Physical Separation

In a tier-based architecture, we separate code physically into different assemblies (or a set of assemblies). For example, we may have a single assembly for the web project, and another one for the class project having business code. If we want to deploy our application across multiple servers, spanning different geographical locations, then we need to use an n-tier architecture (which we will study in the coming chapters of this book).

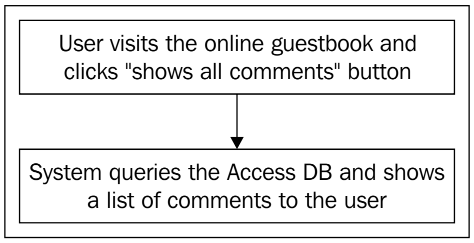
Logical Separation

Separating into layers mean that we logically separate the code, but the entire application will be a part of a single physical assembly (or a set of assemblies depending on the compilation model). We may put the code files into separate folders, each having its own namespace for easier code management and readability, but we won't have a separate assembly for each different namespace or part of the code. Also, unlike physical separation, it will not be possible to deploy parts of the application in a distributed manner.

So a "tier" is a unit of deployment, while a "layer" is a logical separation of responsibility within the code. A layer becomes a tier if it can be physically separated from the layers consuming it. Alternatively, a tier is a layer which could be physically separated from the layers consuming it.

[If we are using the Visual Studio 2005 Website model, then we may have a set of assemblies for each page/folder, whereas if we use a Web Application Project (WAP) model (similar to the one used in VS 2003) we will have only a single assembly for the entire project.]

Let's say we have a simple online guestbook system, which is a web-based application developed in ASP.NET. Here is a simple flowchart in a very basic form:


User visits the online guestbook and clicks "shows all comments" button
System queries the Access DB and shows a list of comments to the user

The user logs on to the website and visits the online guestbook, and clicks on the **show all comments** button. As a result, the system will show a list of comments to the user. For the same, the system sends a query to the Access DB, which in turn replies with a list of all the comments.

Now, one way to program this system is to create a simple web form with button, with the code to get the comments from the database placed inside the ASPX form (without using any code behind classes). The solution will compile into a single DLL. This inline coding approach will be discussed in the next chapter. Another method is to use code behind classes segregating the ASPX code and the C#/VB.NET code. We can introduce further loose coupling by separating the business logic and data access code into separate class library projects.

For a Windows-based project, also known as a **thick-client**, an n-tier project would have:

- Windows forms (or Windows Presentation Foundations, WPF) as the Presentation layer
- C# or VB.NET code handling the business logic as the Business Layer (BL)
- Data access code as the Data Access Layer (DAL)
- The physical database as the Data Layer (DL)

Data access layer (**DAL**) is a set of classes used to encapsulate data access methods like CRUD (Create Read Update and Delete) operations as well as any other methods accessing data from a data store (known as Data Layer). DAL's primary job is to communicate with the Data layer, which can be any RDBMS, set of XML files, text files, and so on. The DAL layer should act as a 'dumb layer' which is used directly by the BLL or any other service layer. The DAL layer should not contain any specific logic in its classes, and it should be used like a "utility" or "helper" class to fetch and store data to and from a data store.

Business logic layer (or the **BLL**) contains the business logic and set of operational rules particular to the application and talks to the data access layer (DAL) to:

- fetch data on which it has to apply rules
- save updated data after applying rules to it
- perform operations and validate data

BLL usually presents the data to the higher Layers (like a GUI layer) after performing business rules on it. This layer may also include error handling, logging, and exception handling strategies, besides encapsulating all the business rules of the project.

UI layer contains the graphical display components and files like ASPX, ASCX, MasterPages, stylesheets and so on. The UI layer usually is the Website or Web Project in the Visual Studio solution for ASP.NET projects.