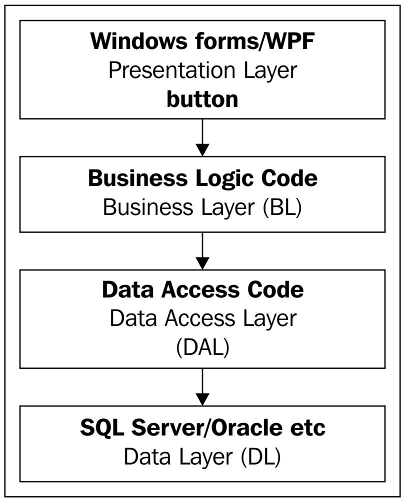
Most developers confuse the data access code (DAL) as the data layer (DL). The data and data the access layer are different. DAL is the actual code that we use in our applications to connect to a database, and the database itself is actually the data layer (DL).

Here is a sample diagram of how the different layers act:



Now in the diagram, if we separate each of the code layers into its own project and class library, then we will have a 4-tier project: Presentation tier, BL tier, DAL tier and DL tier (the physical database).

But with web based applications, we have a built-in 3-tier architecture by default. The presentation tier is the client-side browser (instead of Windows forms), the code (assuming you have web forms, BL, and DAL in one assembly) is the Application tier, and the physical database is the Data tier.

If we break up the web project so that we have the business logic and data access code in one assembly, and the web forms/`ascx` controls and so on in another, we will have a 4-tier architecture. We can go on like this by breaking each component out into its own tier and introducing further loose coupling. We will see more on how to introduce loose coupling in our projects in the later chapters of this book. For the rest of the book, we will be focusing only on **thin-client** based architectures, that is, web applications in ASP.NET.

We will now see what options we have for how we can break the code into different tier and layers in any Visual Studio web project, and thus define a few models. Here I am assuming that we are breaking the main application into tiers, and not focusing on the database and the presentation (browser) tiers.

Single Tier—Single Layer Model

We will have a single project in our solution, which will have UI, BL and DAL code under a single namespace.

ASP.NET Web Project compiling into a DLL in the `/bin` folder and under a single namespace: `MyApp`

No. of project files: 1

No of namespaces: 1

There is no separation of presentation, business logic, and data access code layers. Because we will have only one assembly (or set of assemblies) that cannot be distributed independently, this model would be single tier and single layer. We can use this model for very simple projects, on which only one developer is working and where we are sure there are no major scalability or maintainability issues. For example, a personal guestbook system, small 2 or 3 page web applications, or web sites with mostly static content.

Actually if you make an application based on the above model, it will follow a 3-tier architecture 'overall', if we bring the database and the browser as the other tiers and count them inside the application. This is the reason why I mentioned that for the time being we should forget about the external tiers and focus on how to break the monolithic ASP.NET application into further tiers.

Single Tier—Two Layer Model

In this type of solution, we will still have only one web project, but we will separate the UI code into one namespace, and the BL and DAL into another namespace.

ASP.NET Web Project that has two folders:

- **Code:** This folder will have class files containing business logic and data access code under a single namespace, say `MyApp.Code`
- **Web:** This folder will have the user controls, ASPX pages, and other presentation-related code under the namespace, say `MyApp.Web`

Here, as the business logic and data access code are logically separated from the presentation code, we have two layers. However, as all code files would still be compiling into assemblies under a single project's `/bin`, we will have only one tier. We can use this model for projects that have little or no business logic but need to access a database for content.