**What is a "tier"?**

A "tier" can also be referred to as a "layer". A wedding cake is said to have tiers while a chocolate cake is said to have layers, but they mean the same thing.

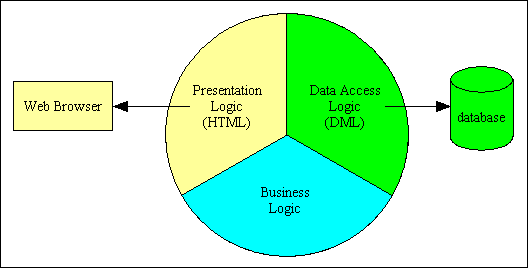
In the software world Tiers/Layers should have some or all of the following characteristics:

* Each tier/layer should be able to be constructed separately, possibly by different teams of people with different skills.
* Several tiers/layers should be able to be joined together to make a whole "something".
* Each tier/layer should contribute something different to the whole. A chocolate layer cake, for example, has layers of chocolate and cake.
* There must also be some sort of boundary between one tier and another. You cannot take a single piece of cake, chop it up into smaller units and call that a layer cake because each unit is indistinguishable from the other units.
* Each tier/layer should not be able to operate independently without interaction with other tiers/layers.
* It should be possible to swap one tier/layer with an alternative component which has similar characteristics so that the whole may continue functioning.

**The 1-Tier Architecture**

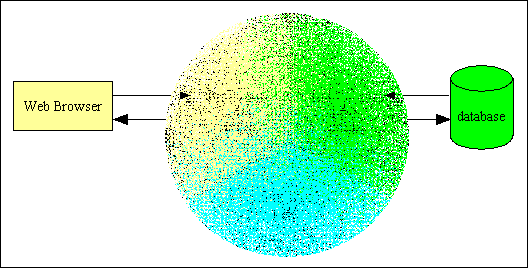
[Figure 2](http://www.tonymarston.net/php-mysql/3-tier-architecture.html#figure2) is a simple diagram which shows a 1-Tier application where the Presentation logic, Business logic and Data Access logic are all contained within a single component:

Figure 2 - 1 Tier architecture



Although this diagram apparently makes it easy to identify the different areas of responsibility, in real life the actual program code may be so inter-mingled, inter-twined and spaghetti-like that it would be extremely difficult to locate the boundaries between each area of responsibility. A blurry mess like this is shown in [figure 3](http://www.tonymarston.net/php-mysql/3-tier-architecture.html#figure3):

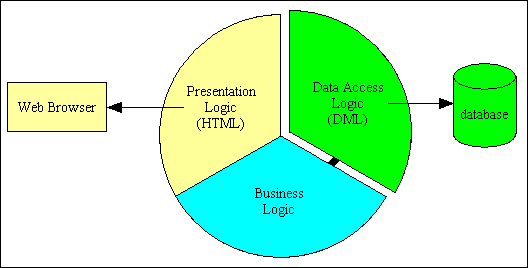
Figure 3 - Typical program with blurry boundaries



**The 2-Tier Architecture**

My first exposure to a software architecture which had more than one layer was with a compiled language where all database access was handled by a completely separate component which was provided by the vendor, as shown in [figure 4](http://www.tonymarston.net/php-mysql/3-tier-architecture.html#figure4):

Figure 4 - 2 Tier architecture



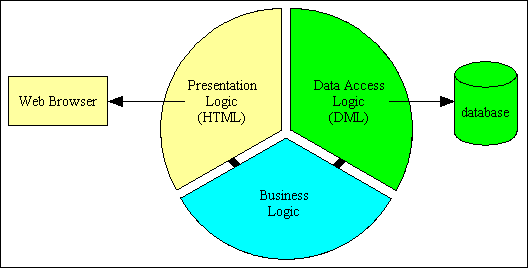
This was a deliberate feature of the language as it enabled an application to be readily switched from one DBMS engine to another simply by loading a different data access component. No other part of the application was allowed to communicate with the database, so this component could be switched without affecting any other part of the application. This made it possible to develop an application using one DBMS, then deploy it with another.

Note that the presentation logic and the business logic are still intermingled.

**The 3-Tier Architecture**

This is where the code for each area of responsibility can be cleanly split away from the others, as shown in [figure 5](http://www.tonymarston.net/php-mysql/3-tier-architecture.html#figure5):

Figure 5 - 3 Tier Architecture



Note here that the presentation layer has no direct communication with the data access layer - it can only talk to the business layer.

Note also that you should not infer from this diagram that the entire application can be built with a single component in each of these three layers. There should several choices as follows:

* There should be a separate component in the Presentation layer for each user transaction.
* There should be a separate component in the Business layer for each business entity (database table).
* There should be a separate component in the Data Access layer for each supported DBMS.

With this structure it is easy to replace the component in one layer with another component without having to make any changes to any component in the other layers.

* You can change the UI component so that you can switch between a variety of different output formats, such as HTML, PDF or CSV.
* You can change the data access component so that you can switch between a variety of database engines, such as MySQL, Oracle or SQL Server.

This structure also provides more reusability as a single component in the Business layer can be shared by several components in the Presentation layer. This means that business logic can be defined in one place yet shared by multiple components.

The main advantages of the 3 Tier Architecture are often quoted as:

* **Flexibility** - By separating the business logic of an application from its presentation logic, a 3-Tier architecture makes the application much more flexible to changes.
* **Maintainability** - Changes to the components in one layer should have no effect on any others layers. Also, if different layers require different skills (such as HTML/CSS is the presentation layer, PHP/Java in the business layer, SQL in the data access layer) then these can be managed by independent teams with skills in those specific areas.
* **Reusability** - Separating the application into multiple layers makes it easier to implement re-usable components. A single component in the business layer, for example, may be accessed by multiple components in the presentation layer, or even by several different presentation layers (such as desktop and the web) at the same time.
* **Scalability** - A 3-Tier architecture allows distribution of application components across multiple servers thus making the system much more scalable.
* **Reliability** - A 3-Tier architecture, if deployed on multiple servers, makes it easier to increase reliability of a system by implementing multiple levels of redundancy.

Another not-so-obvious benefit which can only come from actual exposure to having developed multiple applications using the 3-Tier Architecture is that it becomes possible to create a framework for building new applications around this architecture. As each of the layers specialises in just one area of the application it is possible to have more reusable components which deal with each of those areas. Such components can either be pre-built and delivered as part of the framework, or generated by the framework itself. This reduces the amount of effort needed to create a new application, and also reduces the amount of effort needed to maintain that application.

**E-commerce Architecture**

E-commerce is based on the client-server architecture. A client can be an application, which uses a Graphical User Interface (GUI) that sends request to a server for certain services. The server is the provider of the services requested by the client.

In E-commerce, a client refers to a customer who requests for certain services and the server refers to the business application through which the services are provided. The business application that provides services is deployed on a Web' server. The Web server is a computer program that provides services to "other computer programs and serves requested Hyper Text Mark-up Language (HTML) pages or files.

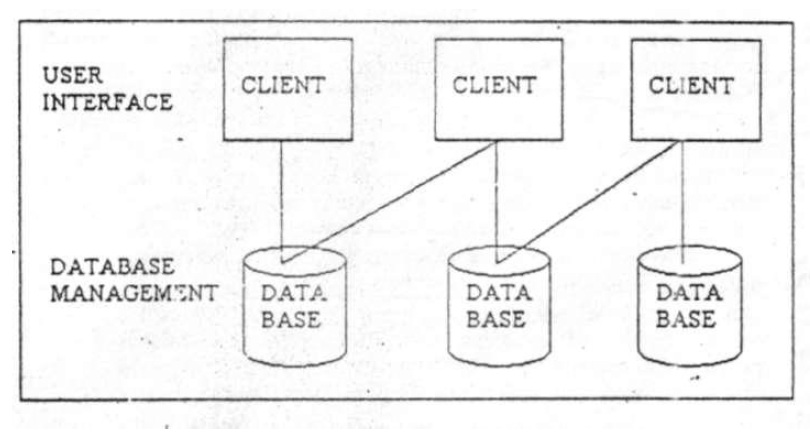
In client-server architecture, a machine can be both a client as well as a server. There are two types of client server architecture that E-commerce follows: two-tier and three-tier.

**Two-tier architecture:**

In two-tier client-server architecture the user interface runs on the client and the database is stored on the server. The business application logic can either run on the client or the server. The user application logic can either run on the client or the server. It allows the client processes to run separately from the server processes on different computers.

The client processes provide an interface for the customer that gather and present the data on the computer of the customer. This part of the application is known as presentation layer. The server processes provide an interface with the data store of the business.

This part of the application is known as data layer. The business logic, which validates data, monitors security and permissions and performs other business rules, can be kept either on the client or the server. The following Figure shows the outline of the two-tier architecture.

  
**Three-tier architecture:**

The three-tier architecture emerged in the 1990s to overcome the limitations of the two-tier architecture. In three-tier architecture, the user interface and the business application logic, also known as business rules and data storage and access, are developed and maintained as independent modules.

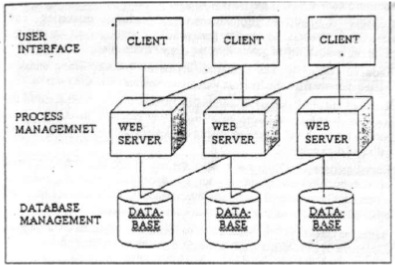
The three-tier architecture includes three tiers: top tier, middle tier and third tier.

The top tier includes a user interface where user services such as session, text input, and dialog and display management reside.

The middle tier provides process management services such as process development, process monitoring and process resourcing that are shared by the multiple applications.

The third tier provides database management functionality. The data management component ensures that the data is consistent throughout the distributed environment, the centralized process logic in this architecture, which makes administration easier by localizing the system functionality, is placed on the middle tier.

The following Figure shows the outline of the three-tier architecture.



**The client server architecture advantages:** 

The client-server architecture provides standardized, abstract interfaces to establish communication between multiple modules. When these modules are combined, they become an integrated business application. Each module is a shareable and reusable object that can be included in another business application.

In the client-server architecture, the functions of a business application are isolated within the smaller business application objects and so application logic can be modified easily.

In "the client-server architecture, each business application object works with its own encapsulated data structures that correspond to a specific database. When business application objects communicate, they send the data parameters as specified in the abstract interface rather than the entire database records.

This reduces the network traffic. In the client-server architecture, a programmer can develop presentation components without knowing the business application logic.

This architecture also helps a database analyst in accessing the data from the database without being concerned how the data is presented to an end user.