rows which will have different data. That is how a database is logically organized in the form of one database having multiple tables and each table having rows and columns. Columns indicate the items of information whereas rows containing the actual data against those items.

**Communication With DB**

There are three things that we need to consider when we want to communicate with the database. First is how do we connect to the database. There are two ways to do so. If our database server software is at a remote place then we carry out the connection using a TCP/IP connection. That is a flexible way of doing it because once we are doing it over the network from which physical machine we connect it does not matter. So in that sense, it is more flexible. That’s why through the web browser that is present on different machine, we connect to web server and that web server present on 2nd machine connect to the database server that is present on the third machine. During development when the web server and the database server are present on the same machine, the communication happens using a technology known as Named Pipes. When named pipes are used, replication of data gets avoided as it gets copied from source to target. Once we try to establish the connection, whosoever is going to get connected he has to be authenticated. Whether such a person is an authentic user of that database or not has to be verified. For doing credential check, are we allowed or are we not allowed that can be figured out using the username and password that we use for logging into Windows. When we do that, it is known as Windows Built-in way of authenticating. As against this we may also try to connect to the database server by passing the username and password to the SQL server and SQL server internally maintains a table of valid users and their passwords. So every time a user connects along with his username and password, it would be figured out whether is this an authenticated user or not. Once that is done, we have to now start communicating with the database and to do so, different commands are to be used. Earlier different commands are used for different database servers to be able to retrieve the data or store the data in a database. Now all of that has been standardized in the form of a Structured Query Language or SQL. There are some non-standard extensions to SQL also available.

**Why SQL (SeQueL)?**

SQL is more correctly pronounced as SeQueL. This is used to interact with the business layer. When this communication is carried out, it must be simple to do so. That is the reason why SQL has been created almost in English like fashion. It is very easy to make use of a SQL. This is the popular way of communicating with the database. Database server accepts the commands that are there in SQL. Who created this SQL? Lot of forces in the industry, lot of companies together have created SQL. It was created in the year 1986 and it was coined as SQL1. It became an ANSI standard in 1986. In 1992, more features were added to it. For example, features like counting how many records of a particular type exists in a database or comparison operators using which we can selectively extract data from the database or things like proving a average of a particular column in a database or doing a sum of some of the columns or trying to extract information based on range of dates using between clause. All of those features were added later on and SQL2 came into existence as a result of which in the year 1992 and this is when SQL actually got so widen audience. It almost became necessary from there onwards for people to think of SQL to store their data, retrieve their data, so on. This effort in 1992 was coordinated by Federation of Information Processing Standard or FIPS. Later on in the year 1999, SQL3 came into existence. Some extensions were added to SQL2 and that’s when it became an ISO standard rather than ANSI standard and a FIPS standard. But these extensions have not been accepted very whole heartedly. Different implementations provide different features of this SQL3. Not every implementation would provide every feature. So, most of the times SQL2 commands are to be used rather than SQL3 to communicate with a database.
Important SQL statements

There are three basic types of commands; one belongs to the Data Definition Language. Another belongs to the Data Manipulation Language or DML and 3rd set of commands is belonging to Data Control Language or DCL. DDL allows to create different tables, make changes them using ALTER command and even delete the table using a DROP command. DML is very useful to us. It provides commands like SELECT, INSERT, UPDATE and DELETE. DCL provides commands like GRANT, DENY and REVOKE.
Contact Book Using Database

In this lecture you will understand:

- Creating database visually
- Important SQL Server datatypes
- Understanding SQL queries
- Important SQL classes in .NET framework
Class Organization

When we propose to create the SQL database based contact book, we must be aware of what different classes are we proposing to deal with. The slide shows two figures with the help of which we can compare SQL based contact book with the XML based or text based contact book. When we created those contact books, we used to have a contact class which merely had protected strings standing for name, telephone number and mobile number and then we had properties through which we can set values into these or retrieve the current values through those properties. So we have get and set clauses within those properties. From the Contact class, we had derived a FileContact class or an XmlContact class. FileContact and XmlContact would be able to read one contact information from the contact book or write one contact information into the contact book. Then we had a ContactList class which could do this for multiple contacts and in the process of doing so, it did make use of the FileContact or XmlContact class to read or write one record into the XML file or text file. ContactList used to maintain a list of contacts. It also used to manage a list of contacts in memory using a variable of the type ArrayList. We had a ContactsList.txt file or ContactsList.xml file where the contacts were stored in a persistent fashion. This all became the backend logic. To interact with this backend logic, we had created a Webpage which had ContactsManage as the class within it which we had derived from the Page class. ContactsManage had different helper functions and the button handlers. For example, we had a button handler for Add, Search, Delete and Save button. ContactsManage become a front end. In our SQL based contact book, more or less things are similar except for FileContact or XmlContact class now we propose to have a SqlContact class and in place of file ContactsList.txt or ContactsList.xml, we are going to have a ContactsDB.mdf file. SQL database files are always stored with .mdf extension. So ContactList and ContactsManage we are going to change in this project whereas we are going to derive a SqlContact class from the Contact class and we are going to create this class completely newly. As we had copied Contacts project into the Contacts_Xml folder, likewise we can copy Contacts_Xml project into a new folder wherein we can have a copy of Contacts_Xml and then try to make changes in it.

Steps Involved

1st thing is we have to copy everything. Subsequent to that, we are going to create a database and a Contacts table inside that database. Then we will create a SQLContact class which is derived from the Contact class. Subsequent to that we will update the ContactList class which is a backend class and which is going to maintain a list of contacts for us. Then we will update ContactsManage class which is the front end class.

Step 1: Create Database

Go to the Solution Explorer. As shown in the slide, there is C:\..\Contacts_SqlSimple. It means we must have copied our Contacts_Xml project into a new folder i.e. Contacts_SqlSimple. Now right click on the App_Data folder. Select Add New Item from the menu that pops up. As soon as we do that, a new dialog would appear. Select SQL Server Database from the dialog that would appear. Give the name as ContactsDB.mdf and click on Add. Now the ContactsDB.mdf file gets created within App_Data folder.

Result

Now as shown in the slide, on the screen we would start noticing a Database Explorer is come up in place of Solution Explorer. In the Database Explorer, we can notice that there is a ContactsDB.mdf entry already present. Several items are present here. In ContactsDB.mdf, we can create database diagrams. Means when we create multiple tables, what all tables in the database, what is the relationship between those tables, all of that can be depicted in a pictorial fashion using the database diagram. In the Tables branch, we can notice all the tables that we would create in a database. So if
we create one table then it goes into this Tables entry. Then there can be Views, Stored Procedures,
Functions, Synonyms, Types and Assemblies. Select Tables from here. Then right click on that.
Select Add New Table from the menu that pops up.

**Slide 7**

The moment we select Add New Table, the screen would change. On the screen a window would
appear in which we have Column Name, Data Type and Allow Nulls as three columns. We propose
to create three columns in our database. We know that within a database there are going to be
multiple tables. We are going to create a 1st table. It asks how many columns we are proposing to
have within the table. We propose to have three columns; one for Name, another for Telephone
number and 3rd for the Mobile number. Each of these columns has certain properties. For example,
each column will have name. Then each column will have some or the other type. Type would
indicate what kind of data we would store under this particular column. Allow Nulls indicate should
we allow a Null entry within that column or not. For that the checkbox is given. If we click on that
checkbox then in that case a null value will be allowed in this column. Means whenever we add a
new row to the table, will this column be allowed to have a null entry or not is decided by Allow
Nulls column. We try to create three columns. In the 1st column we give Name of the person. So
column name is going to be Name and data type we give to it is nvarchar(50). nvarchar(50) indicates
that we propose to have maximum of 50 characters against a Name. Means even if somebody has a
long name, we would be able to accommodate that. Then we have a Telephone as the column name
and its type is nvarchar(15) indicating that maximum of 15 characters can be accessed as a mobile number. In the third column, we would give column name as Mobile with type as nvarchar(15)
indicating that maximum of 15 characters can be accessed as a mobile number. We might think that
nvarchar(50) as we give type for the Name then there would be too much wastage of space because if
we give Name of the person as John then John needs only 4 characters and there is large amount of
wastage of space. But that is not so. Even though we are specifying nvarchar(50), it means is length is
going to vary. So we have a var in nvarchar. It is going to vary from 1 to 50 characters. Minimum can
be 1 and maximum can be 50. If the person name is John then 50 characters will not be sacrificed. It
will definitely save based on what is the length of the name. Type the name of the column directly in
the textbox which is against the column name. Data Type column shows all the available types. There
are other data types also available. We can have types like smallint, int, bigint. smallint is like a short
whereas bigint is like a long int. In addition to that we can have datetime field, money field stores big
numbers and we also have varbinary which can be used for storing binary data like images. For all
the three columns Name, Telephone and Mobile, we uncheck the Allow Nulls checkbox.

**Slide 8**

Three columns stand created Name, Telephone and Mobile within our ContactsDB.mdf database.
These columns will have to belong to some or the other table. To store all these columns in a table,
we have to click on the Save button. The moment we do so, it will ask for the name of the table. The slide shows the names that we had given for columns as well as their types and also we can notice the
Allow Nulls field in all the three checkboxes are unchecked or empty. As soon as we click on the
Save button, a dialog will pop up where we have to provide the name for the table. Give the name of
the table as Contacts and click on OK.

**Result**

As soon as we click on OK, the database stands created. A shown in the slide, if we look at the
Database Explorer, we can see inside the Tables there is a Contacts table entry is also present. Those
rows and columns indicate that it’s a table and it’s a visual way of communicating to us and Contacts
is the name of the table.

**Connecting To SQL Server**
Now through our application, we need to connect to the SQL server. That is typically done using a connection strings. The reason why these connection strings are used is that tomorrow we should be able to move the database to some other place other than our own machine and still be able to communicate with that database by connecting to it without being required to make changes in our program. We just make a change in the connection string and we would be able to connect to a database even though it is moved to some other machine. So that’s the way to access the database without hard-coding anything in the project that we are creating. Connection string is a string without any break, one complete string. As shown in the slide there is a connection string having different name, value pairs like Data Source=\SQLEXPRESS; AttachDbFilename=|DataDirectory|ContactsDB.mdf;Integrated Security=True;User Instance=True. So the connecting string is going to begin at Data Source and end at True. Within the connection string there is different name, value pairs which are separated using ;. The connection string specifies where is this database server and the credentials using which we can connect to that database server. So, physical location of the database server as well as username and password kind of thing using which we can connect to that database server are given. That is called as credentials. Within the connecting string shown in the slide, we cannot see the username and password. Here credentials are given by just giving name, value pairs by saying Integrated Security=True. When we say Integrated Security=True, it means is use the same login name and password to connect to the database, the one that we use for logging into Windows, same username and password we used for connecting to the database server. Connection string is similar to URLs. In Data Source=\SQLEXPRESS, \ indicates that our database is going to be present in the local machine. SQLEXPRESS is the default instance name of the database server. Microsoft has come up with different database servers. We are using SQL Server 2005 Express Edition. So default instance name is SQLEXPRESS. Then we say AttachDbFilename=|DataDirectory|ContactsDB.mdf. DataDirectory is written within | |. DataDirectory finally expands to C:\ASP.NET\Contacts\App_Data. To represent this whole thing we had written DataDirectory. Advantage of mentioning it in this manner is that tomorrow if physically the location of ContactsDB.mdf changes, still within our connection string we have not hard-coded the path for that mdf file. We will just mention DataDirectory wherever the ContactsDB.mdf file is present physically that path will be pasted in place of DataDirectory. This ensures that we do not hard-code any entry within the connection string. If we hard-code data directory then there would be a problem in deployment when we move the database from one machine to another or one folder to another or one path to another. If the path changes then in that case deployment will become difficult. Integrated Security=True means use current windows account User ID and Password. If we want, we can also avoid this Integrated Security=True and then use SQL’s username and password to make a connection to the database. SQL server internally maintains a table of valid usernames and passwords and if we do not want to use the Integrated windows Security, we can think of connecting to the database server using the login name and password by mentioning that within the connecting string. Then we say User Instance=True. Whenever we mention User Instance=True in the connection string, it means create a separate server instance for the current user. There can be multiple instances of MSSQL server’s exe file. Name of that is sqlserver.exe. Whenever this sqlserver.exe is executed, we say that there is an instance of sqlserver.exe now running in memory. There can be multiple such instances running in memory. Each instance can have different databases associated with it such that even if one server exe crashes still only the databases associated with that, those who would crash others can continue to be alive. That’s the idea behind permitting multiple instances of sqlserver.exe in memory. Databases are not going to be shared across multiple instances. The default name given for the instance is the name of the computer if it is MSSQL server 2005 and the default name happens to be SQLEXPRESS if it is MSSQL server 2005 Express Edition.

**Step 2: Create SQLContact Class**

We will right click on the App_Code folder. When we do so, a menu will pop up. Select Add New Item... from the menu that pops up. When we do so, a dialog will appear. Choose Class template from the dialog that appear and give the name of the class as SQLContact.cs. Then import using
System.Data.SqlClient at the top. We have classes like SqlConnection, SqlCommand, SqlDataReader and SqlParameter in SqlClient namespace. There are many other classes in SqlClient namespace. SqlConnection allows us to connect to Microsoft SQL Server. SqlCommand class is sued to send a command and get the response. SqlDataReader class reads the data row by row from a collection of rows that are returned after executing a command using SqlCommand class. To pass parameters to SQL command we can think of using a SqlParameter class.
SQLContact Class

In this lecture you will understand:

- Types of SQL FCL classes
- Methods of SqlCommand class
- SQLContact class
- Why is it different
- How to implement its methods
SQL FCL Classes

Different classes that are available are SqlConnection, SqlCommand and SqlDataReader. Along with that one important class is SqlParameter. In this project we do not need SqlParameter class. The job of SqlConnection is, it lets us establish a connection with the Microsoft SQL Server. Unless we establish a connection with the database server, there is no question of issuing any commands to the database server. So, 1st job at hand is always to establish a connection with the database server and to do that SqlConnection class is used. Then once the connection stands established we need to issue the commands and to issue the commands we propose to make use of the SqlCommand class. We will have to send the command and then get a response for that command from the database server. That is managed for us by the SqlCommand class. The SqlDataReader class allows to read data rows that are returned after having executed the command using the SqlCommand class’s methods.

SqlCommand Methods

The three methods in SqlCommand class that are of our interest are ExecuteReader(), ExecuteNonQuery() and ExecuteScalar(). Each one of the method has a different ability with them. Each one of them ultimately executes a SqlCommand which is commonly known as an SQL query. Query word is used for SQL commands. So ExecuteReader(), ExecuteNonQuery() and ExecuteScalar() is ultimately going to execute some or the other SQL command. When ExecuteReader() goes to work, it returns all the records that qualify a particular query. For example, we may fire a query which looks like Select * from Customers. When we say this query then Customers become the name of the table. * indicates all the columns in that table. This query gives all the records and fields present in that database table called Customers. Once we establish the connection with a particular database server, there onwards we have to deal only with the tables inside that database. So ExecuteReader() returns all the records that qualify a particular query. We can also do selective searches. We can say Select * from Customers where AmountOfPurchase > 10000. Then not every record would qualify this query. The records which qualify this query only those records would be returned by this ExecuteReader(). Then there is ExecuteNonQuery() which executes the query. Typically this query could be insertion of records into the database or updation of existing records. It returns back is how many rows in the database got affected as a result of executing this query. For example, we may decide to update all the records in the database based on certain condition. We wish to update all those records wherein amount purchased in more than 10000, we propose to give 10% discount. Only some records would qualify this query and only some of them would get updated. Whenever that updation takes place, if we had done that using ExecuteNonQuery() then out of 1000 records if only 50 got affected then 50 will be returned in the form of a number which indicates the number of records or rows got changed or affected. So the return value of ExecuteNonQuery() indicates how many rows really got affected as a result of execution of the query. Then we have ExecuteScalar() which returns the first column of the first row whenever we fire a query. Typically ExecuteScalar() can be used to count how many records are there in the database which satisfy a particular criteria or we can use the Min command or the Max command to find out the minimum value that is there in a particular column or the maximum value that is there in a particular column or we can also think of finding out average of a particular column. In all such cases, one value is to be returned i.e. the value present in the first column of a first row or sometimes we may want to update a particular record or may be insert a particular record. If we have a ordered table and we insert the order information then first column will be the order id, that order id would be returned to us when this ExecuteScalar() method is called.

Step 2: SqlContact class

In the XmlContact class, we used to have only a 0-argument constructor, WriteToXml() method and LoadFromXml() method. WriteToXml() method is used to write only one contact information and LoadFromXml() method is used to read only one contact information. To be able to do so, we used
to pass the suitable reader object to WriteToXml() and LoadFromXml() method. The SqlContact class is going to be a bit different. There are several methods. Apart from 0-argument constructor, we also propose to have Insert( ), Delete( ) and Update( ) methods here. We are also planning to have three more methods; LoadFromReader( ), ExecuteCommand( ) and ExecuteReader( ). LoadFromReader( ) is going to read only one record from the reader. Insert( ), Delete( ) and Update( ) allows to insert a new record, delete an existing record or update an existing record respectively. Each one of these Insert( ), Delete( ) and Update( ) to actually do an insertion, deletion and updation are going to call the ExecuteCommand( ) method. Since the code to execute the command is similar for all three, rather than writing the same repetitive code for Insert( ), Delete( ) and Update( ), we propose to take out that code and write that in ExecuteCommand( ). ExecuteReader( ) is going to read all the records from the database. In the FileContact or XmlContact project, we were managing our contact book through a file like a txt file or XML file. There we did not have a mechanism to insert one record into the XmlContact book or the text based contact book or delete one record or update one record. Now we are making use of the SQL database based contact book, it’s not our headache to figure out how one record can get inserted or one can get deleted or one can get updated. That headache is SQL server is going to take care for us. Hence, we propose to have Insert( ), Delete( ) and Update( ) methods within the SqlContact class.

**SQLContact Class**

We have SQLContact class derived from the Contact class. At the beginning we have a public constant string called connectionString and within the connection string we will have a Data Source as .\SQLEXPRESS. Then the name, value pair that is going to follow Data Source is AttachDbFilename=|DataDirectory|ContactsDB.mdf. Then Integrated Security=True and User Instance=True. This is one full string. We have concatenated different name, value pairs to build one common full connectionString. Then we have to make sure that this connectionString is used for actually connecting to the database. But by making it as a public const string, we are making sure that it’s available to us wherever we need it. Then we will create a method called ExecuteCommand( ) which is called from Insert( ), Delete( ) and Update( ) methods. Since the code that we propose to do to execute a command is going to be common for Insert( ), Delete( ) and Update( ), hence we are taking out that code and writing that in ExecuteCommand( ) method. This method receives a queryString which indicates what command is to be executed on the connection that we propose to establish. ExecuteCommand( ) is a shared or static method. Means anytime this ExecuteCommand( ) is to be called, we will have to call it by saying className.method-name. Classname is SQLContact and method name is ExecuteCommand( ). This method connects to the SQL server and executes the desired command i.e. the command which is passed to it in the string called queryString. Whatever command we receive in queryString that’s the one ExecuteCommand( ) is going to actually execute. ExecuteCommand( ) first will have to create a SqlConnection object. It does so by saying SqlConnection conn = new SqlConnection( connectionString ). At this juncture the connection has not established, we had merely created a SqlConnection object. Once the object is created then we can also create a new SqlCommand object to which we pass the queryString and a conn reference. At this stage we have merely created two objects, a connection object and a command object. Once both these objects are created, now the connection would be established when we say conn.Open( ). This is responsible for establishing the connection with the database server. Now that the connection stands established, on this connection we would now try to execute the command by saying cmd.ExecuteScalar( ). It executes the command on the specified connection object, the one which stands opened and information about this connection object is there with SqlCommand object because when we created the SqlCommand object, we can see, the 2nd parameter that we had passed to the constructor of SqlCommand is a connection reference. It stores this connection reference in one of its private variables. So cmd.ExecuteScalar( ) is going to finally execute our command on the connection that stands established. Whatever values returned by ExecuteScalar( ), same thing we would return back and that type we are giving as object when we are defining the ExecuteCommand( ) method. So return type of ExecuteCommand( ) is object.
Read, Insert

To read a record, we are going to use a method called LoadFromReader(). It receives SqlDataReader reference and we write three lines. By using the indexed property we access the Name, Telephone number and Mobile in the SqlDataReader object called reader. We are saying this.name = (string) reader["Name"], this.telno = (string) reader["Telephone"] and this.mobile = (string) reader["Mobile"] where this.name, this.telno and this.mobile are the elements present in the SQLContact object. name, telno and mobile are the properties which we have derived from the Contact class. We are using these properties to setup the name, telephone number and the mobile number. Then we have an Insert() method in which we are going to construct a query that is needed for inserting a record in the database. When we say string queryString = string.Format(). This method allows to construct a string. We are going to construct a string which reads like INSERT INTO Contacts (Name, Telephone, Mobile) VALUES ('Ramesh', '223344', '9856712345'). That’s the final queryString that we propose to construct. Contacts is the name of the table that we had created in a database called ContactsDB.mdf. To be able to do so, we are saying "INSERT INTO Contacts (Name, Telephone, Mobile) " + "VALUES ( '{0}', '{1}', '{2}' )", name, telno, mobile. This string is passes as an argument to the string.Format() method. This string finally would get constructed in the form of a string which is shown in the box in the slide. Once this queryString stands constructed, we call ExecuteCommand(queryString) where ExecuteCommand() actually going to open the connection and fire this query.

Delete

In the Delete() method, we are again going to construct the queryString. Subsequent to that we call ExecuteCommand() method. This time the queryString will look like DELETE FROM Contacts WHERE Name = 'Ramesh' AND Telephone = '223344' AND Mobile = '9856712345'. So for this query to fire, we will construct a query as "DELETE FROM Contacts WHERE " + "Name = '{0}' AND Telephone = '{1}' AND Mobile = '{2}'", this.name , this.telno , this.mobile and then call ExecuteCommand(queryString) method once again. Once again a connection is opened with the database and this query is fired on it. As a result of which that record in the database got deleted whose name is Ramesh, whose telephone number is 223344 and whose mobile number is 9856712345.

Update

In the Update() method, we will again construct a queryString. This time in place of INSERT and DELETE, a UPDATE SQL command is used. While updating set the Name, telephone number and mobile number to a set of values and which name, telephone number and mobile number has to be updated the one which has name, telephone number and mobile as present in this.name, this.telno and this.mobile. So whatever n, t and m are received, those are the new values whereas this.name, this.telno and this.mobile are the current values present in the database. So it will search according to whatever is present in this.name, this.telno and this.mobile and once that is searched, it will change these values to the one that are there in n, t and m. So wherever 0, 1, and 2 are present, there values of n, t and m were go whereas 3, 4 and 5 would be replaced with this.name, this.telno and this.mobile. So create the string using string.Format() and then using it call again ExecuteCommand() to execute this command after having established the connection with the database. For example, as shown in the slide, the query is UPDATE Contacts SET Name = 'Ramesh K', Telephone = '556677', Mobile = '9912356745' WHERE Name = 'Ramesh' AND Telephone = '223344' AND Mobile = '9856712345'. So use one set to locate the record and use another set to write the final values on top of the record that got searched.

Read All Records
ExecuteReader() method reads all the records from the database. queryString would be the command that would be executed on the SQL server. ExecuteReader() is going to execute this command on the specified connection object and moreover its going to return the SqlDataReader with rows that qualify the query. Means multiple rows are going to return in the form of SqlDataReader. So return type of ExecuteReader() is SqlDataReader. So create a SqlConnection object. Having done that create a new SqlCommand object. As yet the connection has not been opened. Now open the connection and having opened the connection, execute the command that is there in queryString by saying cmd.ExecuteReader(). Means now we are calling the 2nd method from the SqlCommand class and the one which is capable of returning multiple rows. So cmd.ExecuteReader()’s return type is SqlDataReader and that SqlDataReader is we are returning.
In this lecture you will understand:

- Details of ContactList class
- Details of ContactsManage Class
Step 3 – ContactList Class

In the XML based contact book, the ContactList class was supposed to maintain in memory a collection of XmlContact objects and then to perform operations on this ArrayList we had functions like a 0-argument constructor and a 1-argument constructor. Then we have additional functions like LoadFromXmlFile() which is used to load all the contact information from XML file and build the ArrayList and SaveToXmlFile() is used to do the reverse. Whatever contacts are there in the ArrayList right now, we used to write them into an XML file using SaveToXmlFile(). Then we had functions like Search(), AddContact(), DeleteContact() and UpdateContact() to carry out searching, addition, deletion and updation of an existing contact. We are done with that. In the ContactList class that we propose to create in the SQL based contact book application; here we do not need 1-argument constructor. This is so because in the XML based contact book’s ContactList class’s 1-argument constructor, we used to load the contents of the XML file such that the contacts can be displayed whenever the webpage is requested, the current list of contacts is used to get displayed. That is no longer necessary because all the contacts are there in the SQL database. We do not have to have a 1-argument constructor for it. We will have LoadFromSql() function which will load the contacts from the SQL database into the ArrayList. Then we will have Search(), AddContact(), DeleteContact() and UpdateContact() functions.

ContactList Code

At the beginning, we will have to import using System.Collections and using System.Data.SqlClient namespaces. In the ContactList class, we have a protected ArrayList m_list. In the constructor, we would create a new ArrayList object. So any time a ContactList object is created, we can assume that an ArrayList object would also get created. Then we expose a property List and it is going to return m_list. So anytime from outside the class, somebody to have an access to the ArrayList, he can utilize this property List and he would then be able to access the ArrayList from this class. Then we have the Search() function. In the Search() function, we are going to receive a string needle. This string would represent the name of the person whom we wish to search in the contact book. We need not supply the entire name to the Search() function. Means if part of names we want to search through the Search() function, we can do so. Inside this function, in a for loop, go through the ArrayList. Each time through the ArrayList, we should try to find out the name that we have in needle, is it there in name of the current contact as we traverse through this ContactList. For that we first create a Contact reference. This is necessary because when we extract the contact from m_list that time we have to cast it appropriately. Then we can check if (con.name.ToLower().Contains(needle.ToLower())) and if it is true then we return con. Outside the for loop we should return null i.e. in case of failure we return null.

Slide 5

To be able to load the data from the SQL database, we are proposing to use LoadFromSQL() function. Here we will have a SqlDataReader reference reader which is set to a value null. In the try block, we would first create an ArrayList and stores its reference in m_list. Then we will call a method called ExecuteReader() which is static function of SQLContact class and to that we are passing a query string SELECT Name, Telephone, Mobile FROM Contacts. ExecuteReader() would collect this query and then fired that query after having opened the connection with the database. It returns SqlDataReader. Now the reader that we have, we must use it to iterate through all the records that qualify this query which is done in the while loop. We have to keep saying while (reader.Read()). The Read() function loads the next row from the result of the query means if 10 records qualify the query then the next record that is there from the qualified list of records would be loaded and this would continue to happen till the time we are not done with the last record from the list of qualified records. Qualified indicates those records which were searched successfully based on the queryString. The Read() method returns true till the time we have records to go to. If we have gone through the
last record of the qualified list of records then reader.Read( ) return false and the while loop would come to an end. Inside the while loop, we must first try to create the SQLContact object. Its reference is collected in con. Then call SQLContact’s function con.LoadFromReader ( reader ). It used to extract from the reader i.e. from the current record it used to extract the name, telephone number and mobile and assign it to the name, telephone number and mobile of the SQLContact object. Once this assignment is over then we can add it to the ArrayList using m_list.Add ( con ). So this reference is added to the ArrayList. In the finally block, we propose to check if ( reader != null ) and if it is not null then we close the reader using reader.Close( ).

Slide 6

Any time the contact is to be added the SQLContact c would be received in the AddContact( ) function. We have to use that c to insert contact by calling SQLContact’s Insert( ) function into the ArrayList i.e. c.Insert( ). Likewise in the DeleteContact( ), which contact is to be deleted that information will come to us as a parameter in SQLContact c. In DeleteContact( ), we do c.Delete( ) which will eliminate the contact from the contact book by calling the Delete( ) function of the SQLContact class. On similar lines we will have a UpdateContact( ) function which receives the contact which is to be updated. That information would be in the SQLContact c. n, t and m would be the new name, telephone number and mobile which are to be updated into the database. For that we will call c.Update( n, t, m ) method. We do not have to pass the current contact information because we are calling Update( ) using current contact i.e. c.

Step 4 – ContactsManage Class

ContactsManage is the front end class which is derived from the Page class. If we compare ContactsManage of XML based contact book with ContactsManage of SQL server based contact book, there are many methods similar. Out of these methods shown in the slide AddContactRow( ), AddHeader( ) and ShowEditTable( ) remain same. They are not going to change at all. Other functions like Page_Load( ), AddButton_Click( ), WriteToSql( ), UpdateListTable( ), LoadFromSql( ), SearchButton_Click( ), DeleteButton_Click( ) and SaveButton_Click( ) would change.

List Contacts

The Page_Load( ) method always receives an object and an EventArgs. Within this, we have to list the contacts on the webpage. To do so we say ListTable.Rows.Add ( AddHeader( ) ). AddHeader( ) adds the three column headings. ListTable is the name of the table that we had created. Subsequent to that we call a function called LoadFromSQL( ). In the LoadFromSQL( ) function, we are going to create a new ContactList object. As soon as we create the new ContactList object, we are sure that a new ArrayList object should also get created. Once that has been done we can call clist.LoadFromSQL( ) which loads the data from SQL database and populate ArrayList with references to multiple SQLContact objects. Now we have to display this list of contacts on the webpage. To do that we will run a loop going from for ( int i = 0 ; i < clist.List.Count ; i++ ). Each time through this loop, make a call to the function AddContactRow ( ( Contact ) clist.List [ i ] ).

Add Button Handler

The name of the handler is AddButton_Click( ). Within this handler we have to first verify the validity of the name that has been entered and the validity of the telephone number that has been entered. So we check if ( nameInput.Text.Length > 0 && telInput.Text.Length > 0 ) and if the condition is true means the textboxes contain something. Then we need to extract that and write the information into the SQL database. So we call a function WriteToSql( ). This function creates a new SQLContact object and then fill it with suitable data. Once this is done, it’s going to insert that SQLContact into the database and its also going to update our list of contacts. Hence we say
UpdateListTable( ), that’s the one that is going to update the list of contacts on our webpage the moment addition of a new contact is done.

**Helper Functions**

WriteToSQL( ) and UpdateListTable( ) are helper functions. In WriteToSQL( ) function, we are going to create a new SQLContact object and then we are going to replace any commas that are there with spaces by saying con.name = nameInput.Text.Replace ( ',', '' ), con.telno = telInput.Text.Replace ( ',', '' ), con.mobile = mobInput.Text.Replace ( ',', '' ). Once that is over we will create a new ContactList object and call a AddContact( ) method on it i.e. clist.AddContact ( con ). AddContact( ) function updates the ArrayList. In the UpdateListTable( ) function, we will first clear out everything that is there in the ListTable right now. To do that we say ListTable.Rows.Clear( ). Then we say ListTable.Rows.Add ( AddHeader( ) ) which adds new header of Name, Telephone number and Mobile and then call LoadFromSQL( ). LoadFromSQL( ) loads all the contacts from the SQL into the ArrayList. It also writes all the rows that are there in the ArrayList into the webpage.

**Search Button Handler**

In the Search button handler, we are first going to check whether the name that has been typed, its length is more than 0 or not i.e. if ( nameInput.Text.Length > 0 ). If it is more than 0 then in that case, we will first create a new ContactList object. As a result of which a new ArrayList object will also get created. Now we will call LoadFromSQL( ) which first loads the data and then populate the ArrayList. Then call the Search( ) function of the ContactList class and while doing so, pass to it the name that has been entered in the AddTable for carrying out the search, convert that into lowercase and pass it to the Search( ) function. Search( ) function would search out the name, return its reference in con and then check whether it is null or not i.e. if ( con != null ). If con is not null i.e. search has been successfully carried out then we display the information of this particular contact on the label DelContact i.e. DelContact.Text = con.name + "," + con.telno + "," + con.mobile. Once this is done, we can store this in the Session by saying Session [ "contact" ] = con. Then we say AddTable.Disabled = true, DeleteButton.Enabled = true and then the EditTable would be shown with exactly the same contact name, telephone number and mobile number. EditTable is useful for editing a record once the search is over. So we say ShowEditTable ( con ).

**Delete Button Handler**

In the Delete button handler, we would extract the contact that we had earlier saved in the Session and then call the Delete( ) function of the SQLContact class by saying con.Delete( ). Here searching from the database would be done by the SQLContact class’s Delete( ) function. Then we update the ListTable by calling UpdateListTable( ) function. Then we can say AddTable.Disabled = false, EditTable.Visible = false, SaveButton.Visible = false, DeleteButton.Enabled = false and DelContact.Text = "".

**Save Button Handler**

In the Save button handler, we would extract the contact that we had earlier saved in the Session. Then we will call the Update( ) function using SQLContact object. We will pass the name, telephone and mobile number from the EditTable i.e. we say con.Update ( editName.Text, editTel.Text, editMob.Text ). This change has to be reflected in the list of contacts that are there on the webpage. To achieve that we will call UpdateListTable( ). Then we say AddTable.Disabled = false, EditTable.Visible = false, SaveButton.Visible = false, DeleteButton.Enabled = false and DelContact.Text = "".
Secure Database Programming

In this lecture you will understand:

- Database Security concerns
- Secure database programming
- Parameterized Queries
Get ready to be shaken!

Whenever somebody tries to compromise the security of the database, lot of very bad and serious things can happen. For example, tomorrow somebody might steal confidential information from the database. Worst still, somebody might even want to modify the information that is there in the database. Suppose we have a stores database where different products are there. Again prices of those products are there. If somebody actually is able to change the value of the price of different products, imagine what will that mean to our business? In that case disaster is not far away. So one is that somebody can steal the data, 2nd is that somebody can modify existing data and worst still somebody with sadistic intention might even end up deleting data from the database. That’s going to herd real bad. Problem is people are able to achieve this and that is because of the laxity on the part of the database programmers or the website programmers. So these threats are real. So we have to take the threats to the database security very seriously. For example, we might get a threat from a cracker. The word Hacker has got a bad reputation even though a hacker is somebody who is going to give insight into the implementation of the system and try to make good use of that and device mechanisms to prevent others from breaking into the system whereas cracker is somebody who is having all the time malicious intentions of doing bad things on the website. We might get a message or a threat from the cracker that I am going to delete your entire database. Typical response for that is don’t bother, I know what I am doing. I have secured my databases. There is no way that you can really touch me. Then the cracker says, do you want me to show this to you? Some people say, go ahead and show me. Then he gets to work and he inserts a new entry into your database and no sooner does he do that, all your data has gone. How does he achieve this? All that he does is, if the database has Name, telephone number and the mobile then he tries to add name, telephone number and mobile with values like may be Bad for the name, 11223344 for the telephone number and 11’ ) ; DELETE FROM Contacts ; -- for the mobile. We know that this is the correct mobile number but unless we have made the checks on this input in our programming of the website then this mobile number would get accepted and as soon as it gets accepted then he has attacked your database by using a technique known as SQL injection.

Where are my contacts!

To insert the new contact information into the Contact book, we had used a query like INSERT INTO Contacts ( Name, Telephone, Mobile ) VALUES ( '{0}', '{1}', '{2}' ). Now it gets expanded to INSERT INTO Contacts (Name, Telephone, Mobile) VALUES ( 'Bad', '11223344', '11' ) ; DELETE FROM Contacts ; --' ). -- is a syntax that is used for giving a comment in a SQL command. So anything that begins with -- is treated as a comment. The moment he gave 11' ) ; DELETE FROM Contacts ; --' it got accepted as a valid mobile number but the value for the mobile number is now become only 11. So INSERT command ends right there after 11’ ) then there is a ;. That ; is a facility of separating two SQL commands. He has actually achieved is, he has been able to inject a new SQL command into our existing SQL command. Our existing SQL command is the INSERT command, that is going to get executed any how. 'Bad', '11223344', '11' becomes the new contacts name, telephone number and mobile number. But once this insertion is done, now the next SQL command is going to get executed and that’s going to be disasters because we are going to delete all the records from the contact book. DELETE FROM Contacts is going to delete every single record that’s there in the Contacts table. So by doing this SQL injection, he has been really able to delete all the records from the database. The lesson that we have to learn from this is never underestimate the database security threats. They are present and we have to see how to tackle them.

Dealing with “SQL Injection” attacks

SQL injection means SQL statements get injected into existing SQL commands. The way within our case, the DELETE command gets injected within the INSERT command. They became separate command, both got executed. Insertion did happen; deletion also took place. It’s a very common
flaw. Moreover, the security implications of that are quite severe. To deal with this SQL injection attacks, we should never trust input that comes from the different textboxes because that’s when the user types new input. Never trust it. Don’t think that users will always use well-formed input. You got to validate before using any input in your database. We should never attempt to build a “query string” directly from the input unless we have validated it. Moreover, instead of building the query string directly from the input, we should use parameterized queries which MS SQL provides to us. We should also try to reject special characters like ; ' --. These characters we should never be accepted in an input. So if we notice these characters in the input, reject them. We have to do this programmatically because once he enters and you receive it, we will have to validate it naturally in a programmatic fashion. So better way to build a query is that to pass “parameters” to the INSERT command and those parameters should be type safe parameters and MS SQL Server supports parameterized queries. Once we know how to build them, we can know thoroughly well that nobody is going to touch the data within the database.

**Parameters with dynamic SQL strings**

We will take same SQLContact book and try to improve the Insert( ), Delete( ) and Update( ) methods that we wrote within the Contact book. In the Insert( ) method that we had in the SQLContact class, we had constructed the query dynamically based on the input that we had received from the user in the three textboxes. We will first declare a string query and then we say "INSERT INTO Contacts (Name, Telephone, Mobile) " + "VALUES (@name, @telephone, @mobile) ". Here we are not going to pick name, telephone and mobile straightway from the fields. We are going to construct that using @name, @telephone and @mobile which are really going to act as parameters. @name, @telephone and @mobile are variables which we are going to assign certain values. In this function, we also need to create a SqlConnection object based on the connectionString. Then we will construct the SqlCommand object which needs the query as well as the SqlConnection object’s reference. Then we say cmd.Parameters.Add ( new SqlParameter ("@name", this.name ) ). This way we are trying to construct the parameter. Parameters is the property within the SqlCommand class. This property returns a reference to a SqlParameterCollection. The SqlParameter object contains a name and its value. @name is the parameter and its value will be whatever is present in this.name. this.name gives us an access to the property of the base class. Similarly we say cmd.Parameters.Add ( new SqlParameter ("@telephone", this.telno ) ) and cmd.Parameters.Add ( new SqlParameter ("@mobile", this.mobile ) ). Now we can open the connection by saying conn.Open( ). Now the connection with the database would get established. Now we can execute the SQL command by calling cmd.ExecuteScalar( ). Once this is done, now the actual addition of a new record into the new database would happen. This has happened in a safe manner because we have not constructed the string dynamically. On the basis of the input that we received, we pass whatever input we receive itself as a parameter.

**Change Delete( )**

In the Delete( ) function, we will have a query which is "DELETE FROM Contacts WHERE Name = @name " + "AND Telephone= @telephone AND Mobile = @mobile". Once the query is constructed, we can create a SqlConnection object using the connectionString. Then create a SqlCommand object and pass the query as well as the SqlConnection reference to the constructor. Then add parameters to the SqlParameterCollection by saying cmd.Parameters.Add ( new SqlParameter ("@name", this.name ) ), cmd.Parameters.Add ( new SqlParameter ("@telephone", this.telno ) ) and cmd.Parameters.Add ( new SqlParameter ("@mobile", this.mobile ) ). Once this is done, now we can open the connection using conn.Open( ) and fire the query on this connection by calling cmd.ExecuteScalar( ).

**Change Update( )**
In the Update( ) function, we will first construct the query as "UPDATE Contacts SET Name = @newname, " + "Telephone = @newtel, Mobile = @newmobile WHERE " + "Name = @name AND Telephone = @telephone " + "AND Mobile = @mobile". Name should have the value as present in @newname, Telephone should have the value as specified by @newtel and Mobile should have a value as specified by @newmobile. Before we set the values we must be able to search the record. So we have used the WHERE clause. In the WHERE clause, we will search for the name, telephone and mobile as specified by @name, @telephone and @mobile. Means we are having 6 parameters here; 3 of the old contact and 3 of the new modified values. All these we will have to be added to the Parameter Collection. First we will create the SqlConnection object and then create the SqlCommand object. Then call cmd.Parameters.Add ( new SqlParameter ( "@newname", n ) ), cmd.Parameters.Add ( new SqlParameter ( "@newtel", t ) ) and cmd.Parameters.Add ( new SqlParameter ( "@newmobile", m ) ) i.e. we were added new SqlParameter object each time. Old values are assigned using cmd.Parameters.Add ( new SqlParameter ( "@name", this.name ) ) and cmd.Parameters.Add ( new SqlParameter ( "@telephone", this.telno ) ).

**Slide 9**

Old value of mobile is assigned using cmd.Parameters.Add ( new SqlParameter ( "@mobile", this.mobile ) ). Then we will open a connection using conn.Open( ) and then fire the query by calling cmd.ExecuteScalar( ).

**Moral**

We have to validate every single value that we receive as input. Assume that every input that is going to come from the user is harmful till the time it is proven otherwise i.e. validates it thoroughly. Look for valid data and reject everything else other than that valid data. Constrain, reject, and sanitize user input by performing type checks thoroughly, perform length checks correctly. Range checks are also important where the value of a particular field to be within a range and also formatting is checked.
Employee Management Website

In this lecture you will understand:

- Different types of Web Controls
- Employee Management Web application
- Steps involved in building the application
- Displaying Flash files on web pages
Web Control Categories

The different categories of Web controls fall under five major categories. First amongst them is the Standard Web Controls. This Standard Web Controls are numerous. For example, we had used Textbox to receive name, telephone number and mobile number and Buttons. While using web controls in the application, we will find that there are parallels for these web controls in the form of HTML controls. HTML controls are older way of creating the webpages whereas Web controls are more modern way of dealing with different controls in the webpage. So Textbox, Button these are standard controls. In addition to that we will also see Radio buttons are frequently used in a webpage or a drop down ListBox is very commonly used in a webpage. Apart from the standard controls, there are many other controls. One of them is the Data control. There are very powerful controls under Data control category. One of them happens to be a GridView control and another is a SqlDataSource control. GridView is basically used to display the data in a columnar format whereas SqlDataSource is used for tying different controls with different data sources. The third category of controls is the Validation controls. Lots of times when we visit a website and we are required to register ourselves there and when we do this registration process, we are being asked to do some form filling. In the form filling, we may be required to type name, email address, postal address, etc. All these details we are required to fill in. In the process of doing so one or two fields we get to ignore. Then we click on the Submit button. No sooner did we click on the Submit button, we are immediately flashed the error saying that one or two fields we are forgotten to enter any data into. Those are considered to be the required fields. This can be done using ReqdFieldValidator control. We are required to enter things like Email addresses. If somebody enters the Email address without the @ symbol then that’s an invalid Email address. So whenever Email addresses are received, it is verified whether that Email address is a valid Email address or not and that can be done by using a standard control called ReqExpValidator or when a phone number is entered whether the country code, area code and the actual number is there or not can again be figured out using a RegExpValidator. Many types of validator controls are present. But we are mentioned only two of them. Whenever we deal with web controls, almost every time we are also required to deal with validation control because we cannot assume that any input that is coming from the user is indeed in the correct form. Hence validation of that input is utmost necessary and this is achieved using different validation controls that Visual Studio has made available to us. In addition to that we have Navigation controls. For example, typically at the top of a webpage we see menu items through which we can navigate to different parts of a website. There is a SiteMapPath control available which lets the user see at a glance what all is there in the website using the SiteMapPath control. These controls are also commonly used in all professional websites. We also have Login facilities at most of the sites and for getting that done, there are controls available in the form of Login control and then there is a ChangePassword control wherein whatever password we are using right now, if we think that, that needs to expire and we want to login with new password then ChangePassword control is used. There is also a ForgottenPassword control through which if somebody forgets a password then whatever mechanism wants to be done to inform him about the new password that can be managed through it. Most of the commonly required operations that are to be done on a website can indeed be handled using the different controls in the form of web controls that are made available to us.

Web Controls

Suppose we decide to have two controls on a webpage. One is the textbox in which a name can be typed and then can be a button like Add which we click. As a result of which, we are able to add this name may to a database or to a XML file or to a text file or to an ArrayList in memory. So there are two controls inserted on a webpage. The moment we insert the textbox control, an <asp> tag is inserted within the aspx page which reads like <asp:TextBox id = "fname" runat="server"/>. runat="server" indicates that control is going to get executed on the server. When we say id = "fname", this id is really gets converted into the reference to an object that gets created of the type
TextBox. There is also a tag for button which is `<asp:Button id = "ab" Text="Add" runat="server"/>`. The id given for the button is ab. There is a Text attribute which says that its value is Add because Text attribute has a value Add, on the button surface we see that the text Add is written over there. On the server side, suppose the MyPage class is created which is derived from the Page class. Inside the MyPage class, we can see that there are two references created; one by the name fname and another by the name ab and TextBox and Button which we used as the asp tags become classes on the server side. There is a class TextBox which is derived from the WebControl class and there is a Button class which is also derived from the WebControl class. For every single control that we drag and drop on the webpage, there is a parallel class available in the Framework Class Library. Most of these control classes are derived from a generic WebControl class. Each of these classes contains properties and events. For example, TextBox can have properties like Text, Width and Enabled. Text lets you determine what text is being typed out within that TextBox control and Width decides the width of the control. Enabled property depending upon it is set to true or false will decide whether the webpage should appear or should not appear on the webpage. Using the properties we can control the behavior and appearance of the control. In addition to that, there are events as well. For example, with Button there are events like Init( ), Load( ) and Click( ). Whenever the button is clicked, its Click( ) handler is going to get called but that does not mean that with a button only one event can be associated. There are multiple events that can be associated with a particular web control. Based on what action is going to happen, a suitable event will be fired. As a result of which a handler would get called. Handler would be present not only in the Control class but in the Page class that we have. In the MyPage class, there is ab.Text = "ADD" means we are trying to change the Text property of the ab control. Means properties that are there within the TextBox class can be easily accessed through the reference for that TextBox whereas if somebody clicks on the Add button then the event Click( ) would occur for which the event handler called ab.Click( ) would get called. For every single control that we put on the webpage, there would be a Control class present which would contain events and properties. Those properties we can use within the Page class to manipulate the appearance and behavior of the control whereas the event handlers can be written for corresponding events that have been declared inside that Control class. As and when those events occurred, suitable event handlers would then get called.

**Web Controls Basics**

For every single control there is a class present in Framework Class Library. Control has Properties as well as Events. Using properties we can modify the appearance and behavior of the control whereas events are there to convey the user interactions with those controls. Different properties and different events exist for different controls. Two broad categories of controls are With User Interface and Without User Interface. Some of these controls might have a user interface; some of them may not have a user interface. For example, a Timer control. Timer does not get displayed but as soon as a certain amount of time is over then an event will be fired for that control and for that event we may write an event handler within the webpage. Those who have a User Interface, for example, Button and a TextBox have a User Interface, those are typically derived from WebControl class which in turn derived from a still more generic Control class. Those controls which do not have a User Interface i.e. Timer and SqlDataSource control, they are derived from the Control class.

**Goal**

We wish to display a webpage with different controls and some flash animation at the top. In the Employee website, we would want to receive information about an employee may be that we wish to store in a database. For doing so, we will have different TextBoxes, Drop down list, Calendar control, so on. These we would use to receive information like name of the employee, address of the employee; within the address, there are two address lines and then there is a state and then the city. State and city information is important in any address. State information can be done using a drop down list box. Whenever we have a drop down list box, we will have names of different states. The
moment we pick a state appropriately the cities within that state will be listed in the list box for city. Then we have a TextBox to receive the pin, Radio buttons to decide the gender, Male/Female, Calendar control to accept the date of joining of that employee in our organization. When all this is being displayed, at the top we can see that there is a small snail is being displayed there. It’s an animated flash file. So we want to play that animated flash file where the snail moves from right to left. At the top this flash file would continue to get played. Then the name, address, gender, date of joining, etc. are displayed. In addition to this information, we would also want the user to upload his photograph. To upload the photograph, we will have a browse button wherein we can navigate through different drives and directories. Select the file in which photograph is there. The path of the image file will now get displayed within the TextBox for photo. Email address would also be there. If Email address is entered then we will have to validate that Email address. Following that we will have the hobbies of the employee stored in the database. One person can have multiple hobbies, so checkbox is used. When we click on Add Now! Button, then only the addition should happen.

**Desired Result**

Not only all the information that is either typed or selected from the different controls, all this information has to go into the database. In addition to that whatever was entered, the summary of that is displayed on the webpage and to do so, we will put a webpage showing clearly the entire information as shown in the slide. So, all the information along with the photograph, all of that should be displayed in the form of a result of the addition that we just now done. So for every employee, we would go on adding information, it should get displayed in the form of summary along with the photograph on our webpage.

**Steps Involved**

First step would be to create a new website. Once that is done, we would proceed to creation of a database. Then we would also be required to create a backend class called Employee. Then we will also be required to create a Webform using which we would try to retrieve the information or receive the information from the user and then display it in the form of a summary. For that we need to deal with a Webform. Now we would try to create a 1st step i.e. creation of a new website. In this process, we will have to give name of the project and we provide it as EmpMgmt. Then we will change the title of the webpage to Employee :: Management. Then we will add a flash file to that webpage is flyingsnail.swf. Flash files have a .swf extensions standing for shockwave file. Then the background color of the webpage is changed to cream color. These four steps are carried out as a part of Part I of building of this website.

**Step 1.1 – Create Website**

Go to the File menu. Select New Website. When we do so, we get a dialog as shown in the slide. From the dialog that appears, select ASP.NET Web Site. Then give the location for that as File System. In C:\ASP.NET the website is created. Give the name of the website as EmpMgmt. Then click on OK.

**Step 1.2 – Change Title**

Once that is done, now we would be required to change the title for the website. Go to the Default.aspx page of the website. There change the title from Untitled Page to Employee :: Management.

**Step 1.3 – Add Flash Animation**

To add the flash animation, we have to insert a `<object>` tag. Within the `<object>` tag we have information about what is the location of the swf file and that is indicated by the data attribute of the `<object>` tag. We are saying ~/flyingsnail.swf. ~/ indicates the current directory of the website, within
that there is flyingsnail.swf. That indicates the location of the shockwave file. Then we use another attribute called type which stands for MIME type i.e. Multimedia Internet Mail Extension type which basically indicates the contents of that swf file. The <object> tag can not only be used for playing swf files but it can also be used for variety of other things like displaying images, displaying photographs, playing sound, playing video. Hence what type of MIME content is there, that we have to indicate and for our swf file, the MIME type that we have to mention is application/x-shockwave-flash. It is typed as it is when you create this <object> tag. Then we have to mention the width and height of the swf file. Then we are giving the runtime initialization data which will be useful for actually playing the shockwave file. There we have to give the name which is given as movie and value as flyingsnail.swf. So parameters that are to be given to the ActiveX control which is actually going to play this shockwave file that initialization data is always given in the form of name, value pairs using the <param> tag. So we mention that within the <object> tag and then we are ready to play the swf file.

Result

The slide shows the result of the swf file. The flyingsnail.swf file is shown in the Default.aspx page in the Design format.

Step 1.4 - Change b/g Color, Font

We are also intended to change the background color and the font. For that we will go to the Default.aspx file. There in the <body> tag of the form itself, we are trying to change the color. We are marking the background color as FFFFCC. Then we are mentioning the font-family as Verdana and font-size as small font-size. This is going to work for the entire page.

Result

When we see the result, we find that the page color is changed and the flyingsnail moves from right to left. All the information about an employee is displayed on the webpage which is a cream color webpage.
EmpMgmt Website – Database Design

In this lecture you will understand:

- Goals of Database Design
- Database related terminologies
- Database design approaches
- Creating database & tables for EmpMgmt app
Goals Of Database Design

The database should be simple to understand. Don’t create too many tables and then establish complex relationships between those tables. We may be able to design a database but unless and until, it is easy to use later in a programmatic fashion then the whole idea itself get defeated. 2nd thing is we should not be required to change the design too often. If we are required to change the design too often then it shows a myopic vision on your part. It means that we have not given enough thought while designing the database in the first place. That is the reason why we are required to change it often. We should factor in the possibilities while designing the database. If we design it properly then we are not required to change it very often. So we should think carefully, give enough thought to different possibilities, factoring the newer things that may come up as the applications starts getting used by multiple users and then only design the database. Once the website goes online, there onwards we cannot really touch the database because that is a very scary thought because we are trying to deal with a live website where people are interacting with it. That time we really do not have time to change the database. So we have to think carefully when we sit down to do the database design. It should not be an after thought. While doing so, we should avoid duplication of data. Because the moment the data gets duplicated, the performance of the database getting deteriorates drastically. It will occupy more space on the disk. Lastly, it should be possible to easily update the data that is there in the database. When we fire queries like UPDATE, it should not become too complicated.

Employee Database Observations

Every employee is going to have a name, address, gender, photograph, email, hobbies. An address consists of two address lines; line1 and line2, then city, state and PIN. One state is likely to have many cities. Means for those cities, state becomes common. For example, Mumbai, Pune, Nagpur, Aurangabad, these are cities within Maharashtra state. Each city can have multiple employees. Means if there are 20 employees working in Nagpur, then against those employees we do not write Nagpur 20 times because it will be again a duplication of data. This fact we have to factor in when we design the database for Employee Management website. So cities as well as states should not get duplicated. One employee is likely to have multiple hobbies.

Simplistic Database Design

One of the possible and simplistic database design would be as shown in the slide. That we have Name, Address1, Address2, City, State and then the different hobbies, Cricket, Reading, Traveling, Singing and Swimming. Against those hobbies we can put True/False based on which particular checkbox is checked on the webpage. Accordingly we will mark those hobbies as True or False. If the checkbox is checked then we will mark it as True, if it is not checked then we will mark it as False. This is a workable table. But it is inundated with problems. In this the major problem is data duplication. In the city column, there is Nagpur is repeated twice, Mumbai is repeated twice, Hyderabad is also being repeated twice. So if the employee is belonging to same city then why do you want to store the complete string Nagpur. Instead we can give a code for Nagpur lets say 1 and it is going to occupy less space as compared to string Nagpur. So we will have to avoid this data duplication. In the State column also, there is a lot of data duplication. The way we decided to give code for cities, we can also give code for states and put the codes in place of the strings. Somewhere else we store the information that 1 means Maharashtra and 2 means Andhra Pradesh. Another problem with this database is that there is no mechanism in this database to identify a row in a unique fashion because even though the possibility is less but we cannot rule out the possibility that name of the person, address, city, state as well as hobbies would be same. It might be. If that is so, if two employees have exactly same data then how do we identify one differently from the other. In this database design which is shown in the form of a table as shown in the slide, it’s not feasible. So this also needs to be accounted for that if we have any information about employees then there must be
some mechanism to identify each employee in a unique fashion which will have no conflict with any other employee even though the data of that employee is same as the first one. So the uniqueness aspect has been accounted for while doing the database design. Thirdly this is an inflexible database that we have created because today there are only 5 hobbies. But tomorrow if we decide to add two more categories of this then it will create a problem. Because we agreed that we will not change the database once it is designed. Hence even though this is a workable way, this cannot be the best way. We cannot change the columns later on. We have to do it in some other fashion.

Solution

Solution to the problem I is we can create different tables for cities, so also for states. In cities table we can have list of cities along with the unique identifier for each. In the States table, we can have names of different states along with unique identifiers to identify each state in a unique way. So we will have to take out this data from the table where we have employee information and create separate tables one for City and another for States. The problem of inability to identify a row in a unique fashion can be solved by creating identity columns within our tables. So we now finalized at least three tables. One is Employees, 2nd is Cities and 3rd is States table. For each one of these tables, we will have some or the other way to identify each row in a unique way and to do that, we will establish an identity column within each of these tables. The inflexibility can be sorted out by creating a separate table which maintains a list of possible EmployeeHobbies. So if there are 5 rows in EmployeeHobbies table; one row can contain Cricket, another can contain Swimming, so on. If there is a 6th hobby, we can add one more new row because that’s the data. We are not talking about addition of a new column because when we add a new column, we are modifying the database or we are changing the database design. But when we are adding a new row, we are adding a new record to the existing database without changing any of its design. So, all these three problems can be sorted by creating multiple tables. We have at least four tables here; Employee, City, States and EmployeeHobbies. Once we create these tables, they should not remain there in isolation. Whatever natural relationships that exist between these tables, that relationship we need to establish. Because once it is established then only we will find that the true benefits of a good database design will keep accruing to us.

Terminology

Primary Key – A column whose value is guaranteed to be unique in each row of a table.

Foreign Key – A column whose values are primary keys in other table.

Identity Column – A numeric column whose value is automatically assigned by SQL server and incremented after every insertion.

Composite Primary Key – A combination of columns which together gives a unique value in each row of a table.

Steps Involved

First step that we would do is creation of the database. Part I we had done last time. Creation of a database involves creation of the database means the mdf file and subsequent to that creating different tables inside that mdf file. Within one mdf file, we can have multiple tables. In ContactsDB.mdf we had created a Contacts table. Similarly here we will also create a mdf file in which in place of one table, we will have multiple tables.

Step 2.1 – Create Database

To create the database, go to EmpMgmt website. In the Solution Explorer, right click on App_Data folder. When we do so, a menu would pop up. Select Add New Item from the menu that pops up.
When we do so, a dialog would appear as shown in the slide. Select SQL Server Database from the
dialog that appear and give the name as EmpDB.mdf. Then click on Add.

**Step 2.2 – Create Tables**

Once the mdf file is created, we need to add tables in it. To do that, go to the Database Explorer. Here
right click on the Tables.

**Save Table**

Once we do that, a menu will pop up from where we can Add New Table. Once we select Add New
Table, we will be allowed to give names of the columns, the data types and either check or uncheck
the Allow Nulls column. So for Employees table, we are going to have several columns; EmpId
which is bigint, EmpName which is nvarchar(100), AddLine1 which is nvarchar(100), AddLine2
again nvarchar(100), then CityId is smallint, Pin is nchar(6), Gender is bit, PhotoUrl is nvarchar(100),
Email is again nvarchar(100). All these columns we should create within the database. We will
uncheck every Allow Nulls because we do not allow to have null values. Then click on the Save
button. As soon as we do so, a dialog will pop up where we will be asked to enter the name for the
table. Give the table name as Employees. So EmpDB.mdf is the database file. Inside that we have
created our first table i.e. Employees. Then click on OK. So Employees table now stands created.

**Result**

We can watch this Employees table against the Tables entry in the Database Explorer. The slide
shows that within EmpDB.mdf, there is Employees table now stands added.

**Create More Tables**

We can add other tables also. We propose to create States table wherein there would be StateId and
StateName. Stateld is a tinyint whereas StateName is nvarchar(50). Once again Allow Nulls are
unchecked. We can also create a Cities table. In the Cities table, we are going to have CityId which is
smallint, Stateld which is tinyint and CityName which is nvarchar(100). Here Stateld becomes a
foreign key whereas CityId would be the primary key. Again we will uncheck the Allow Nulls. Then
we will create a Hobbies table. In the Hobbies table we have HobbyId which is tinyint and
HobbyName which is nvarchar(100). Then we will create an EmployeeHobbies table which contains
EmployeeId which is bigint and HobbyId which is tinyint.
EmpMgmt – Database Relationships

In this lecture you will understand:

- What are PKs and FKs
- How to set PK
- How to set Identity column
- How to establish relationships between tables
Steps Involved

We had divided the entire project in four parts as shown in the slide. Pot of four parts, first part is complete. 2nd part is partly complete. While creating a database, we had given the database name as EmpDB.mdf. Then we had created five different tables in the EmpDB.mdf. Those five tables are Employees, Cities, States, Hobbies and EmployeeHobbies. Hobbies table is used to keep list of all valid hobbies. EmployeeHobbies is going to have which employee has which of the hobbies that are there in the Hobbies table. We will also need to establish this relationship clearly. Out of creation of database, first two parts are completed i.e. creation of database and creation of tables. The other two parts are setting the primary key and the identity column in each table and then represent the relationships that exist between these tables.

Step 2.3 – Primary Key, Identity Col.

We will begin with the Employees table. Go to the Database Explorer. Locate Employees table there. The moment we click on Employees table, the design of the Employees table will be shown in the left hand side pane. There go to the EmpId which we want to mark it as a primary key. Because Id of an employee is going to be unique. So we mark that as the primary key. To do so, right click on EmpId in the left hand side pane. When we do so, a menu would pop up. From this menu, select Set Primary Key as shown in the slide.

Result

Once we set EmpId as primary key, we would find that in the table, now a key would be marked in yellow color. The key indicates that EmpId has now been marked as a primary key within this table.

Set IDENTITY Column

To mark the identity column within this table, first locate the Identity Column in the Properties of the Employees table. Employees table can be found from the Database Explorer. Its properties are shown in the pane below that. Locate the Identity Column there. When we do so, click on the triangle on the right hand side. Once we have done that, EmpId and CityId is shown to us. Click on the EmpId because that is the one which we want to mark it as an Identity Column.

Result

Once we do that, the result is shown in the slide. We can see in the Properties window, against Identity Column an EmpId is shown indicating that that is being accepted as the Identity Column. Then we save whatever we have done i.e. setting of the primary key and setting of the Identity column for Employees table. The EmpId column is going to act as the primary key column and it is also going to act as the identity column. Means the values that are there inside this column, those are going to be unique. By setting EmpId as Identity Column, we are ensuring that entries within this Identity Column will not be made by us. They will be made by the SQL database server. As and when we add a new row to the Employees table, automatically the EmpId for that row will get incremented. We can never touch the Identity Column. Then only database server can guarantee the uniqueness in that column. So the primary key column itself can act as the Identity Column. With what unique number an Identity Column should begin and by what increment should it keep on changing every time we add a new row, we can control that. That’s known as initial value and the step value. We can set initial value to 5 and step value to 10. If we do so then the 1st record that we add in the Identity Column 5 gets entered. After 5, next record is marked as 15, so on. CityId here is the foreign key because CityId’s value will be unique in the Cities table.

States Table: PK & Identity
In States table, we have StateId and StateName. For that go to the Database Explorer, locate the States table selected. The moment we do so, in the left pane all the columns along with the Data Type, Column Name and Allow Nulls will be shown about the States table. From here, we can mark StateId as the primary key. Then also set its Identity Column to StateId. Once again the StateId column is acting as the primary key column and the Identity Column. StateId acts as an Identity Column means any time we add a new row automatically the Id of the State will get incremented.

**Cities Table: PK & Identity**

In the Cities table also, we need to mark the primary key and the Identity Column. Locate the Cities table in the Database Explorer. In this Cities table, we will mark CityId as the primary key. In the Properties window, mark CityId as the Identity Column. Once again the CityId is going to act as the primary key as well as the Identity Column. Here StateId is going to be the foreign key because in this table typically we would have Ids for the cities, names for the cities. Multiple cities can belong to the same state. So StateId when we marked as Maharashtra for Pune, Mumbai, Nagpur then lets say StateId would be 1 for Maharashtra. All these 1 would be same for all the cities. But even though its 1 here that 1 is unique in the States table. Hence StateId become foreign key for this table. We have not marked any key as foreign key. When we establish relationship, it will be established. But now we are marking the primary key and the Identity Column.

**Hobbies Table: PK & Identity**

In the Hobbies table also, we will do exactly the same thing. From the Database Explorer, select the Hobbies table. Its content will be shown in the left hand side pane as shown in the slide. Mark HobbyId as the primary key. Go to the Properties window. There again mark HobbyId as the Identity Column. Again HobbyId is acting as the primary key and the Identity Column.

**EmployeeHobbies: PK & Identity**

Go to the EmployeeHobbies table. From the Database Explorer, select the EmployeeHobbies table. Its content will be shown in the left hand side pane as shown in the slide. In this table we have EmployeeId and a HobbyId. In the rows, we are going to have EmployeeId, against which the Id of the Hobby that the Employee has is going to be marked. Suppose EmployeeId is 20 and this employee with Id 20 has hobbies like Cricket, Swimming and Reading. If Cricket is having Id 1, Swimming as 2 and Reading as 3 then in the EmployeeHobbies table, there would be three records for this employee. When we have these three records, since Cricket, Swimming and Reading are going to be hobbies of Employee number 20, the column EmployeeId will have 20 being repeated 3 times. So in the 1st row, 20 is marked with hobby number 1, in the next row 20 is marked with 2 and in the 3rd row 20 is marked with 3. Means 20 is being repeated thrice. Means even though the Employee Id in the Employees table was a primary key, here it cannot be a primary key. HobbyId cannot also be a primary key because the way the employee number 20 has Cricket as the hobby, employee number 21 can also have Cricket as the hobby. So in the HobbyId, there would be 1 being repeated 1ce of employee number 20 and 1ce for employee number 21. Neither EmployeeId can be a primary key nor HobbyId can be a primary key. But the combination of EmployeeId and HobbyId is indeed going to be unique. Means 20 as the EmployeeId and 1 as the HobbyId for Cricket that is a unique combination i.e. 20 and 1 together. 20 and 1 is never going to be repeated in the list. Means we are saying about the composite primary key which is built out of a combination of more than one column. In this case more than one columns are EmployeeId and HobbyId. We will have to select both of them, right click on it, a menu would pop up and then select Set Primary Key from the menu that pops up. As shown in the slide, both the columns are marked with yellow color key indicating that both together form the composite primary key. EmployeeHobbies has multiple records standing for one employee record. Whenever we have multi-valued scenario, we would usually have composite primary key and here we did not have an Identity Column at all.
**Step 2.4 – Relation : Employees, Cities**

To establish the relation between Employees table and Cities table, go to the Employees table. Right click on that table. When we do so, a menu would pop up. From the menu that pops up, select Relationships. As soon as we do so, we will get a Foreign Key Relationships dialog as shown in the slide. Means we are trying to establish the relationships between Employee table on one hand and some other table on the other hand which in this case is Cities table. But to tell it that the other table is Cities table, we need to click on the Add button.

**Foreign Key Relationship**

As soon as we click on the Add button, another dialog will come wherein we are required to indicate the Tables And Columns Specifications. Means we are trying to establish the primary key, foreign key relationship. In the process of doing so, we are required to mention the Specification property of Tables And Columns. Within this dialog, locate Tables And Columns Specification. Go to the right hand side of that row, a rectangular button with three dots i.e. … is present. Click on that button.

**Select Primary Key Table**

As soon as we do that, another dialog will appear as shown in the slide, where we are required to mark the Tables and the associated columns to establish relationships between different tables. Here right now Primary key table and Foreign key table, both are being shown as Employees. But we do not want this. So we will go to the 1st drop down listbox, click on the triangle there.

**Select Cities Table**

As soon as we do so, the other tables will also shown to us. From the list, select the Cities table because we want to establish relationship between Employees and Cities. We began with Employees table. We are trying to indicate the Primary key table as Employees because CityId is common between Employees table and the Cities table. So CityId is primary key in Cities table. So we choose Cities as the Primary table.

**Select Primary Key**

Once we have done that, we can see that against Primary key table, earlier there was Employees and now Cities come there. Within the Cities table, we have to indicate which column is the column with the Primary key value. For that, click on the triangle in the drop down listbox.

**Select CityId**

Once we do that, the column names of the Cities table will be shown. We can see CityId, CityName and StateId in the slide. From here, we will select CityId as the primary key for the Primary key table.

**Select Foreign Key**

Then we will go to the Foreign key table which is Employees. Click on the 2nd drop down list box. From here, again we select CityId. So CityId is the primary key in Cities table and its being marked as the Foreign key in the Employees table.

**Relationship Name**

The name given to this relationship is FK_Employees_Cities. Employees is the foreign key table and Cities is the primary key table and both tables have common Id as CityId. Then click on OK.

**Close**
Lastly we close this by clicking on the Close button. We can save this relationship by clicking on the Save button.

**Relationship: Cities, States**

Now we try to establish relationship between Cities and States. Go to the Cities table. Right click on it and select Relationships from the menu that pops up. As soon as we do so, a dialog will appear and then click on Add button to add a new relationship.

**Select PK & FK Tables**

As soon as we do so, a dialog will appear where we have to mark the Primary key table and the Foreign key table. We will follow the same procedure as we had done while establishing the relationship between Employees and Cities table. We will select States as the Primary key table and Cities as the Foreign key table. Against both the tables we will select Stateld i.e. Stateld is primary key in the States table but it’s a Foreign key in the Cities table. This relationship is called as FK_Cities_States. Then we will click on OK button.

**Close**

Finally we close this by clicking on the Close button.

**Relationship: EmployeeHobbies, Employees**

The third relationship that we need to establish is between EmployeeHobbies table and the Employees table. We will begin with the EmployeeHobbies table. When we click on Add button, we will get the dialog as shown in the slide. We will set Employees as the Primary key table and EmployeeHobbies as the Foreign key table. We also select EmpId as the primary key from the Employees table which is a Primary key table. But do not try to establish the relationship between EmpId and a combination of EmployeeId and HobbyId of the EmployeeHobbies table. Hence make sure that HobbyId is related with the EmpId value of the EmployeeHobbies table. Then click on OK.

**Relationship: EmployeeHobbies, Hobbies**

Now we will establish relationship between EmployeeHobbies table and Hobbies table. Go to the EmployeeHobbies table. Then right click on the table. We will get a menu. Select Relationships from the menu that pops up. Then we click on Add button to add a new relationship. Then we will get a dialog. Here Primary key table is the Hobbies table whereas EmployeeHobbies table is the Foreign key table. We will select HobbyId as the primary key from the Primary key table i.e. Hobbies. But do not try to establish the relationship between HobbyId and a combination of EmployeeId and HobbyId of the EmployeeHobbies table. Hence make sure that HobbyId is related with the EmpId value of the EmployeeHobbies table. Then click on OK.

**Save Changes**
All the changes that we have made, we have to save them by clicking on the Save button. It will ask you that the changes have been made, do you really wish to continue to save them. Click on Yes button. Now all the relationships that we have made get saved in the database.

**Relationship Summary**

The relationships are shown in the slide. We have five tables; Employees, Cities, States, EmployeeHobbies and Hobbies. Each table has a primary key. EmployeeHobbies has a composite primary key. CityId is foreign key in Employees table and StateId is foreign key in Cities table. As shown in the figure, between Cities table and Employees table, there is a linkage. At one end of the linkage there is yellow colored key. At the other end there is infinity sign (∞) is present. The moment we have a ∞ sign, it means that there is 1 to many relationship between Cities and Employees. Means in the CityId, there is going to be a unique key. That unique key may have several occurrences in the Employees table. If there are 20 employees belonging to the same city then CityId would be occurring at multiple rows in the Employees table whereas that Id will occur only once in the Cities table. That’s why one city occurring against multiple employees which is 1 to many relationship. So ∞ and yellow colored primary key clearly indicates 1 to many relationship. In the linkage between Cities and States table, at one end there is ∞ symbol and at other end there is a yellow colored key which is a primary key symbol. This also indicates 1 to many relationship. There is again 1 to many relationship between Hobbies and EmployeeHobbies table. There is also 1 to many relationship between Employees table and EmployeeHobbies table. EmpId and HobbyId are foreign keys in EmployeeHobbies table. EmployeeHobbies has no Identity column. So the relationship that we establish can be seen in the database diagram. There is a provision to create this database diagram through the database server.
EmpMgmt – Database Diagrams And Data Entry

In this lecture you will understand:

- How to create Database Diagram
- How to enter initial data in tables
- How to create backend class - Employee
Database Diagram

Go to the Database Explorer pane. Right click on the Database Diagram. A menu will pop up. Select Add New Diagram from the menu that pops up. Then it will ask that do you want to create a new diagram? We say Yes. So we click on Yes.

Add Table

As soon as we do so, a dialog as shown in the slide is displayed. What tables do we want to display in this database diagram is asked to you. The dialog shows five tables; Cities, EmployeeHobbies, Employees, Hobbies and States. We want to select table from here. Suppose we select Cities table. Then click on the Add button.

Add All Tables

When we do so then in the diagram the Cities table would appear along with all its three columns namely CityId, StateId and CityName. So Cities table has been added. Exactly on similar lines we can add other tables i.e. right click, Add New Diagram. From the table’s list, select EmployeeHobbies table and click on Add. So in this way one by one all tables get added.

Result

Once we do that, in the Database Diagram, all the five tables will be shown along with the relationship that we established between them as shown in the slide. The database diagram gives the visual appearance of how we want the data to be presented.

Data Entry - States

Now we need to enter some initial data into some of the tables of the database. We want to do it for the States table. For that, go to the Database Explorer. Select EmpDB.mdf from there. Against the Tables category, all the five tables will be shown. Select the States table by right clicking on it. When we do so, a menu will pop up. Select Show Table Data from the menu that pops up.

Type State names

When we do so, an empty table as shown in the slide will be shown because nothing has been added to the table so far. Table shows two columns; StateId and StateName, both of them having NULL values. We will go and try to add values there. We will type states name here since we cannot add StateId because it’s an Identity Column. We never add values to the Identity Column. Database server adds values to the Identity Column the moment we add a new row. So we are required to straightway type out the names of the states against the StateName column. Suppose we add first state in the StateName column and hit the Enter key, no sooner do we hit the Enter key, the StateId will be automatically entered.

Result

So having entered few states, the States table data would look like as shown in the slide. The slide shows the difference in color in the values in StateId column and the values in the StateName column. StateId column values are shown in gray color whereas StateName column values are shown in black color. StateId column values are gray in color means they have not been typed by us; they are auto generated. This is the purpose of Identity Column.

Data Entry - Cities
Again go to the Database explorer. Select EmpDB.mdf. Expand the Tables branch from there. Then right click on the Cities table. As soon as we do so, a menu would pop up. Select Show Table Data from the menu that pops up.

**Type Data**

When we do so, an empty table as shown in the slide will be shown because nothing has been added to the table right now. We are required to enter data in the table. In the Cities table, CityId, StateId and CityName columns are present. Again CityId is gray in color whereas StateId and CityName are black in color. Means in StateId and CityName, we can type values whereas values in the CityId will be auto generated by the database server, the moment we enter a new row in the table.

**Result**

After typing certain values for StateId and CityName in the Cities table, the result is shown in the slide. While entering these values, we have to keep States table data handy to indicate for which city the State belong. CityId values are unique which get incremented when we went on adding new row to the table. StateId is not unique but it is unique in the States table.

**Steps Involved**

The four parts of this application were creation of a new website, creation of a database, creation of a backend class and creation of a webpage. First two steps are now stand done completely. Now we will try to create a backend class where name of the class is Employee.

**Employee class – App_Code/Employee.cs**

To add this class, we will go to the App_Code folder in the Solution Explorer. Right click and say Add New Item. Select Class from the dialog and give the name as Employee.cs. The purpose of the Employee class is for every row in the Employee table, there is going to be one Employee object. At the beginning, we import using System, using System.Collections, using System.Data and using System.Data.SqlClient. Then we have the Employee class. Inside the Employee class, we will first create the connection string called EmpConStr and it is initialized to @"Data Source=\SQLEXPRESS ;" + @"AttachDbFilename=| DataDirectory|\EmpDB.mdf ;" + "Integrated Security=True ; User Instance=True". Then we would declare a private variable long mEmpId. To retrieve the value present in mEmpId, we have a property called EmpId. This property has get clause which will return the mEmpId value. EmpId is a readonly property. We can only get values from it but we cannot set the value in it.

**Slide 15**

On similar lines, we will declare a variable private string mName. Then we will declare a Name property having get clause which returns the value of mName and a set clause which sets the value in mName. Then we have private string mAddLine1 variable and a property called AddLine1. This property has a get clause which will return the value of mAddLine1 and a set clause which sets the value in mAddLine1. On similar lines we have private string mAddLine2 and a property called AddLine2. This property again has get clause to return the value of mAddLine2 and a set clause to set the value in mAddLine2. Then we have private string mState variable. We have a property called State and within the property we have get and set clauses which returns the value present in mState and sets the value in mState respectively.

**Slide 16**

Then we create a variable private string mCity. Then we create a property City having get and set clauses. get clause returns the value present in mCity and set clause sets the value in mCity variable. Then we create a variable private string mPin. A property Pin with get and set clauses are created. get
clause returns the value present in mPin and set clause sets the value in mPin variable. Again we create a variable private bool mGender. A property Gender with get and set clauses are created. get clause is used to return the value present in mGender variable whereas set clause is used to set the value in mGender variable. Then we will create a GenderText property which is a readonly property. This property will check the value present in mGender. In this property we are saying return Gender. The moment we say Gender, we are going to have access to the get clause of the Gender property which will return mGender value. If the mGender value is true then this GenderText property should return Male, otherwise it should return Female.

Slide 17

Now we will create a variable private DateTime mDateOfJoining. Then we will create a property DateOfJoining having get and set clause. get clause returns the value present in mDateOfJoining whereas in set clause, we set the value in mDateOfJoining. Then we will create a variable private string mPhotoUrl and a property called PhotoUrl. This property has get and set clauses. get clause returns the value present in mPhotoUrl variable whereas set clause sets the value in mPhotoUrl variable.

Slide 18

Now we create a variable private string mEmail and a property called Email with get and set clauses. get clause returns the value in mEmail variable and set clause sets the value in mEmail variable. Then we will create a variable private ArrayList mHobbies. We need ArrayList because there is a list of hobbies. We have a property called Hobbies which has a get clause. Means this is a readonly property. get clause returns mHobbies. Then we have a constructor of the Employee class. Inside the constructor, we will create a new ArrayList and set its reference in mHobbies.
EmpMgmt - WebForm

In this lecture you will understand:
- Creation of webform
- Insertion of different controls
- Data-binding of controls
- Resultant aspx tags
Steps Involved

In this example, the steps involved are creation of a new website, creation of a database, creation of a backend class and creation a webform. In this lecture we will study how to create a webform.

**Step 4.1: Create Table + Row1**

Firstly we are going to create a table as shown in the slide. We had created a table manually just below the object that we inserted in the form of a flash animation, the flyingsnail. So we created `<table>` tag with `id = "tblReg"`. Within that table, we had created three columns. Column1 is for the heading of that particular row. In this case Column1 is `Name*` where * indicates that it’s a compulsory field. In the 2nd column we are going to have textbox in which user is going to type name of the employee which is to be inserted in the database. Relative width of each of these columns is managed using the `Width` attribute for `<td>` tag in HTML. So within `<tr>`, we had created three `<td>` tags to indicate that there are three columns. Name is aligned to the right of its cell. Middle column has unspecified alignment. Third column has a validator control which validates whether name has been entered or not because when we make certain field compulsory, we also want to make sure that user does enter something before that record is treated as a valid record before insertion in the database. So for that we are using validation control. We are managing the relative geometry such that 100% of the width of the webform is utilized in such a manner that left column occupies 20% width, middle column occupies 40 % and right column occupies remaining 40% of the width. This sort of relative geometry ensures that nothing is hard coded. Depending on the width of the browser appropriate space gets allotted to these controls on the form of user’s browser. If we use absolute width in terms of points or pixels, it becomes very difficult to manage the geometry. On the left hand side, in the Toolbox, we have highlighted set of controls which together are known as Validation controls. The purpose of this control is to validate the text that is entered in some other controls on the form. On their own, they have no functionality. They are there to validate the data entered on the form. So in the 3rd column, we have entered validation control.

**Step 4.2: AddLine1, AddLine2**

Now we will put AddLine1, AddLine2 and the textbox corresponding to them. Since these two rows are identical, we have considered them together. The corresponding asp tags would be `<asp:TextBox ID="txtAddress1" runat="server" Width="100%" MaxLength="100" />`. MaxLength="100" means in the address line1 maximum length permissible is 100 characters. Address line1 is compulsory, address line2 is not. So we have also placed another validator control in the row belonging to address line1 which is in the 3rd column. The `RequiredFieldValidator` has a property called `ControlToValidate`. In that we have to specify the Id of control which is supposed to be validated. There we entered `txtAddress1`. In case the textbox containing address line1 is left empty, the ErrorMessage property says what error is to be displayed. So in this case the error message is Address Line1 is required. In address line2, if we do not type anything, it is completely acceptable because it’s an optional field.

**Step 4.3: States DropDownlist**

In most of the countries address begins with State for which state the employee belongs to. We have created States table. We have also entered information about States in India in this table. So we have dragged and dropped a DropDownlist control from the Toolbox in the 2nd column in the next row. The column heading is States. This is a DropDownlist which allows one of the items from the list is to be selected. Information about states is in the database. So, to ensure that we always use the latest information and not the information which is hard coded, for that we have to perform data binding. Data binding mechanism is very popular in the world of ASP.NET. DropDownlist happens to be one of the controls where this mechanism is available. First we select the DropDownlist control. A small
triangle appears next to that. This is called as the smart tag. If we click on that with mouse button, it shows the tasks or the kind of things that we can do with this data bound control. Right now it says that it is unbound which means that the data is not there. We are trying to bind it. So among the options, it says there is option to PostBack automatically. We click this button to ensure that every time the selection of state changes, a POST message should be generated and send to the server. The list of cities that we are going to see down below after this row, is going to be dependent on the state which is selected. So every time the state that user had selected changes, the box that appears below shows the cities also has to be refreshed. That means we have to let the server know that the user has changed the selection. For that the AutoPostBack property has to be turned on. Then we have to say from where exactly the data is going to come from. So we click on Choose Data Source.

Create Data Source

Once we click on Choose Data Source, it asks for selection of the Data Source among the Data Sources available on this page. It only shows the sources of data available on the page. To see what data sources are available, we click on the drop down box as shown in the slide. Initially there was none. Then we click on the triangle to select the Data Source.

New Data Source

As soon as we click on the triangle in the drop down list, we get a New Data Source. New Data Source indicates that there is no existing data source present on this page. So we select New Data Source.

Choose Data Source Type

As soon as we click on the New Data Source, we get a dialog as shown in the slide. What kind of sources we can use to retrieve the data about states? We can use SQL database, Access database, link object, regular object created using classes in C# and we also use XML file and sitemap types of sources to retrieve the data about states. So many choices are available. In this case we use a SQL database. We have already created a SQL database in the App_Data folder. We have to establish linkage between this control and that database. Once we select this, we have to also select the name for this data source. We are trying to create a DataSource control and from that we are going to extract the data and use that data in the DropDownList control. For that we have to select the Id for this DataSource control. Make sure that every time we select this Id, it should be meaningful.

Specify Data Source Id

We have selected sqlSourceStates as the Id for this data source. Then we click on OK.

Choose Data Connection

Now we have to tell the wizard that how do we connect to the database. How do we connect to the database? Our database server is present locally. The file which contains the data is present in App_Data folder. We can create a new connection to the remote database server if our database server is not running locally. But otherwise, we have click on the drop down link as shown in the slide.

Data Connection – EmpDB.mdf

It shows that there is EmpDB.mdf file is present in App_Data folder. Select EmpDB.mdf file. Clicking on that will give us the connection string.

Connection String
What is the connection string? If we want to see the connection string then we have to expand the + link.

**Connection String Details**

As soon as we expand the + link, in the window below that connection string is displayed. That connection is exactly identical to the kind of connection string that we have used in case of contacts book application. In this case the wizard has generated the connection string automatically. We can also create it manually. To move ahead, we click on the Next button.

**Save CS in Web.config**

In this step, it is going to ask do we want to save the connection string somewhere. Whenever you walk through these wizards, make a habit to read the title in the wizard. In this case the title is Save the Connection String to the Application Configuration File? Description says that we are going to try and save to Application Configuration File. Such a file exists in the Application’s own directory and the filename is Web.config. If it does not exist, at this time it will be created. Why are we saving this connection string in a file? This connection string is not going to be used in just one page; it is going to be used in multiple pages, multiple forms in the same web application. Every time we do not want to repeat the process of defining the connection string. This web application configuration file stores it so that every page can use it. Suppose we had deployed our application to the actual hosted web server, there the connection string is totally different. Since we are not hard coded the connection string inside the aspx file, if we change the web.config file with the correct connection string directly, the new connection string is automatically used in all the places where it is referred. We have to give some name to this connection string. So in this case we are going to call it EmpConStr. So connection string stands defined. Then we click on Next.

**Configure Select**

When we click on Next, the wizard is going to connect to the database server, get the list of all the tables, all the columns, types of all the columns in the table and so on. It might take some time depending upon how many tables exist in the database. As shown in the slide, by default the Cities table is highlighted. It shows that we are trying to configure a Select statement. Select will give the list of anything that we want. But in this case we want list of States. For that we click the drop down link.

**Choose States Table**

As soon as we click the drop down link, it shows the list of tables. Select States table from the list.

**Choose Fields**

Once we select the States table, in the textbox below that, it shows those columns which are in the States tables. So it shows Stateld and StateName columns. If we want to select all the columns, we can click *. If we do not, we can select one by one column by clicking corresponding boxes. These checkboxes allow to select only certain columns if we are not interested in all the columns. We click on * because we want to select all the columns.

**Select Statement**

As soon as we select *, the Next button becomes highlighted because the SELECT query in the SELECT box clearly indicates that the SELECT expression is SELECT * FROM [States] which means it is going to return all the states in the table. Then we click on Next.

**Test Query**
When we click on Next, it allows to test the query that we have configured. Sometimes we write the SELECT query manually. In the manual process, if there is a mistake, we may not get exactly what we are looking for. So the test step allows to check if our expected results are what exactly we are going to see. So we click on the Test Query button.

**Result**

As soon as we click on the Test Query button, it shows the list of 5 states. These are the states that we had entered manually in the step where Data Entry is to be made. Then we click on Finish. This completes configuration of data source.

**Configure Data Source**

When we come back, we will see that, in the Data Sources box the name of sqlSourceStates would be found. In addition down below that, the text to display in the DropDownList that would be the first choice that we have to make. So we click on the triangle from the DropDownList.

**Select Display Field & Value Field**

Select StateName from the list. The value that will be selected in case that state is to get selected will also be displayed. Two things are required here because as far as end user is concerned, the state like Maharashtra, Madhya Pradesh, Chhattisgarh, these states indicate what exactly is the purpose. But as far as our application is concerned, the primary key StateId, the Identity Column StateId is the one which we are interested in. In case the user makes the selection, based on that we can easily find the name of the state. So many times in such data driven controls, what get displayed to the user and what value gets stored in a field in that control is different because it is more convenient to deal with numeric Identity Columns. Then click on OK.

**Result**

The result is as shown in the slide. It shows that now the source has been selected. In that same row, we will see a DropDownList control. The word Databound comes instead of Unbound. In the task, there is Configure Data Source, Choose Data Source, more options are present. Down below there is SqlDataSource control and its id is sqlSourceStates. SqlDataSource is also a server control but its purpose is to feed the data to a DropDownList control. So depending upon what kind of purpose a control has, it may or may not have a user interface.

**Effect In aspx – DropDownList**

For DropDownList, the tag in aspx file is `<asp:DropDownList ID = "lstState" runat = "server" Width="100%" AutoPostBack = "True" DataSourceID = "sqlSourceStates" DataTextField = "StateName" DataValueField = "StateId" />`. runat = "server" indicates that it's a server control. Width="100%" says that within this column, whatever is the maximum width, we have to occupy it. Since the data for this control is going to come from somewhere else, we have to also specify the DataSourceID that is the ID of the DataSource control which is going to feed the data to the DropDownList and that’s a sqlDataSource control. Once the data become available, what value gets displayed is indicated with the help of a property called DataTextField and that value is of StateName column. What value gets selected is that of StateId. This style of programming is called as declarative programming. In the Display field the text is visible and in the selected the value of that corresponding state is visible. Suppose Gujarat was selected. If Gujarat is selected, in the SelectedValue property of this control, we are going to get a number 3. If we want to access the SelectedValue programmatically or Selected text programmatically, the selected text can be accessed using lstState.SelectedItem property in the code behind. To get the index of corresponding item in the DropDownList we can use lstState.SelectedIndex property and to get the number 3 corresponding to
Gujarat, we have to use lstState.SelectedValue property. SelectedItem and SelectedValue returns string. We have to convert it to correct data type.

**Effect In aspx – SqlDataSource**

The tag for SqlDataSource control is `<asp:SqlDataSource ID = "sqlSourceStates" runat = "server" ConnectionString = "<%$ ConnectionStrings:EmpConStr %>" SelectCommand = "SELECT * FROM [States]" />`. The ConnectionString property indicates, what connection string we are using. So there is special tag `<%$` which indicates that we are referring to a section in the web.config file. Name of that section in the web.config file is ConnectionStrings and one of the values in that section is EmpConStr. The SelectCommand is a field or property which indicates the text that is used to send a SELECT request to the database server using this ConnectionString. The return value of that is fed to the lstState control. Other properties can also be used with SqlDataSource like Insert command, Delete command and Update command. We are not hard coded the connection string here even in aspx. If we change the connection string directly in web.config, the new connection string would be automatically used everywhere, where we are referring it using EmpConStr name. So no hard coding is required. The syntax for SQL or logic for connecting to the database server in C# has been eliminated. Because we are not firing the queries. We are configuring the SELECT statements but we have done it declaratively. We do not actually generate the query. We do not read, we do not populate the data structures the way we had done in contacts book using SQL. This is the completely new way of doing things. We learnt that because ultimately under the hood exactly same kinds of things are happening. Same things happen except that instead of we do the repetitive kind of things, Microsoft has created controls which allow to prevent these sort of repetition. Insert, Delete and Update operations can also be performed declaratively by specifying the values of corresponding properties in the SqlDataSource control.

**Step 4.4: Cities ListBox**

Once the State information is collected from the user, we also need to know which city he belongs to. The cities are to be displayed in a ListBox control. List of cities is dependent on which State the user had selected. If the selection of state changes, corresponding list of city names displayed in the ListBox also change. So we have enabled the AutoPostBack property of the DropDownList. We have to repeat the data binding process for tables of cities. Select Choose Data Source.

**Choose Data Source**

As soon as we do, a dialog will appear and now we want to choose the data source. There is already sqlSourceStates present. We will select New Data Source from the list.

**Specify Data Source Id**

Now we select the type of data source as Database and name of the data source is sqlSourceCities and then click on OK.

**Choose Data Connection**

On we have to specify the connection string. Since we saved the connection string earlier, EmpConStr appears as the default choice. Then we click on Next.

**Select Fields**

Now we come to the selection of tables and creating a SELECT query. Cities table is already selected. We want to select all the columns from the Cities table. So we select *.*. But we do not want all the cities. We want cities in the state which has been selected using the State’s DropDownList. For that we have to click on the WHERE clause which allows to add a portion to the SELECT statement. So we click on WHERE clause.
Add WHERE Clause

When we configure the WHERE clause, first we have to select which Column we are going to select by. So we click on the drop down list as shown in the slide.

Choose Column

As soon as we click on the triangle, three column names would appear in the list. We have to select by StateId column. So we select StateId column.

Choose Operator

We have to use the type of Source here with a = operator as Control on the same page. So we first select = operator. Then we click on the drop down list for the selection of source.

Choose Source

From the list we choose Control as the source.

Choose Parameter Properties

Now we want to select the Control ID. So we click on the drop down list.

Choose Control Id

From the list, we select lstState.

Sql Expression

In lstState whichever StateId gets selected that becomes the input parameter for us, that becomes the parameter in the WHERE clause. So the SQL Expression is [StateId] = @StateId and Value is lstState.SelectedValue. Then click on Add.

Add WHERE Clause

Together if we combine this then our parameter becomes [StateId] = @StateId where StateId is taken from lstState.SelectedValue. So that gives us the lstState.SelectedValue as an input parameter. Then click on OK.

Resultant SQL Statement

If we come back, we will see that we are selecting a city where StateId = lstState.SelectedValue. But we see SELECT * FROM [Cities] WHERE ([StateId] = @StateId). Then click on Next.

Finish

We do not want to test the query this time. So we click on Finish.

Display Field, Value Field

When we come back, this time also we are going to ask which field to display and which field to get the value of. So the text field and value field are going to be CityName and CityId respectively. Then click on OK.

Result

Now as shown in the slide, both Databound controls stand created.
Effect In aspx – ListBox, SqlDataSource

lstCity is a Databound control of type ListBox. So the tag is `<asp:ListBox ID="lstCity" runat="server" Width="100%" DataSourceID="sqlSourceCities" DataTextField="CityName" DataValueField="CityId" />`. I'd of SqlDataSource is sqlSourceCities. So the tag is `<asp:SqlDataSource ID="sqlSourceCities" runat="server" ConnectionString="%$ ConnectionStrings:EmpConStr %" SelectCommand="SELECT * FROM [Cities] WHERE ([StateId] = @StateId)" <SelectParameters> <asp:ControlParameter ControlID="lstState" Name="StateId" PropertyName="SelectedValue" Type="Int16" /></SelectParameters>" />`). In this the connection string attribute is same as we had seen earlier. SelectCommand is SELECT * FROM [Cities] WHERE ([StateId] = @StateId). To decide the parameters for this, there is a section in this control i.e. asp:SqlDataSource control and that section is SelectParameters section. In the SelectParameters section, the 1st parameter and the only parameter to this SELECT statement is going to be @StateId. The parameter is going to come from lstState where inside lstState there is property called as SelectedValue because SelectedValue gives the StateId. Its type is Int16 and name of that parameter is going to be StateId. This allows to create parameterized SELECT statement.
EmpMgmt – Finishing Touches

In this lecture you will understand:
- Regular Expression Validator
- Calendar Control
- CheckBoxList Control
- File Upload Control
- Database Transactions
- Inserting a record in DB
Steps Involved

The steps involved in creating this application were, creation of a new website, creation of a database, creation of a backend class and creation of a webform. In this lecture we will perform the 4th step i.e. creation of a webform. Part of it is done in previous lecture. Remainder of that is being done in this lecture.

Step 4.5: Add Pin Code Row

After the Cities row, we want to add the row for Pin code of the address. Then the address part gets complete. Pin code has to be numeric. In India, Pin code has to be 6 digits. So we need some kind of validation here. This kind of validation is provided with the help of a validator control. For that we are going to use RegularExpressionValidator.

Effect In aspx – Pin Validation

Row for pin code is going to begin with <tr> tag and ends with </tr>. 1st column contains the text Pin and 2nd column is going to contain a textbox. Id of the textbox is txtPin. Width is set to 50%. MaxLength is set to 6 such that user cannot type more than 6 characters. But this does not validate that user only types digits there. For that in the 3rd column, we have added the control which performs validation. RegularExpressionValidator is one such control with a tag <asp:RegularExpressionValidator ID="txtPinValidator" runat="server" ControlToValidate="txtPin" ErrorMessage="Invalid Pin" ValidationExpression="\d{6}" />. Id of this validator is txtPinValidator. We have to also mention which control will it validate. So ControlToValidate property is set to txtPin. In case the pin is incorrect the message to be reported back is Invalid Pin. To ensure only 6 digits have to be entered and not anything else, for that we have to understand how regular expressions are created and what exactly are regular expressions. Regular expression is a special language which allows to define a pattern of characters to match. In this case the pattern is identified with \d{6}. \d in regular expression will match any digits and the number of digits are there is 6. Unless this condition is satisfied, the pin code will be treated as invalid. In regular expression syntax there are many such patterns which one can use to construct a very versatile way of matching arbitrary input that user types.

More about validators

The CompareValidator allows to validate two input controls where text of these two controls will have to be same. In almost every Email application like gmail, rediff, google, if we go and try to open an account, it will ask to choose a password. Every time we have to type the password twice because if we type it once, we might make a mistake and we may not remember it. If we type it twice, the web application turns verifies that we have typed the same password in both the text boxes; if not it will report a mismatch. For that Microsoft has created a CompareValidator for it. If this validator is specified in our webform, we have to specify it two Ids and not just one control to validate Id. 1st one says the 1st input control to validate and 2nd one identifies the control which is the 2nd one having the input text to be entered. Sometimes we want to validate the range in case of an input control. Suppose we want to ensure that the age that is entered in a textbox when we are taking age as input is > 18 and < 70. For that we have to use the RangeValidator which specifies the min and max and within that range only, it will allow the values. RegularExpressionValidator allows to construct a pattern to match the input in a flexible and powerful way. Suppose we want to match Email address. For example, \w+([-+.]\w+)*@[\w+([-+.]\w+)*.\w+([-+.]\w+)]. In a regular expression \w matches any word character. Word means a to z. It could be capital or small. So \w+ says that one or more occurrence of any word character followed by -, + and . as mentioned in the example. Followed by another set of one or more word characters. * says there may or may not be any. We are trying to ensure that before @ in an email address, there is at least one alphabetic character. Followed by that alphabetic
character, there could be -, . or + sign. These are valid characters in Internet email address. After that at least one alphabetic character comes, followed by a . , + or – sign and then this is itself optional but it has to be followed by at least one actual . (dot). To represent actual . character, we are writing \. which means it’s a literal dot. So it’s an escape character. . matches any character. In [ ], we do grouping. Sometimes these standard validators that Microsoft has provided, does not meet our requirement. In that case we can use the CustomValidator control which allows to write C# code, which takes care of the complete validation of all the input parameters that a user might have typed in a web application and report error in case the validation is found to be incorrect or the combination of input is found to be incorrect.

**Step 4.6: Gender Row**

In Gender row, we are trying to use a RadioBulletList control. Radio buttons allow to mutually exclusively select among several possibilities. We have to select Male or Female.

**Effect In aspx – RadioBulletList**

We write the text Gender in the 1st column. In the 2nd column we create a control `<asp:RadioButtonList ID="rbGender" runat = "server"> <asp:ListItem Selected = "True" Value="True">Male</asp:ListItem> <asp:ListItem Value = "False">Female</asp:ListItem> </asp:RadioButtonList>`.

Id is rbGender. 1st item is going to be identified with the help of a tag `<asp:ListItem>`. Selected property is set to True to indicate that by default that is the one which is going to appear as selected. Text is Male and the value there is also we can control. Almost all data driven controls have one Text field and one Value field. The 2nd item is Female and its value is set to false. So there are only two possibilities. We had made it to true / false so that we can convert it to a boolean easily. There can be more than two items.

**Step 4.7: Date of Joining Row**

The date on which the employee joins can be selected with the help of a Calendar control. So we have chosen to add a Calendar control.

**Effect In aspx – Calendar**

For this we have to use a tag `<asp:Calendar ID="calDoJ" runat="server" />`. Its Id is calDoJ. Date of joining is never in future. There is a SelectedDateField. We must ensure that selected date is never greater than today’s date and for that we have to deal with events of the Calendar control. In Calendar control, we can navigate back and forth between months. So every time somebody navigates to next month, we must prevent it.

**Step 4.8: Add Calendar Evt. Hdlr.**

For that we have to select the Calendar control and in the Properties window that appears, we have to select the lightening symbol which allows us to decide to use one of the events that the Calendar control generates. One of the event that the Calendar control generates is SelectedDateChanged.

**Slide 12**

If somebody changed the selected date, chance will be given to us and if the selected date has changed, we can write C# code to correct that and change the selected date back to what we want. So we will double click on the SelectionChanged event.

**Add Evt. Hdlr. Code**

As soon as we do, a function calDoJ_SelectionChanged( ) is generated and we will have to type the code in event handler function calDoJ_SelectionChanged( ).
Slide 14

We write the code which ensures that the one which checks if (calDoJ.SelectedDate > 
DateTime.Now) and if it is true then we say calDoJ.SelectedDate = DateTime.Now. It can never go 
beyond the current time. This allows to reset the selection back to a date of our choice. Calendar 
control when sees this value being changed, automatically reverts back to the current date and will 
never allow the user to go beyond today. So we avoided date in future with the help of event handler.

Step 4.9: Photo Upload Row

Every employee must have a photograph in the database. So we have asked the user to upload the 
photograph. For that there is a readymade FileUpload control available.

Effect In aspx – FileUpload

The aspx for the FileUpload control is as shown in the slide. In the 1st column we have text Photo. In 
the 2nd column, we have control <asp:FileUpload ID="fileUplPhoto" runat="server" Width="100%" 
/>. Its Id is fileUplPhoto. The Browse button appears automatically. We do not have to do anything 
for that because the user is asked to browse the file on his desktop and when someone submits the 
form, the data for that file is also included in the data that is send as a part of postback. There is no 
validation required.

Step 4.10: Email Row

Now we are going to add the Email address row. Email address has to be validated because people 
sometimes type junk in the Email address field. So for Email address validation, we have chosen a 
regular expression. In ASP.NET Visual Web Developer 2008, some readymade regular expressions 
are already available. We create the row, in 1st column we type Email, in 2nd column we put a textbox 
and in the 3rd column we put the regular expression validator.

Email Validator: Reg. Expr.

We will select the regular expression validator. In addition we have also put a required validator. For 
the regular expression validator, there is a property ValidationRegularExpression, in the Properties 
window, we click on that.

Slide 19

When we do so, a new window opens. It shows some readymade regular expressions for commonly 
validated input. Among them, we will find Internet email address as one of the expressions. We will 
select that.

Slide 20

Once we select that, the corresponding regular expression will be shown in the textbox below that. 
We haven’t written that regular expression. We are used it. Then click on OK.

Effect In aspx – Email Validation

In the Email address component, 2nd column contains ASP textbox field and its Id is txtEmail. Third 
column contains the RegularExpressionValidator and its Id is txtEmailValidator. The property 
ControlToValidate says which control it is validating and it is txtEmail. ErrorMessage says that 
Invalid Email Address, in case regular expression matching decides that the pattern is incorrect. Then 
the actual pattern that is going to be used in the regular expression is specified with the help of 
ValidationExpression property. So the expression is "\w+([-\.+]\w+)*@[\w+(-\.[\w+]*\.\w+[-\+\w+]*)". Email address is also to be a required field. For this textbox, we are using multiple
validators. One is a RequiredFieldValidator control and 2nd is a RegularExpressionValidator control. In both cases we are trying to target the same input control. The Id of RequiredFieldValidator is txtEmailRequiredValidator.

Step 4.11: Hobbies Row

Now we will see the row which specifies the hobbies of a person. Hobbies is a kind of selection where more than one hobbies can be selected. For that we can use the CheckBoxList which allows to select more than one items. Then we will create a data source for it.

Effect In aspx – Hobbies

1st column contains the text Hobbies. In the 2nd column, we create the CheckBoxList control i.e. 

```html
<asp:CheckBoxList ID="chkHobbies" Width="100%" runat="server" DataSourceID="sqlSourceHobbies" DataTextField="HobbyName" DataValueField="HobbyId" />
```

Its Id is chkHobbies and DataSourceID is sqlSourceHobbies. sqlSourceHobbies is a SqlDataSource control i.e. 

```html
<asp:SqlDataSource ID="sqlSourceHobbies" runat="server" ConnectionString="<%$ ConnectionStrings:EmpConStr %>"> SELECT * FROM [Hobbies] */. Connection string can be specified declaratively. We have also specified the SelectCommand. [ ] is a convention that is used such that even if there is a space in the name of the table, there is no harm. In the CheckBoxList, which text is to be displayed for that we say DataTextField="HobbyName" and DataValueField="HobbyId". This control must be able to communicate with us that more than one checkboxes are selected.

Step 4.12: Button Row

Finally we will add the action button which is used to add the information to the database. The text of the button is Add Now!.

Effect In aspx – Add Button

The button tag in ASP is 

```html
<asp:Button ID="btnAdd" runat="server" Text="Add Now!" />
```

This completes the webform.

Step 4.13: Label & Image

We also want to ensure that once the information is successfully entered in the database, corresponding details are also entered to the user so that user knows what exactly got inserted in the database. For that we had added one label below the table and one image control so that user is notified about what image got uploaded and what information got inserted.

Effect In aspx – label & image

The label which displays the employee information is called lblEmpInfo and photograph is identified with the Id imgPhoto. Both these are set as invisible to start with.

Add Button event handler

Now we will have to add the event handler for the Add button. For that we will select the button and double-click on that.

Slide 29

As soon as we double-click on the Add Now! button, a btnAdd_Click( ) event will get added in the code behind.

Step 4.14: Employee.cs
In the Insert( ) function, the parameter that is going to be cityid in which the employee resides. We are going to make use of the connection string in the web.config, also in this class. So we create new SqlConnection object. ConfigurationManager is a class that allows us to access a section in the connection strings in web.config. We use EmpConStr and say ConfigurationManager.ConnectionStrings["EmpConStr"].ConnectionString. Now we must create the INSERT query. We must create a parameterized query and set it in the SqlCommand object. We have created SqlTransaction object reference in variable t. We create a query "INSERT INTO Employees " + "( EmpName, AddLine1, AddLine2, CityId, Pin, " + "Gender, PhotoUrl, Email) VALUES " + "(@EmpName, @AddLine1, @AddLine2, @CityId, " + "@Pin, @Gender, @PhotoUrl, @Email); SELECT @@IDENTITY ". If there is @@ then it indicates that there is a special variable in SQL server which we do not have to define. At the end of INSERT, the record is also going to get automatically assigned a numeric identity field value. If we write the SELECT statement which selects the variable @@IDENTITY, the SQL database server will give us the value of that Identity Column. Unless we do that, we will not know the Id of the employee. In this SQL statement there are two statements, one followed by another. To execute the command, we set the cmd.CommandText = insertQuery and then say cmd.Connection = conn.

Slide 31

Now we need to add all parameters. So we have added all the parameters one by one as shown in the slide using cmd.Parameters.Add( ). To execute the command, we open the connection. But instead of executing the command immediately, we start database transaction. In the process of insertion, we are going to update the Employee table. So, one record gets created there. We are also going to update the EmployeeHobbies table multiple times. Because if the employee has 4 hobbies then we are going to make 4 insertions in that table. So together we are performing 5 insertions. Any time when we are changing database in more than one rows, in more than one tables, we have to use transactions to ensure, in case the operation is terminated for any reason, in the middle of everything the database is not left in an inconsistent state. Transactions allow us to do that. Transaction will keep track of all the changes that are made against a particular transaction that we begin and unless we commit it those changes will never be visible to anybody else. As the transaction is happening, nobody else can make changes and mess around with these changes in the middle of it. If we say that in a middle of a transaction, we cannot finish the transaction because we have encountered the error, we have to roll back the transaction and then every change that we have made so far will be reverted back. So database will be brought back to the original state. So transaction ensures atomicity; either the old state or the completely new state. To begin a transaction, we have to use a connection object i.e. conn.BeginTransaction( ). The method BeginTransaction( ) can only be called after the connection has been opened. We have collected the return value of that in variable t. Then we have assigned this transaction reference to the property cmd.Transaction i.e. cmd.Transaction = t. Then we fire the query. For that we use ExecuteScalar( ). So we say cmd.ExecuteScalar( ) because we are going to return the Employee’s Id. We collect the return value in mEmpId. So we get the Id for newly created employee.

Slide 32

Now we want to clear the parameters array in the ArrayList because we are feeding new parameters. So we say cmd.Parameters.Clear( ). Then we want the text for insertion. So the text is "INSERT INTO EmployeeHobbies ( EmployeeId, HobbyId ) " + "VALUES ( @EmpId, @HobbyId ) ''. For every hobby, EmpId is not going to change. So for 1st parameter we say cmd.Parameters.Add( new SqlParameter( "@EmpId", SqlDbType.BigInt ) ). Then we add the 2nd parameter using cmd.Parameters.Add( new SqlParameter( "@HobbyId", SqlDbType.TinyInt ) ). Then we assigned the value to the 1st parameter by saying cmd.Parameters[ "@EmpId" ].Value = EmpId. We obtained the EmpId after doing insert in the Employee table. Then we run a for loop on the list of hobbies that we have obtained. Inside the loop, we say cmd.Parameters[ "@HobbyId" ].Value = Convert.ToByte( (TBD)
hobby) and then fire the query using cmd.ExecuteNonQuery(). Now we commit the changes to the database using t.Commit().

**Slide 33**

If there is an exception, the transaction is no longer committed. For that inside the catch block, we check if (t != null) and if it is true then we roll back the transaction using t.Rollback(). In the finally clause, we check if (conn != null) and if it is true then we close the connection using conn.Close().

**Step 4.14: Change Default.aspx.cs**

The GetEmployeeInfo() function returns object of Employee class. So we begin by calling the default constructor and create a new object. Then we set Name, AddLine1, AddLine2, etc. other properties to the corresponding controls. For example, txtname.Text in case of name, lstCity.SelectedItem.Text in case of city, etc. For gender we say rbGender.SelectedItem.Value and convert it using Convert.ToBoolean(). Finally we want to construct the list of hobbies. For that we go through the collection of chkHobbies.Items and for each ListItem in that list, we check if the item is selected using if (item.Selected) and if it is selected then we say emp.Hobbies.Add(item.Value). Finally we return the emp object to the caller. So this reads all the things except image.

**Default.aspx.cs: DisplayEmployeeInfo**

When we display the employee information, we have to put line breaks between different properties of that employee and then create a string using String.Format(). We set the return value of that into the property lblEmpInfo.Text and it ensures that the correct text had displayed. For hobbies we walk through collection of hobbies using foreach loop and then append it to the label lblEmpInfo. Then we make lblEmpInfo.Visible = true and imgPhoto.Visible = true. The URL for that image control that we had inserted has to be set to the PhotoUrl Employee object i.e. we say imgPhoto.ImageUrl = emp.PhotoUrl.

**Default.aspx.cs: btnAdd_Click**

Inside this first we create the Employee object and get the object’s properties using GetEmployeeInfo(). Then we check if the file has indeed been uploaded by checking fileUpFile.HasFile. So we check if (fileUpFile.HasFile && fileUpFile.PostedFile.Length < 200 * 1024) and if these two conditions are satisfied then we are going to create a directory called as images in the root directory. For that we use Server.MapPath() method to map ~/images/. Then we use Directory.Exists() method to check if the directory already exists. If it is there then we do not want to create it. If it is not there then we call Directory.CreateDirectory(dir) to create the images directory. Then we call fileUpFile.PostedFile.SaveAs(dir + fileUpFile.FileName). Then we say emp.PhotoUrl = "~/images/" + fileUpFile.FileName. Once this is done, we have to actually insert the employee inside the database including the URL for the photograph. For that we have to select city using lstCity.SelectedItem.Value and pass it as a parameter to the emp.Insert() function. This actually inserts the record in a database. Then we say tblReg.Visible = false and then we call the function DisplayEmployeeInfo(emp).

**Testing the app.**

When we execute this project using CTRL + F5, we will get the web browser as shown in the slide.

**Slide 38**

Then we insert the name, address line1, address line2, state, city and pin code. Then we selected the gender and lastly the date of joining for the employee as shown in the web browser in the slide.
Then we browse the photo url, enter the email address and select different hobbies. Then we click on Add Now! button.

Result

When we click on the button, actual addition will happen to the database. Then we will also see the label which indicates that information is inserted in the database and photograph tells that file was actually uploaded to the server. So the final result is as shown in the slide.
 GridView Control

In this lecture you will understand:

- Data access approaches
- Declarative approach - Data-binding model
- Players involved in declarative model
- Using GridView control to display contact book
Data Access Approaches

The first approach that we have seen is a programmatic approach. We have seen that we can write a backend class, put the implementation of that class in App_Code directory and use that class to connect to the data source. It might be XML file, it might be text file or it might even be a database. In the process, we involved several classes in the two important namespaces. System.Data is one of those two namespaces and System.Xml is another when data is coming from the XML file. Another approach that ASP.NET 2.0 made very popular with ASP.NET developers is the declarative approach. In the declarative approach rather than writing the code, we create a set of rules or tags in the aspx file itself which is enough to describe the intension and get the data displayed in a webform. This approach is the primary approach on the basis of which GridView control is found. So this serves as the basis for GridView control. Once we know how to use the declarative syntax with GridView control, even other controls can be programmed with the declarative approach. The advantages of declarative approach will become apparent when we actually take a look at.

Declarative Approach – Key Players

Data that we are going to display or edit inside the webform is going to come from some source. Sources could be of variety of nature. It could be a Microsoft SQL database, could be XML file, could be a Oracle database or it can be a MYSQL server database as well. It does not really matter, what exactly is the source of data. To be able to get that data and use it in ASP.NET, we need DataSource controls. DataSource controls connect to the database. Connection mechanisms could be different for each database server or each DataSource represented in the system. But most of these DataSources are connected to using a connection string. Then these controls automatically have features to be able to read the data from the source, write the data to the source, create a new record in the source and so on. So these controls are specialists in connecting to the database and reading and writing data between the application and a database. We do not have to worry about connection mechanisms and actual procedures involved in reading and modifying the data. Although we use the word Controls here, remember that DataSource controls do not have any user interface. So as far as end user is concerned, he does not see anything visible on his page when he loads a ASP.NET form. So these controls have the sole purpose of getting the data and making changes to the data from the DataSource. So they act as an intermediary for a broker between the source and the controls called as DataBound controls. GridView also happens to be a DataBound control. The word Bound indicates that they are somehow connected to the data. It is with the help of these DataSource controls that the DataBound controls connect to the data. So they do not connect to the DataSource directly. They use these DataSource controls as an intermediate. To specify which DataSource control, we are using, we have to use a property called as DataSourceId property. So this property allows us to specify the Id for the DataSource control within the page which is responsible for connecting to the DataSource and loading the data along with modifications if any. DataSource has the data. DataSource controls have the ability to work with the data and DataBound controls have the specialty in displaying the data to the end user. Because the DataBound controls also generate the markup i.e. HTML required to allow user to interact with the web application. They are the ones which are actually seen by the user. User never sees the database. User never sees the DataSource controls. But user definitely sees the DataBound controls.

Data Binding Model

When we talk about a source for data, three primary sources comes; SQL server which could be Microsoft SQL server, XML which is a XML file or other kinds of data sources like an object created for a class that we have created. So object data source is another. To fetch the data from these sources in ASP.NET 3.5, there are several DataSource controls. So there is a SQL DataSource control. There is an object DataSource which allows to connect to something that is really an object of a class in C# or in VB.NET. XML DataSource for fetching the data from an XML file, writing a data to a XML
file and the new is LINQ DataSource. LINQ stands for Language Integrated Query. These four primary DataSource controls feed into the popular DataBound controls that we have in ASP.NET 3.5. We have seen ListBox, CheckBoxList and DropDownList. There are several others like DetailsView, GridView, TreeView, a Menu control and FormView. There is large number of such DataBound controls. Their ability to work with the display purpose of data makes them special. But they could be receiving the data from any one of these DataSource controls. For example, now the GridView control might be displaying the data from the SQL server database but the GridView is capable of displaying data even from an Object source. So DataSource control is a generic mechanism whose purpose is to feed the data to the corresponding DataBound controls. So this three layer separation between the DataSource and the DataBound controls allows to plug-in different kinds of sources that feed to various DataBound controls.

**Goal**

Our goal is to load the data from ContactsDB.mdf which is a Microsoft SQL server database file. To work with that, we are going to use SQLDataSource control so that we can generate queries for loading the data, inserting the data, updating the data and deleting the data and the data that gets retrieved is displayed inside a GridView control. GridView control is about displaying the data in a tabular format. Earlier we wrote our own code to create the table, to add the rows, to add the data to the rows. All of that was done ourselves. Now we are going to see a declarative approach for doing this with the help of GridView and SQLDataSource control. Our end objective is shown in the slide.

**Steps Involved**

In this case there are four distinct steps. First is we copy the previous project Contacts_SqlSimple to the project Contacts_GridView which is the name of the new project. Once we do that, we have to open the Default form and then add the GridView control to that form. Once the GridView control is added, we have to specify where the data for this GridView control is going to come from. For that we need DataSource control. Once the DataSource control is bound to a database, we are going to display the data in the GridView.

**Step II : Add GridView Control**

Once the project is copied, now we need to add a new form to the existing project. Rather than using Default.aspx, we are going to create a new form ContactsGrid.aspx in our project. For that we go to the Solution Explore, right click on the project title and say Add New Item. From the dialog that appears select Web Form, and name the form as ContactsGrid.aspx. Then click on Add.

**Change Title**

Once the form is added, the default title is Untitled Page. We change it to Contacts :: GridView. Then go to the Design view.

**Drag & Drop GridView**

If we go to the Design view, we will initially see a blank page. We have added Heading1 to the page. Our intention is to display the contacts information in a GridView form. For the GridView control, we have to open the Toolbox. If we open the Toolbox on the left hand side, there is a category of controls called as Data controls. Among the several controls, there is a GridView control. We can either double click on it or we can drag it and drop it on the form with the help of left mouse button. By default the name of the GridView control that we drag and drop is GridView1. We rename it as ContactView.

**Step II : Bind To DataSource**
Once we drag and drop, in the Design view, we will see a default view of this GridView control which says that there is Column0, Column1 and Column2. There is abc in the cells of the columns. This is the pictorial representation of how the data would be presented to the user or displayed to the user. GridView right now does not know what is going to be displayed in Column0, Column1 and Column2. We have to help it to do that. For that we have to go to the tasks for GridView. For that we have to select the GridView and click on the Smart tag which is next to the GridView control. Among the tasks it says, there is a Choose Data Source task. If we click on the drop down, we will find that there is currently no data source. So we will begin with New Data Source.

Choose Data Source Type
When we select New Data Source, a dialog as shown in the slide would appear. Here we will have to begin with the selection of the Data Source Type. We will select Database as the type of the Data Source. We are going to call it ContactsDataSource. Then click on OK.

Choose Data Connection
When we click OK, we are going to be asked to select the connection string. Our database file ContactsDB.mdf is in the same project inside App_Data folder. So we click on the drop down arrow.

Choose Data Connection
In the list, there is ContactsDB.mdf. We will select that. The corresponding connection string gets created.

Choose Data Connection
As soon as we select ContactsDB.mdf, the Next button will get highlighted. We click on Next.

Save Connection String
Now the wizard will ask us, if we want to save it to our application configuration file which is web.config. We are going to call it ContactsCStr as a contacts connection string. Click the Yes checkbox so that the connection string gets saved and then click on the Next button.

Configure SELECT Statement
When we click on the Next button, the wizard will actually connect to the database server. Download the list of tables and the list of columns in those tables. Here only one table is present. The name of the table is Contacts table. From the Columns, we select * to configure the SELECT statement which includes all the columns from that table.

Select Fields
So the default SELECT statement is SELECT * FROM [Contacts]. Then we click on Next.

Finish
If we want, we can test the results of the query and then we click the Finish button.

Test ContactView
When we come back, on the page just below GridView, we will actually see the SqlDataSource control with the name ContactsDataSource and that is the DataSource to which our DataBound control GridView is bound. Now we see a 3 tier model; DataBound control, DataSource control and an actual database. We have to set ContactsGrid.aspx as the startup page by right clicking on that page and then say Set As Start Page. Then we hit CTRL + F5.
Result

When we load the form, we will see that in the GridView window, the column Name, Telephone and Mobile number appears. Then the name appears Sachin R Tendulkar appears in the 1st row. His telephone number and the mobile number appear in the next two cells of the same row as shown in the slide. The SELECT query once we configure is being fired by the DataSource control to fetch the data. The results of that query are being fed into the GridView control and GridView control takes it and creates the HTML markup necessary to display the data in a tabular format.
GridView Edit And Delete

In this lecture you will understand:

- Operations supported by GridView
- Why can't we perform Edit and Delete
- What changes to make to perform Edit & Delete
Goal
As we did in the past, our objective is to display the data in the SQL server database file ContactsDB.mdf in a GridView. For that we have used the SqlDataSource control as an intermediary and the GridView is no longer going to be expected to perform the display operation. We expect that the GridView should allow to edit existing information and delete the existing contacts from the table. Snapshot given in the slide contains links to edit a contact and to delete a contact in every row of the GridView. Those capabilities certainly exist.

Edit & Delete
GridView is capable of editing existing information about the data that is being displayed in the table. It can also delete existing information from the DataSource. GridView is capable of selecting an item within any row in the GridView and that selection can be an event which might be used by some other control on the page. GridView requires that data which is displayed in each row of a GridView has to be uniquely identifiable. In the set of operations that we have performed, we never ensured before inserting a name into the database table that the name is unique. Secondly we have not inform the GridView that the name is the unique attribute among the set of columns that are available in the database table Contacts. GridView automatically could not figure out that name is unique because in the database that we have created, we created a table which contains three columns Name, Telephone and Mobile number. We never made any column a Primary Key column. GridView is capable of figuring out that a primary key exists in a database table but since we never created any column which is a primary key, GridView was incapable of making changes. Because GridView needs to identify a row and only then it can make changes. Fix is to change the database design. We are going to change the Contacts table. We will add an IDENTITY column in the Contacts table. Name of that column is to be ID. Once there is an Identity column, there is a guarantee that each row can be identified uniquely using that Identity column because database server assigns a unique number to each row in this Identity Column. Once this becomes unique, we can also mark it as a Primary Key. There after we have to reconfigure the control on the page to take care of the changes that have been made in the database and then we see that GridView is capable of making changes like Edit and Delete.

Steps Involved
There are total 4 steps of doing this process. First is to change the database design and then save it. 2nd is to change the Data Source control because right now the SqlDataAdapter control fetches only three columns from the database. We have to change it. In addition to be able to make the changes, we have to also make it aware of the Update query and the Delete query. We will then provide some additional event handlers. Each control generates a set of events. GridView also generates events when certain action happens with the GridView. For example, a contact gets deleted. If the contact gets deleted, our requirement is that, the display should be refreshed automatically. Similarly if the contact gets updated, the display should be refreshed automatically. Once all this is done, then we will build it and test it.

Step I : Change Database
For that go to the Solution Explorer. Then double click on the file ContactsDB.mdf file. When we do so, the Database Explorer view will be opened. In that in the Tables tree control, there is a single table called Contacts. Double click on that Contacts table.

Primary Key & Identity Column
As soon as we do, another window opens in the Editor and that window displays the design of the table. Only change is required here is to create an Identity Column called ID. Type of that column is
going to be int. Mark it as a Primary key. Then go in the Properties window and then select ID as the Identity Column. Once this is done, then we save the changes so that the database is updated. For existing data in the database, the Identity Column automatically assigns unique values. Database server takes care of this.

**Step II : Configure Data Source**

Then in the Design view, we will select the Smart tag which is next to the GridView control. When we do so, some tasks will appear. We will select Configure Data Source.

**Choose Data Connection**

Then we will choose the connection as ContactsCStr which is the connection string. Then we click on Next.

**Generate Other Statements**

When we do so, this time again a connection is established to the database server and the list of tables and columns in that table are fetched from the database server again. So the table Contacts is selected. Now there are four columns in which ID appears at the bottom because database server changes have been taken into account. Here we select *. Our SELECT statement is good enough because we say SELECT * FROM [Contacts]. So, all the four columns will be selected. Now we want to enable GridView to do editing and deleting. The source must support editing and deleting which means source must have queries for Update and Delete. We can provide these queries ourselves. But instead of that, we will click on the Advanced button in this step.

**Generate Other Statements**

When we do so, it shows another dialog. There is a check mark there saying that Generate INSERT, UPDATE and DELETE statements. If we click this checkbox, the corresponding SQL statements along with their parameters are generated automatically by the Data Source Configuration wizard. So we do not have to write the query statements ourselves. So we select the checkbox button and generate all update related queries. We select that button and then click on OK.

**Finish**

Then we come back, we click Next and then finally by clicking the Finish button, we finish the Data Source configuration. We have selected additional column from the database table Contacts, we have added a set of queries that would be required by the DataBound control to be able to make changes. All these things are necessary before GridView can perform editing and deletion. So GridView is capable of doing Editing and deletion only if the DataSource control to which it is bound, is capable of doing it.

**Enable Editing & Deleting**

When we come back, as shown in the slide, we will see that the GridView is automatically refreshed. The fourth column is ID. So we are now in a position to make changes. At the bottom in the GridView task, there are additional checkboxes appear. There are Enable Paging, Enable Sorting, Enable Editing, Enable Deleting and Enable Selection. Out of these we select Enable Sorting, Enable Editing, Enable Deleting and Enable Selection.

**Add DataKeyNames Attribute**

Now we will go to the Source view. In the Source view, we will go to the GridView tag. <asp:GridView ID = "ContactView" runat = "server" AllowSorting = "True" DataSourceID = "ContactsDataSource" DataKeyNames = "ID">. We will add a line DataKeyNames = "ID" in this
GridView tag. Everything between `<asp:GridView>` and `</asp:GridView>` remains same. Once the GridView is made aware of the fact that the key in the database column or table that it is being displayed is a column name ID, it can now uniquely identify a column and generate that as a parameter for the Update as well as Select statement. So it can generate a right kind of request and make those parameters available for the SqlDataSource. GridView and SqlDataSource work together closely to actually carry out these operations and unless GridView knows which is the primary key column, it will not be able to supply the correct parameters during the Update as well as deletion process. Then save the aspx file.

**Result**

If we go back to the Design view, we will notice that in the Design view, the first column has changed. In the 1st column, there are three hyperlinks for each row. 1st hyperlink is Edit, 2nd hyperlink is Delete and 3rd hyperlink is Select. These are the links that the user will have to click on to carry out those particular operations.

**Step III : Add Event Handlers**

Select the GridView control. Select the lightening symbol in the Properties window. It shows the events. We will select two events; RowDeleted and RowUpdated. So we will double click on RowDeleted and RowUpdated events.

**Event Handler Stubs**

When we do so, in the code behind file ContactView_RowDeleted( ) and ContactView_RowUpdated( ) event handler functions get added. These handlers allow to do something which is specific to these actions. These events are generated after the deletion and after the updation respectively.

**Type Event Handling Code**

Here we need to retrieve the updated data from the source and refresh the display. For that we add a ContactView.DataBind( ) statement in both the event handler functions. When we call a method DataBind( ) on a control, it will connect to the source again and retrieve the data back, thereby displaying the most up-to-date data.

**Step IV : Build & Test**

Once we build it and launch it with the help of `CTRL + F5`, we will see that GridView contains one extra column which contains Edit, Delete and Select links as shown in the slide. First we will see Edit operation. So we will click on Edit.

**Make changes**

When we click on Edit button, GridView automatically converts all the columns into editable format. It converts Sachin R Tendulkar as a textbox. It converts telephone, mobile and ID columns into the editable columns by making them look as a textbox. In the 1st column instead of Edit, Delete and Select links, now two different links are displayed; Update and Cancel. In this we can make changes in the name, telephone and mobile. ID cannot be changed because it’s a database primary key and it’s an Identity Column.

**Commit Changes**

If we want to confirm the changes after the typing, then click on Update button. If we want to cancel the changes then click on Cancel. It is like a Yes / No link. Suppose we change the name from Sachin R Tendulkar to Sachin Ramesh Tendulkar and then click on Update link. Behind the scene this
generates the PostBack to the server. PostBack to the server causes the GridView to send a request to update that specific record in the database with the help of SqlDataSource. After the Update has been done, a RowUpdated event will be generated by the GridView. Since we have an event handler registered, our event handler ContactView_RowUpdated( ) would be called automatically and inside that we rebind the control to the DataSource, thereby loading the most up-to-date source.

Delete Contact

So finally when we go back to the normal view of a GridView which is the readonly view, the name Sachin R Tendulkar has been replaced by Sachin Ramesh Tendulkar. So the database indeed has been updated. Our display has also been refreshed. If we want to do deleting, we have to click the Delete button. Delete button actually does the deletion immediately. There is no Yes / No type of prompt which is available to the user. Once we click Delete, in the next page the Delete query is generated by the SqlDataSource control. It is send to the database server. The row gets deleted. After that the GridView generates a RowDeleted event. We have an event handler registered for this event with the name ContactView_RowDeleted( ). So this event handler is going to be called on the server. All these requests are Post requests which are send back to the server except that actual operations are not being performed by us. We are doing something after the fact that the deletion has happened.

Result

When we do that, there is only one row which gets deleted. So in the final result the page is empty. There is only title Contacts :: GridView. Data has been deleted because there was only one row in the database table.
DetailsView Control

In this lecture you will understand:

- What is DetailsView control
- Using DetailsView ASP.NET control
  - Add New contact
  - Bind to database
  - Update existing contact
  - Delete existing contact
What is DetailsView Control

DetailsView control is a control in which the items in a record are displayed one row at a time and these items are displayed in such a manner that title of the column or heading of the column appears as the left column whereas the value appears as the right column. So this is a row by row layout where each row contains only two columns; on left we have the heading column, on right we have the value column. So Contact record which is being displayed in a DetailsView will have Name, Telephone and Mobile as the headings for individual rows and their corresponding values in the 2nd column in those rows. GridView control has the purpose of displaying one record in the database that we are trying to connect to. GridView occupies the width of the screen quite heavily and if the number of columns in a database table are too many then GridView will not fit in the horizontal width of the browser without having to do scrolling. DetailsView has a column by column layout where each column is displayed as a row in the DetailsView table. It does not matter how many columns are there, we can comfortably display the details using a DetailsView control and hence the name DetailsView. If GridView and DetailsView are present on the same webform, not all the columns would be displayed in the GridView control but if somebody selects a corresponding record from the GridView, every single detail about that record is displayed in a DetailsView control that appears below that control.

Steps Involved

To perform the task of adding a DetailsView control to a page, there are 5 distinct steps. First is to add the DetailsView control to the page. Then we will bind it to the database. We will enable the Edit, Insert operations. Then we will add some event handlers. At the end we will test the capabilities of DetailsView.

Step I : Add DetailsView Control

We will open the same project. On the form there is already a GridView control and its SqlDataSource control. We should expand the Data tab in the Toolbox and double click the DetailsView control or we can drag and drop it and place it below the SqlDataSource control that feeds the GridView. So now on the form, we will have three controls; one GridView, its DataSource control and just below that the DetailsView control. In the Properties window, we will change the ID of the DetailsView control to ContactDetails.

Step II : Bind To Database

DetailsView control displays Column0, Column1 and Column2 as the heading for respective rows which indicates that DetailsView is currently not bound to any source. So we will begin with the process of binding to the DataSource. So we will expand the Smart tag. We will click on the drop down button as shown in the slide. There we will see the SqlDataSource for GridView. But we are not going to use that. So we select New Data Source.

Choose Data Source Type

Now we are required to choose the Data Source type. We will choose Database as the type of Data Source and ContactDetailsSource as the name of the DataSource. Then we click on OK.

Choose Data Connection

When we do so, we will be asked to identify the connection string. The connection string is already saved in the web.config file. So we select ContactsCStr. We have to accept that and then move on to next step by clicking on Next.

Configure SELECT Statement
Now we are required to configure the SELECT statement. We want all the columns from the table Contacts but we want only single row. So we need a WHERE clause. So we select * and click on WHERE.

**Add WHERE Clause**

Now we add the expression to the WHERE clause. We have to select the Column on the basis of which the expression is going to form. So we click on the drop down list.

**Choose Column**

From the list of columns we select ID as the column.

**WHERE Expression**

Expression is going to be a equality expression. So we select = operator. We have to specify where the value of ID is going to come from i.e. we have to specify Source. So we click on the Source drop down list.

**Choose Parameter Source**

There are only two DataBound controls on this page. One is GridView and another is DetailsView. So what we display in the DetailsView has to be something that depends on what is being selected in the GridView. So nature of the Source for the parameter ID in this case is going to be Control. So we select Control from the Source drop down list.

**Parameter Source - Control**

Once we select Control, we have to specify what sort of Control it is going to be because the expression is [ID] = @ID. So we click on the Control ID drop down list.

**Parameter Source Control – ContactView**

We know that it’s going to be a ContactView control which is a GridView control on the same page. So we select ContactView from the list.

**Add Expression**

Once we select that, in the Value box we can see that ContactView.SelectedValue. So SelectedValue is a property in the GridView object. We use that as the input for selecting the single record to be displayed in a DetailsView control. Then we click on Add button.

**Result**

When we do so, it adds the expression to the WHERE clause. Then we click on OK.

**Select Advanced**

When we click on OK, we come back to the main screen where it displays the SELECT statement as SELECT * FROM [Contacts] WHERE ([ID] = @ID). To generate Update, Insert and Delete queries we click on the Advanced button. So we click on Advanced button.

**Generate Other Statements**

The dialog shows a checkbox which says click here to generate Update, Insert and Delete statements automatically. So we select the checkbox and click on OK.

**Finish**
Then we will click on Next button.

**Finish (2)**

Now we can test the query to make sure that the results are acceptable or we can click on Finish. So we click on Finish button.

**Step III : Enable Editing & Inserting**

When we come back, on the main screen, we will see that in the DetailsView control four rows appear. First row displays Name, 2nd row displays Telephone, 3rd row displays Mobile and 4th row displays ID. This is because now it has been bound to the DataSource control which is ContactDetailsDataSource control just below the table. So the page contains 4 controls now; 2 DataBound controls and 2 DataSource controls. So for each DataBound control, we have created a different source. Now we need to enable the support for adding new contact and editing existing contact in the DetailsView control. So we will click the Smart tag. Expand DetailsView tasks and then we will select Enable Inserting and Enable Editing.

**Result**

When we do so, we will notice that in a row below the four rows that just got created, two links got created; one link is Edit and another link is New. One is for editing the existing contact and another is for inserting a new contact.

**Step IV : Add Event Handlers**

Now we will select the DetailsView control. In the Properties window, we will click on the lightening symbol. Then select ItemInserted and ItemUpdated events. So we will double click on events ItemInserted and ItemUpdated.

**Event Handler Stubs**

In the code behind, the stubs are generated automatically by the wizard. In the code behind ContactDetails_ItemInserted( ) and ContactDetails_ItemUpdated( ) events handlers get generated.

**Add Event Handling Code**

Both event handlers require us to rebind to the DataSource. After inserting a new record, the information inside the GridView itself must be refreshed because now there is another record to display. Similarly if we edit the existing record, even the GridView needs to be refreshed. So we call two DataBind( ) functions i.e. ContactView.DataBind( ) and ContactDetails.DataBind( ) in both the event handler functions. We do it, in case the item is updated or in case the item is inserted. In either case, we must refresh the display.

**Step V : Build & Test**

When we launch the project with CTRL + F5, the page displays only the GridView. DetailsView in this case is paired with its master which is the GridView control. Unless something is selected in the GridView, it cannot be displayed in the DetailsView. So we click the Select button.

**Click Edit**

When we click the Select button, just below that the DetailsView will appear displaying the information about exactly the same contact. So Sachin Tendulkar is displayed in a row-wise fashion with Name, Telephone and Mobile followed by ID appearing as the rows in the table created by DetailsView control. Final row contains Edit and New links. Now we click on the Edit link.
Make Changes

If we click the Edit button, Edit and New are replaced by Update and Cancel. Depending upon what we are doing, different links appear at the bottom of the table. Existing values are there. There are textboxes to indicate that those values can be edited now. User can edit those values. Once we made the changes, we can click the Update button to actually make the changes.

Commit Changes

So we change Sachin Tendulkar to Sachin R Tendulkar. Then we click the Update button.

Click New

Now both the views are updated as shown in the slide. Because after the item was updated, ItemUpdated event was generated on the DetailsView inside which we called DataBind( ) on both GridView as well as DetailsView control. So both GridView and DetailsView are refreshed appropriately. To insert a new record with the help of DetailsView, we click on New button.

Insert Information

When we click on New button, the DetailsView enters in Insert mode. In Insert mode, textboxes would appear wherever we are required to supply some parameters. So Name, Telephone and Mobile are only become textboxes. ID does not get displayed because ID is a readonly field which comes from the database server itself. We never write to the Identity Column ourselves. So Insert query will have to be generated with only three parameters Name, Telephone and Mobile. Whenever the record gets inserted, its ID will be assigned by the database server. The links have now changed to Insert and Cancel which is like Yes and No.

Click Insert

Suppose we type Name as Rahul Dravid, his telephone number and mobile. Once we had done that, we can click the link Insert.

Result

Moment we do that, on the server the request get send through the POST. Actual effect of that request will be made by firing a query and getting the changes done in the database server. Followed by which the event ItemInserted would be generated. Newly inserted item must appear in the GridView. So we will refresh GridView and newly inserted item also appears in the DetailsView. So we must refresh the data binding for that. So we call DataBind( ) function on both of these in the ItemInserted event handler.