NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

HOTEL RESERVATION SYSTEM USING WEB SITE DESIGN

Graduation Project
COM- 400

Student: Al-Shalabi Mohammed (20021839)

Supervisor: Assoc. Prof. Dr. Rahib Abiyev

Nicosia - 2006

Dedicated to my mother, father,
Brothers and my sisters

ACKNOWLEDGEMENTS

This project is done under the supervision of Assoc. Prof. Dr. Rahib Abiyev, I am very grateful to him who gave his technical and emotional support for the creation of this graduation project.

And my Thanks go to whom my love will never end, my father and my mother, to my brothers and my sisters, that help me a lot and their encouragement in my studies, so that I could be successful in my life.

I will also like to thanks my all friends in Cyprus who gave their ever devotion and helped me for their all valuable information to complete this project.

ABSTRACT

The aim of the Project were to design a hotel reservation form online and letting the users by visiting my hotel website to reserve to themselves.

This project presents the development of hotel reservation system by using PHP, APACHE, and MYSQL. The characteristic of these languages are given.

The following techniques are used in the project, HTML (Hyper Text Markup Language) it the basic language to design a website , PHP (is a scripting language for writing web applications that execute on server-side. Scripting language itself is much alike C, however it still contains many differences) it is used also to active the pages you can say to do a connection between the pages or between the pages and database and MySQL it used to do some operation in the database tables as deleting, inserting, updating and selecting.

CONTENTS

1. INTERNET

1.1 Introduction

The internet is a vast world-wide collection of computers all linked together in such a way that messages can be passed between them independently of their architecture or their geographical location.

Many different computer architectures and platforms and operating systems are in use around the world – for example, PC's running Windows or Linux, Macintoshes, handheld palmtops with internet access via wireless.

The achievement of the Internet has been the development of a set of common protocols by which these computers can communicate, and a set of common languages by which meaningful information can be transmitted between and displayed on these different platforms.

Links are via the same communication mechanisms that are used for telephone communications:

· Satellite links

· Telephone Cables - e.g. fibre-optics

· Microwave links

1.2 The Nature of the Internet

The Internet is an extraordinary global communication system whose origins lie in a combination of government, academia, the military and the work of individuals. Although global communication systems have existed for many years, none has given the opportunities for global communication that the Internet has.

The Internet is a co-operative enterprise which depends on the voluntary support of its users. Without this co-operation the Internet as we know it would cease to exist.

The Internet and the World Wide Web are often confused. We must be careful to maintain a distinction between the Internet, which is the collection of networked

computers, and the World Wide Web which is the collection of hyperlinked documents which are stored on the hard drives of those networked computers.

1.3 The Internet - a brief History

The internet as it exists today grew out of a number of different networks of networks. Traditionally local area networks (LANs) and time share systems (such as Unix) are used within organization to share resources such as printers, common files and so forth. Within these organizations there is a degree of common purpose and trust. There is administrative support for assigning system resources, setting up passwords and discouraging anti-social behavior.

Connecting two different networks together is traditionally cumbersome for technical and social reasons. Incompatibilities between different manufacturers' systems provides one part of the problem - the other concerns issues such as "who is in charge?": Systems administrators from different organizations must co-operate and share responsibility and power. The achievement of the Internet has been to establish a common set of protocols and languages by which computers of different architectures and on separate networks can communicate data in a wide variety of different formats with each other and to solve the problems of administration

1.4 The Origins of the Internet

The impetus for a network capable of linking computers and networks of different architectures originally came from the US military:

· A system capable of supporting (text-based) communication between dissimilar computers and networks

· Minimal central administration to make the system robust

· Flexible in allowing the addition or removal of nodes to or from the network

The network should be decentralized and therefore not vulnerable to the loss of any individual computer site. A project was funded by the US Advanced Research Project Agency in the early 1970's and this led to the appearance of ArpaNet. Arpanet provided the essential more primitive facilities which we have come to expect of the Internet: email, ftp (File Transfer Protocol) and Telnet (remote login). How Arpanet became the

Internet. The US government encouraged the use of the Arpanet by Universities and other educational institutions as well as the military. The Arpanet thus became the backbone of an enlarged network that became known as the Internet. The Arpanet acted as a sort of backbone out of which the Internet as we now know it evolved by the addition and attachment of an ever-increasing number of computers (hosts) and networks.

1.5 The Growth of the Internet

The Internet has grown exponentially as more and more hosts became attached to the Backbone as shown in table 1.1.

Table 1.1 Internet Growth.

First appearance (early 1980's)	213 registered hosts
Feb 1986	2308 registered hosts
1995	> 5,000,000 hosts worldwide
1999	75,000,000 hosts worldwide
Jan 2002	142,000,000 hosts worldwide

1.6 Internet Architecture

The "backbone" of the internet is a distributed network of hosts each of which can act both as a client and as a server. In general there is no single path for communication between any two hosts as shown in figure 1.1.

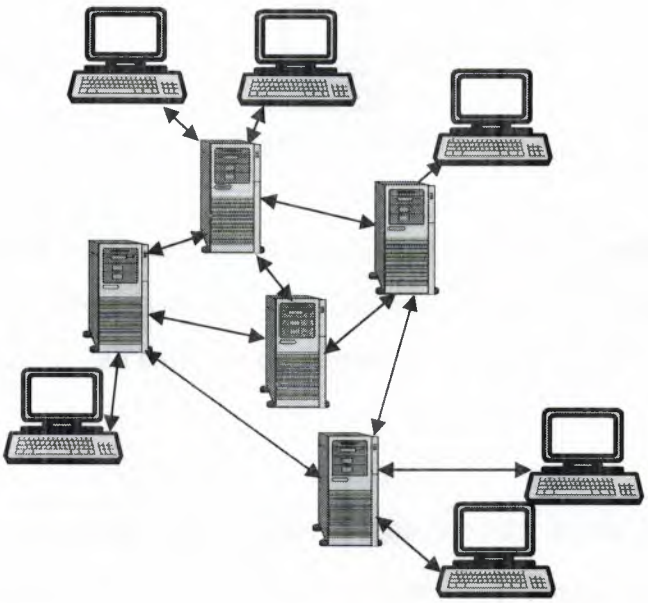

Figure 1.1 Communication Path Between any two Hosts.

1.7 Internet Architecture

It's said that the distributed client-server architecture that comprises the Internet was devised by the US military so that it would be relatively invulnerable to enemy action. In general, computers shown like this in the figure 1.2 will be hosts running software to send information to the clients:

Figure 1.2 Host.

PC's on the network shown like this will generally be clients, running software such as Internet browsers as shown in figure 1.3:

Figure 1.3 Clients

In your workplace or University, the server is likely to be a powerful centrally managed computer, while the client is the PC on your desk. However, this isn't a hard and fast distinction – you can install software on your PC which will enable it to act as a server as well.

1.8 How the Internet is Built- The Client-Server Architecture

The internet is a massive exploitation of one of the principal mechanisms by which computers exchange information: the Client Server Architecture is shown in figure 1.4.

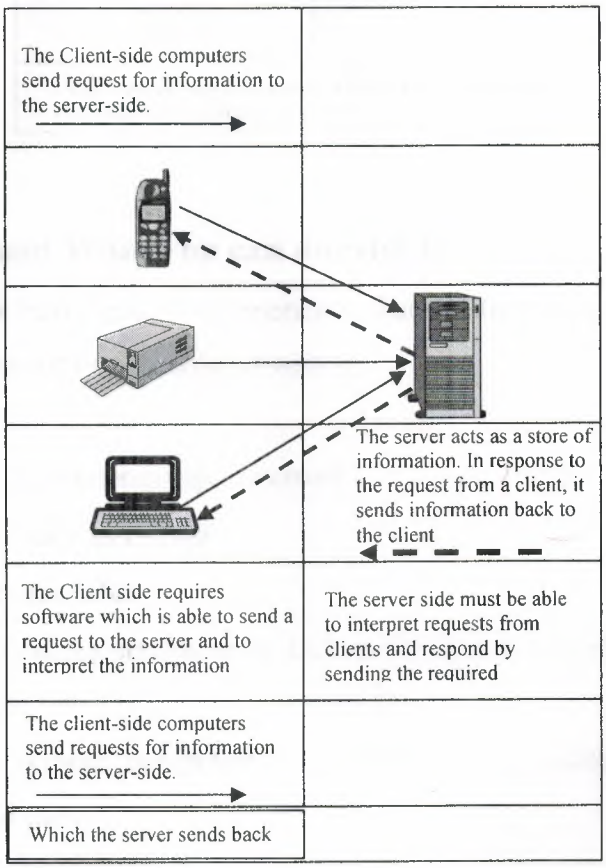

Figure 1.4 Client Server Architecture

Client and server software is shown in table 1.2.

Table 1.2 Client, and Software.

Client Software	Server Software
Internet Browsers: Internet Explorer Mozilla Firefox Netscape Opera	Web Servers: Apache Xitami Microsoft US
Mail Clients: Outlook Express Outlook Evolution Eudora	Mail Servers: Exchange Server
Web page Editors: Dreamweaver Front Page Quanta Mozilla Editor	Server-side programming Languages: PHP, ASP
Scripting Languages: JavaScript VBScript	
Other software might be equally useful on both server and client sides: for example, Java, Perl	

1.9 The Internet and What you can do with it

We can think of four basic classes of operation that the Internet provides, and each of these has a generic type of tool associated with it:

1. Communication

- electronic message interchange by email
- bulletin boards such as Usenet

2. File and Document transfer

- moving electronic documents around using File Transfer Protocols (FTP)

3. Remote Access

- Logging into remote computers by Telnet and accessing the databases and archives stored there

4. Interactive Browsing

- **The World Wide Web**: a hypertext based graphical and multimedia browser for documents stored on the servers of the Internet.

1.10 World Wide Web

The World Wide Web is by far the most recent of these applications of the Internet – the other three have been associated with the Internet since its earliest days. We'll spend some time with the WWW now since it's the most popular application and also the one that gives us most scope for Internet Programming. Many people tend to think that the WWW is virtually synonymous with the Internet, but in fact it is not.

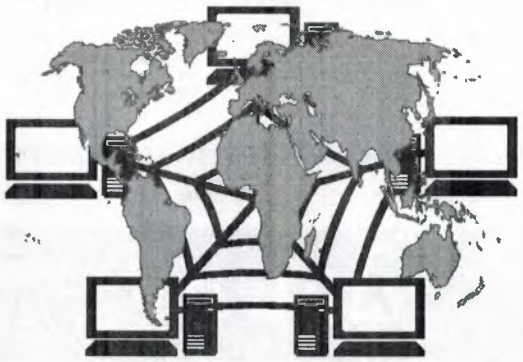

Figure 1.5 World Wide Web

1.10.1 The Internet and the WWW

The Internet is the distributed network of computers linked by electronic and optical communication media. The WWW is the set of documents including text, images, and sounds and so on, which is stored on the servers that are linked to form the Internet. Since these documents are themselves linked by means of hyperlinks they form a network or web in their own right. However, this is a web of documents rather than hardware shown in table 1.3:

Table 1.3 Internet and the WWW

		Consists of	Linked by
Document Layer	WWW	Documents, Images, Sound clips.	Hypertext transfer protocols HTTP
Hardware Layer	Internet	PCs, Workstations, Client Computers, Servers	Internet transport protocols TCP/IP

1.10.2 A "Typical" Web Page

Here's a fairly typical web page shown in figure 1.6 (http://www.awards@hotel.net):

Figure 1.6 Web Page

As you can see it consists of a mixture of text, images, tables and hyperlinks to other documents. In this example, the screen in divided into 2 frames: a sidebar containing links and a main screen with information.

1.10.2.1 Components of the Web Page

Even this relatively simple web page contains a number of components which are stored separately and assembled in the browser as shown in figure 1.7:

- text including tables
- images
- two separate frames

There are also hyperlinks that point to other documents. In fact the components that make up the web page might have been stored on separate servers which are geographically widely separated. Images might be loaded in from a remote server. Clicking on a hyperlink might take you to another part of the same document, or to a completely different document stored on the other side of the world.

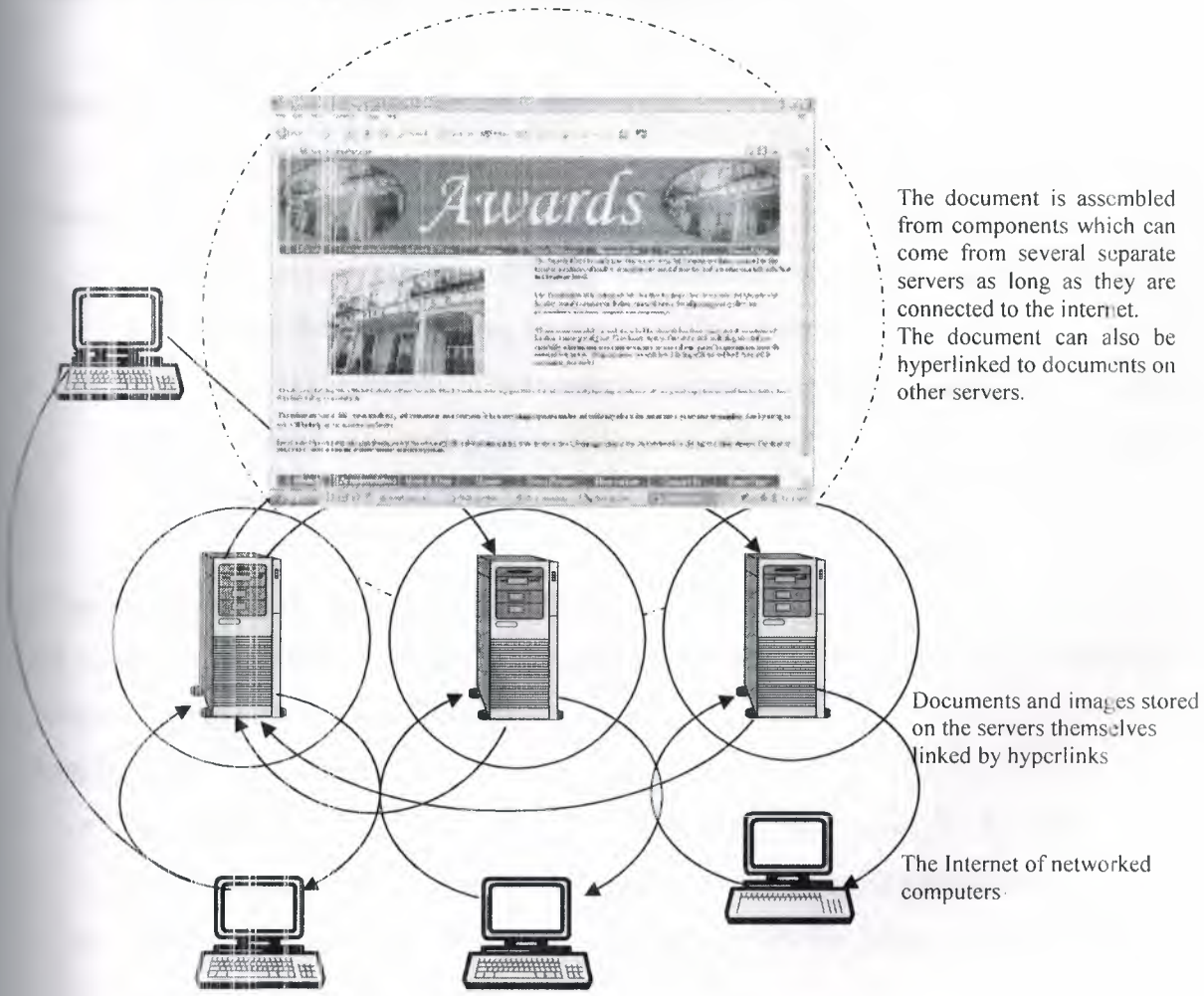

Figure 1.7 How Web Document is Assembled.

1.10.2.2 The World Wide Web

Too many people, the World Wide Web is the Internet, and using the Internet means using a Web browser to "surf" the information stored in the World Wide Web. In fact, the WWW is a relative newcomer– the other facilities have been available via the Internet from its earliest time.

- The WWW became possible with the increasing performance of computers
- The development of the World Wide Web is particularly associated with the name of (now Sir) Tim Berners-Lee of CERN in Switzerland
- The hypertext component of the Web allows you to go directly from one Internet document to another without deliberately keying in URL addresses.

Clicking on a link takes you directly to another document.

Distinction between the Internet and the WWW

- Sometimes the name World Wide Web is used as if it were synonymous with the Internet but this is wrong:
- The Internet is the network of interconnected computers and workstations connected and communicating by means of TCP/Internet.
- The World Wide Web is the network of documents, images, sound-files &c which are maintained on computers linked to the Internet and which are themselves linked into a vast network by the hyperlinks by which they are associated.

Who Administers the Internet and WWW?

It's important to realize that although very closely related, the Internet and the WWW are subject to quite different management

The Internet

- Administering the Internet is concerned with the protocols by which the computers that make up the Internet are able to connect with each others.
- Internet administration is heavily concerned with the Internet communication protocols – TCP/IP and others.

The World-Wide Web

- The administration of the Worldwide web is concerned with maintaining the common standards that ensure that web documents adhere to common languages that can be understood by all the web browsers.
- WWW administration looks after languages such as HTML and XML.

The Social Organization of the Internet

- Any internet user may be called upon to pass on data for others (obviously this only applies to users with more than one connection).
- This requires co-operation and trust between all users.
- In its early years, the Internet was a means of communication between universities and libraries and similar organizations whose primary aims were information exchange rather than commerce. The organization was largely self-policing.
- However, as the Internet has matured and become a medium for business, the laws and restrictions applicable to other media of communication have been increasingly applied to the Internet.

Internet Administration

- The Internet is not "owned" by any one organization, individual or state. However, some central agreement is required to maintain compatibility.
- Until recently, the backbone of the Internet in the USA was administered by the National Science Foundation (NSF); now, however, it is commercially run.
- Other regions of the Internet have their own funding and administration.
- Standards for connection are maintained by an Internet Advisory Board and networks wishing to join the Internet must adhere to the standards laid out by the IAB.
- Reports by the IAB are made public in the form of RFC's - Requests for Comment.

Client-Server Architecture and the Web Browser

- A web browser is the software that runs on the client. The browser obtains data from the server and presents it to the user.
- The web server responds to requests from clients (usually web browsers). A typical client server "conversation" will go as follows:

Client	Request document/ directory/ file.html
Server	Sends some "header" information including the size of document followed by a blank line followed by the document.
Server	Closes the connection
Client	Starts to display the text. Opens a new connection and requests the picture/ directory/pic.gif which is embedded in the page.
Server	Returns the picture, closes connection again.
Client	Displays the full page including the picture.

Where does the Internet Programming come in?

- Web pages are created using a so-called markup language HTML which is itself a form of programming language for displaying text and graphics in a way that is independent of the platform.
- Extensions to Web pages (e.g. to allow user input into forms) is provided by extensions such as CGI (Common Gateway Interface).
- We may want to write programs that can be run and viewed in Internet pages. The programming language Java has been specifically developed with the aim of facilitating the construction of small programs called applets which can be run from Web pages.
- Another application for Internet programming is the construction of software agents which are programs intended to carry out functions - such as the search for information on the Internet -autonomously on behalf of a user The creation of programs and documents which are published worldwide and are able to downloaded onto remote systems inevitably raises questions of security and legality. There is also the matter of etiquette.

Protocols for transfer of Information on the Internet

TCP/IP supports a number of protocols for transferring information across the Internet as shown in figure 1.8.

- We are most concerned with HTTP which is associated with the transfer of information via the World Wide Web
- The different media formats supported by HTTP – such as audio, text/html, video and pictures – are referred to as MIME types, where MIME is an acronym for Multipurpose Internet Mail Extension

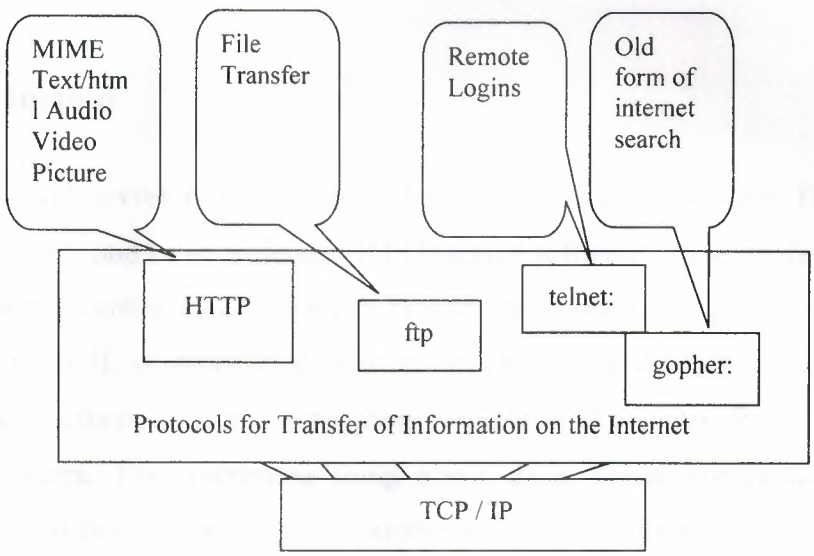

Figure 1.8 Protocols for transfer of Information on the Internet

2. INTRODUCTION TO THE APACHE SERVER

2.1Overview

Learn how to acquire, compile, install, and configure the Apache Web server. A great place to start if you are completely new to Apache, and trying to figure out where to start.

2.2 Introduction

The Apache web server project is more than just a piece of software. The Apache web server is the best, and most preferred, HTTP server software in use on the Internet today, and it was written entirely as a volunteer project, by volunteer programmers, in their spare time. That, in itself, is astonishing. That is, it is to people that are not familiar with the Open Source methodology, and Open Source projects like Linux, Perl, Sendmail, and a variety of others. The interesting thing about these volunteer-written, free software packages is that most of us, and our businesses, rely heavily on them, whether we are aware of it or not.

Before diving directly into talking about what Apache is, it is useful to talk about where Apache came from, and how it came to be.

2.2.1 The WWW

The Internet has been around for a long time. More than 30 years now. But for most of that time, it was entirely the domain of geeks and hobbyists. The main reason for this was that it was hard to use.

In 1991, Tim Berners-Lee developed something that he called the World Wide Web, while working at CERN. His purpose was to give quick and easy access to documents for geographically distributed people collaborating on projects. Along with a lot of help from the standards community (and, notably, Roy Fielding), they defined HTTP, HTML, URLs, and the other necessary components of making the Web a reality. He then went off, and with the help of colleagues around the world, communicating via email, developed the CERN web server, and a simple Web client, which he dubbed a ``browser." The name came about because there was very little of real value on the Web at that time, and all you ever really did was browse. Ironic that the name stuck!

2.2.2 The Apache Server

When Rob left the project, it left a problem. There were still a lot of people using his code, and actively making patches to the code, but there was no longer anyone collecting those patches.

In 1995, Brian Behlendorf and a small group of other developers started collecting these patches in a central repository. Brian got some space donated on a server, and set up a CVS tree so that developers could check in patches. And in April of 1995, they released the first official release (Version 0.6.2), which was given the name Apache, because it was ``a patchy server".

The Apache Group, as they were known at that time, had no formal organizational structure, never met, communicated only over email, and worked entirely in their free

time, on a volunteer basis. Early the next year, Apache passed NCSA as the most widely used server on the Internet, and is now used on more than 60% of all web servers on the Internet.

2.2.3 Apache's Architecture

Since the 1.0 release of Apache (December 1, 1995) Apache has has a modular design. The core of the server is very light-weight, and all other functions are implemented as modules that plug in to the core. This means that you can keep the size of the executable down by leaving out functionality that you don't need. It also means that if there is some functionality missing that you do need, you can write your own custom module to plug into the core.

2.2.4 The Future of Apache

At Apache-Con in Orlando, back in March, Apache 2.0 was released. This is largely a rewrite from earlier versions, and uses a threading model that will increase performance substantially on most platforms. As of this writing, version Alpha 6 of the 2.0 server has been released.

The Apache Group, as mentioned above, has become the Apache Software Foundation, and continues to take on new projects that seem to fit the larger vision that the ASF has for the future. Open Source, and open standards, produce better software. In the end, this makes life better for all of us, and we should support the ASF in all its endeavors, if only for purely selfish reasons.

2.2.5 Obtaining and Installing Apache

Apache is available as source code, and is probably available as a binary installation for your operating system, unless you are running something truly arcane and rare. And, of course, if you are, you can still get the source code, and compile it yourself.

2.2.6 Compiling and Installing

Most of the settings for your server, governing how it will operate, are done at the configuration stage, when you modify the configuration files that the server loads when it starts up. However, due to the modular architecture of Apache, a lot also depends on what modules you enable when you compile the server. The available configuration directives depend on the modules that are loaded.

You can either compile your server the quick, easy way, and get a default installation with the most common functionality, or you can get in there anad pick and choose what you actually want.

2.2.6.1 The Simple Way

The installation process for Apache is really simple for most folks. If you are just wanting to set up a simple web site to do the normal things like serve web pages, and maybe do some CGI, the installation process looks like this:

```
tar -zxf apache_1.3.12.tar.gz
cd apache_1.3.12
./configure --prefix=/usr/local/apache
make
make install
/usr/local/apache/bin/apachectl start
```

Assuming you have a reasonably fast machine, this entire process does not take much more than 10 or 15 minutes, and you have a functioning web site. The configure process figures out reasonable settings for your system, and so the configuration files will have reasonable things in them so that you can immediately start serving web pages from your server. The --prefix setting tells the configure process where you want to install the server.

/usr/local/apache is the normal place to do this, but if you want to put it somewhere else, just specify that on the command line:

```
./configure --prefix=/home/rbowen/devserver
```

apachectl is a handy tool that Apache installs to make it simple to start, stop, and restart the server, as well as some other handy functionality.

2.2.6.2 Advanced Installation

If you're like me, you are not satisfied with the default installation. It does not have all the modules that I want, it has some stuff that I'd just as soon leave out, and there are some things I just do differently because I do them differently. I'm strange that way.

There are two ways to handle this that I'm going to talk about. First, you can actually edit the configuration file, and specifically choose what you want to compile into the server. Or, you can just throw everything in, but do it in such a way that you can go back and add and remove stuff at your leisure. I tend to go with the latter approach, but the former approach gets more coverage in the docs, and so is used more frequently.

You are advised to use the simple method above the first few times you install Apache. Also, in version 2.0, there will only be one installation method, and it will look more like the quick easy method above, than like these methods here.

2.2.6.3 Editing the Configuration Scripts

In our previous example, we ran a script called configure (``small-c configure'') in the main Apache directory. In this method, we're going to go down into the src/ directory and actually look at the configuration files. After all, the motto of Linux is ``do it yourself.''

The process starts out the same:

```
tar -zxf apache_1.3.12.tar.gz
```

But rather than just going into the apache_1.3.12 directory, you need to go down into the src directory:

```
cd apache_1.3.12/src
```

Then, using your favorite editor, edit the file Configuration (``big-C Configuration") and comment, or uncomment, the lines that refer to options that you are interested in.

If you don't see a file called Configuration, copy the file Configuration.tmpl to Configuration, and use that as your template.

Once you have gone through and made sure that you have everything that you want in there, save the file, and run the following:

```
./Configure
make
make install
```

The simple method, which we talked about first, is doing all of this for you behind the scenes. If you run ``small-c" configuration, you will have a file called Configuration.apaci created for you, which will then get used in this configuration step.

2.2.6.4 Dynamic Shared Objects (DSO's)

I get tired of rebuilding and reinstalling my web server every time I want to add in a new module, or when I decide to take one out, because I never really use it. This is where shared objects are handy. A shared object is something that gets loaded dynamically by a process when it needs it. This saves you from having to compile that code into the program executable, which, in turn, makes the executable smaller, and load up faster. By making your Apache modules into shared objects, you can build everything into your server, but only actually use the parts that you want at any one time, and leave out everything else that you're not using.

On Windows, these are things called ``dynamic link libraries'', or DLLs. On Linux, they are called shared objects, or .so files.

In order to enable shared objects, you have to compile a module called mod_so, which, in turn, loads all the modules that you have compiled as shared objects. mod_so itself cannot be shared object, of course, because there would be no way to load it. Chicken, egg.

So, to build your Apache server to use shared objects, run the following commands:

```
./configure --prefix=/path/to/apache \
        --enable-module=most \
        --enable-shared=max
```

(You can either literally type those \ characters, or just put this command all on one line and omit the \'s)

What does this command do? Well, it compiles all of the modules that ship with Apache, except those that are considered experimental or unstable, and enables them all as DSO's.

This means that Apache will be loading up a bunch of modules that you don't really want, so you need to edit the configuration file and comment out those modules that you're not really going to use.

But it also means that if you want to add in a particular module, you can do so by putting it in the configuration file, rather than having to recompile your server from source. This is particularly handy if you change your server configuration a lot, or when you are testing out different configurations to see what it is that you want.

A server that is loading all the modules dynamically, rather than having those modules compiled in, takes a little longer to start up, but this penalty is paid only at server start, and after that the servers run at the same speed. That is, a server running modules as DSO's does not run any slower.

2.3 Configuring

Once you've compiled and installed your server, you need to configure it for your particular environment. Many of the configuration directives got set when you ran configure (or Configure), and so the server should work correctly immediately. However, you will probably want to change some things, since the default installation is very generic, and not precisely suited to your needs.

Apache, unlike most of its competitors in the web server market, lets you configure everything, down to the smallest detail. And if there's really something that you want to configure that you can't, you have the source code, so you can change it if you are so inclined.

2.3.1 Configuration Files

The configuration for your Apache server is located in a file called httpd.conf, which is usually located at /usr/local/apache/conf/httpd.conf.

Note that if you installed Apache with a RPM (don't do that!) then the files will be in bizarre places that have no relation to logic. Uninstall the RPM, and install from source. It's a simple process, and reduces your pain in the long run.

Note: I made a comment like the above in one of my articles on ApacheToday.com, and got thoroughly chastized for it by some Red Hat fanatics that read the article. While I found their comments, and their reasons for their comments, to be rather amusing, I should emphasize that this is just my opinion, and should not be taken as some sort of transcendent truth. If you really want to spread your files all over your file system, go right ahead. You might notice, however, that a *default* installation of Apache puts everthing in /usr/local/apache, so it's a safe bet that the Apache developers agree with me on this one.

The format of httpd.conf is very simple.

There are comments, which consist of a hash sign (#) at the beginning of a line:

```
# Based upon the NCSA server configuration files originally by Rob McCool.
```

There are directives, which look like a name, followed by a value:

```
ServerAdmin webmaster@rcbowen.com
```

There are sections, or containers, which look rather like HTML tags:

```
<Directory /usr/local/apache/cgi-bin>
    AllowOverride None      </Directory>
```

Sections can contain directives, and those directives apply to the the resources defined by the container definition. In the above example, the AllowOverride directive will apply to files located in the specified directory.

You can edit these configuration files with your favorite text editor. You need to restart the server when you are done editing the configuration files in order for the new configuration to take effect.

You can use the apachectl script to test your configuration file to make sure that you did not make any errors.

```
/usr/local/apache/bin/apachectl configtest
```

More about this below.

2.3.2 Comanche

One of the battles that *nix continually has to fight is the notion that it's hard to use. Much of this notion comes from the fact that everything you want to use on *nix has a configuration file, and every configuration file has a different format. Learning all these different formats is a pain, and it's so easy to get it wrong. Sendmail is one of the worst offenders in this arena, but even something as simple as Apache gets difficult to configure. Its modular architecture means that it can be extended forever, and every extension has its own configuration directives. This can be a little overwhelming.

Daniel Lopez took on this problem as his Master's thesis, and developed Comanche - the Configuration Manager for Apache. Comanche is a graphical configuration tool, written in Tcl, which lets you configure Apache in a more intuitive interface. It tells you what each directive means, and asks you questions that make sense. Your answers are put back into the configuration files in a format that Apache can understand.

Comanche can also be used to configure other applications, such as Samba, which have text configuration flies. There is not yet a plug-in for configuring Sendmail, but this is something that Daniel is frequently asked for, so perhaps there will be some day. And you can write your own extensions to Comanche to configure anything that has a text configuration file.

Daniel now works at Covalent Technologies. Daniel is a member of the ASF, and you should attend his talks here at ApacheCon, if you have not already done so.

2.4 Starting, Stopping, Restarting

There are a variety of ways to control your Apache server. We'll focus on a script that ships with Apache, called apachectl, which does a few other things in addition to just starting, stopping, and restarting.

2.4.1 Apachectl

apachectl, which presumably stands for ``Apache control", is located in the bin directory of your Apache installation. It is a shell script which does many of the things that you'll want to do in controlling your Apache server. It can be run with any of the following arguments:

start

Starts the server.

stop

Stops the server.

restart

Restarts the server, if running, by sending a SIGHUP. If the server is not running, starts it.

fullstatus

Displays the full status of the server. Requires that mod_status is enabled, and that lynx is installed.

status

Displays a brief status report for the server. Requires that mod_status is enables, and that lynx is installed.

graceful

Does a graceful restart by sending a SIGUSR1, if the server is running. If the server is not running, it will start it. A graceful restart has the advantage over a simple restart in that child processes that are currently serving content will be permitted to complete their current connection before they are killed.

configtest

Reads the configuration file and parses it for syntax errors.

help

Displays usage information about the apachectl script.

2.4.2 Starting your Apache Server on System Restart

Linux has a process for starting processes on system startup. This consists of a directory /etc/rc.d containing scripts for each of the processes that you want to start.

If you place a file in /etc/rc.d, called rd.httpd, it will be run on server startup. rc.httpd should contain the following command:

```
/usr/local/apache/bin/apachectl start
```

If you're running Red Hat, or Mandrake, or one of the other Linuxes that looke like them, you'll find that there are a number of subdirectories of /etc/rc.d that look like rc2.d, rc3.d, and so on, which contain the startup scripts for all your various services. Actually, symlinks to them. On these systems you should create a file at /etc/rc.d/init.d/httpd, containing the command above. You should then create links to it from the directories rc3.d and rc5.d. Each of those directories corresponds to a runlevel. You'll usually be in either runlevel 3 or 5, so that's when you want to start Apache.

2.5 Integrating Apache with the Rest of Your Business

The common wisdom is that every company needs a web site, because every company needs a web site. And so lots of companies have web sites which are nothing more than a electronic sales brochure.

With more and more people online every day, many users will expect to get just as good service from your web site as they would in person, or over the phone. In fact, the expectation is often higher. After all, this is a computer. They should be able to get direct access to the answers that they need, and it should be instantaneous.

Fortunately, I'm a technical guy, not a marketing guy, so I can only advise you on the tecnical aspects, not on business practices. However, I can say from experience, that if a company's web site does not provide me with the answers I need, I'm very likely to just go somewhere else for solutions. I don't have the time or patience to try to figure out bad web site.

3. PHP

3.1 Introduction

PHP – is a scripting language for writing web applications that execute on server-side. Scripting language itself is much alike C, however it still contains many differences. Here are some basic ones:

- While C program needs to be compiled before execution, PHP script is interpreted at runtime;
- Variables do not need to be declared in PHP;
- PHP operates more gently with string variables (e.g. "5" + "6" = 11);
- C is function-oriented language, while in PHP for execution code can be written straightforward;
- All variables in PHP should have "$" prefix, otherwise identifiers are treated as string constants;
- PHP doesn't make use of standard header files and libraries – all functions are built-in, file inclusion is mostly used for user extensions and text blocks;
- PHP supports hash arrays as native type;
- PHP code should be embedded to some different file (like MS-ASP scripting, or JavaScript) and is extracted by PHP engine before execution, while C requires a complete source file;
- PHP should not execute any user query or delayed request functions; PHP normally cannot display application windows, buttons or other widgets. Its purpose is to generate content (e.g., generation of text files, images, data files, etc);

Loop statements, conditionals and comments are identical to C syntax, with exception that each variable should be prepended with dollar sign "$". PHP outputs data with "echo" command.

The essential PHP concept is to embed scripting code to HTML, using unique delimiting tags to separate raw HTML from script code. PHP scripts are executed on server side (like Perl/CGI or Java servlets), and the final user gets pure HTML in a browser.

Provided that PHP code is surrounded with *<?php ... ?>* tags, web server is able to filter such sections and puts the execution result instead.

3.2 Adding MySQL Support to PHP

This section will describe how to check and enable MySQL API support for PHP. If you are 100% sure that PHP is properly installed on your web server with MySQL support, you can safely skip all installation instructions.

3.2.1 Checking MySQL Support Availability

Before trying to run any MySQL-dependent PHP script, it is necessary to ensure that PHP on web server has such API installed.

Save this single line to *phpinfo.php* file within web server space, and load it through your browser. Note, that simply loading this file from local disk will not work – you'll have to pipe it through web server. In other words, you should make it work by loading some URL, which starts with "http://". The sample URL may look like:

http://localhost/phpinfo.php

If using correct URL you've got an empty page, or the page shows up unmodified example code, PHP is not installed to your web server, and PHP module requires installation itself.

If the information page appeared correctly, seek for "mysql" on the page. There should be a separate section called "mysql" and it will look like table 3.1:

Table 3.1 MYSQL

mysql

MySQL Support	**enabled**
Active Persistent Links	0
Active Links	0
Client API version	3.23.32

If such looking part is shown on the information page, then mysql module is enabled and is working correctly, so you can skip "Enabling…" sections of this guide.

If the information page appeared, but no "mysql" section found, then MySQL API support needs to be installed to PHP and enabled.

3.2.2 Enabling MySQL API Support for POSIX-Compatible OS

When using POSIX-compatible OS (e.g., Linux, FreeBSD, Solaris, etc), then MySQL-enabling option should be applied while compiling PHP from distribution.

> Note, that if precompiled PHP4 is included to your OS installation package, MySQL support is possibly included there by default.

If you've got to compile PHP yourself, specify *--with-mysql* option to configure script. After proper compiling and installation, MySQL support should be enabled. You can check it again by using *phpinfo* function sample (above in this guide).

3.2.3 Enabling MySQL API Support for Microsoft Windows

When running Microsoft Windows, you may run into several issues with PHP itself and MySQL. While C compiler is an essential part of most POSIX-compatible operating systems, Windows system typically doesn't contain any compiler installed. Anyway, compiling PHP under Windows is pain, if you never did this before; so consider downloading Win32-precompiled version from PHP site.

The Win32-precompiled version is available for download from official PHP site, and contains MySQL support integrated.

3.3 Using MySQL API in PHP

PHP provides a set of functions to use for accessing and manipulating data on MySQL server. The following sections will provide a step-by-step description of how to create and manage MySQL connections, work with MySQL tables, insert and remove data, etc.

3.3.1 Database Connections

Before any operations are to be made on the data, database connection should be established. Two general types of connection exist – normal connections and persistent connections.

There is no commonly used term for mentioning normal (non-persistent) connections. Thus, when connection is mentioned to be "normal", or simply not mentioned to be persistent, then it is likely non-persistent connection.
Within simple PHP scripts, single-time MySQL connections are created, and closed once script execution is completed. This is good for rare connections, when PHP page isn't requested too frequently.

The basic idea of persistent connections is to keep connection open for some particular time, and if the page loads multiple times, PHP code will reclaim the same connection.

For high traffic web sites, it would be reasonable to use persistent connections. However, if web site suffers from a very high traffic (million visits per day), for used PHP and MySQL versions the practice showed that performance degrades.

For such web sites, I would not recommend to use persistent connections. PHP developers reported, that under really heavy load, persistent connections reclaim too much of web server resources, and thus performance degrades, as the result of continuous memory swapping. However, if you are not planning to run such heavy traffic sites (e.g., like world immigration center, or Microsoft), persistent connection will act with noticeable performance boost, comparing to normal non-persistent connections.

As for scripting, persistent connection doesn't differ from usual connection, except that different function is used for connection establishment. Trying to close persistent connection will do no effect, however it's often useful to keep closing function calls in PHP code (e.g., for compatibility purposes).

3.3.2 Establishing Connection

To establish simple connection, function *mysql_connect* should be used.

```
$handle = mysql_connect    ( host [, username[, password ]] )
```

Working with persistent connection differs with only one feature – connection function name is *mysql_pconnect*. The function has the same parameter meanings and is used exactly in same way.

```
$handle = mysql_pconnect  ( host [, username[, password ]] )
```

All three parameters are of type string, and connection handle is returned once function is executed. For example, if connection to server *"cassy"* is desired, the line from PHP script may look like this:

```
$link = mysql_connect ("cassy", "george", "greatpassword1105");
```

In the upper example, user name is *"george"* and the password is *"greatpassword1105"*.

If name and password are not specified, the PHP process owner's user ID is taken, and empty password is assumed. If host name is not specified, *"localhost"* will apply.
You will not be able to specify user name, if *sql.safe_mode* option is set in *php.ini* file. In this case, default user will be used instead.

3.3.3 Selecting Database

After connection is established, the default database should be selected to use for SQL queries, which don't specify database name explicitly. The function *mysql_select_db* is used for such purposes.

```
mysql_select_db( database_name [, connection_handle] )
```

The very first parameter specifies name of database to select within connected server space; *connection_handle* – is the variable, resulted earlier from *mysql_connect* function. PHP manual describes, that *connection_handle* can be omitted, if last opened handle is to be used, however this can lead to confusion, when multiple connections are used in the script. So I recommend to specify *connection_handle* explicitly, when more than one connection is planned (e.g., if you are writing some kind of abstraction layer or PHP database engine).

Function *mysql_select_db* returns either TRUE on successful database selection, or FALSE on error (e.g., database not found, connection handle is bad, lightning hit to the server, etc).

3.3.4 Closing Connection

Finally, after connection is not needed anymore, *mysql_close* call should be used to close connection, as you probably guessed yourself from the example in previous section.

```
mysql_close ( connection_handle )
```

Nothing wrong happens when trying to close persistent connections – the function will simply perform no operation, however to preserve compatibility, even for persistent connections I would recommend keeping *mysql_close* call anyway (what if in the future it would be desired to change to simple connections?).

PHP documentation mentions, that it is not necessary to use *mysql_close* at all, however the practice showed, that opposite to manual, unclosed orphan connections are kept in memory, and are closed only in some time (after timeout expires). This is, of course, a resource black hole for web sites under heavy load.

3.4 Obtaining Information by Connection Handle

Starting with PHP version 4.0.5, few *mysql_get_xxxx_info* functions were introduced. These functions will help to obtain some basic information about MySQL API and connection handles:

```
$text = mysql_get_client_info      ( )
$text = mysql_get_server_info      ( connection_handle )
$text = mysql_get_host_info        ( connection_handle )
$val  = mysql_get_proto_info       ( connection_handle )
```

Function *mysql_get_client_info* will return a string, containing current PHP-MySQL client library version. Note, that this version number is not related to MySQL version installed on server.

Either internal PHP-MySQL API library, or MySQL-provided library can be used during PHP compilation. If path to MySQL libraries is not specified during PHP compilation, PHP uses built-in MySQL client library, which typically is older than it could be otherwise. The value of *mysql_get_client_info* will look like this:

3.4.1 3.23.32

Function *mysql_get_server_info* needs *connection_handle* parameter; it queries MySQL server version, using the server connected via *connection_handle*. The return value will be formatted exactly in the same way, as in *mysql_get_client_info* function, however these two values are not related to each other, and *mysql_get_server_info* will typically return different value. For example, on my machine it indicates:

3.4.2 3.23.37

Functions *mysql_get_host_info* and *mysql_get_proto_info* are used to get more information on currently connected host and protocol.

The example below demonstrates usage of *mysql_get_xxxx_info* functions:

```
<html><pre>
<?
    $hd = mysql_connect("192.168.1.2","root","")
     or die("Can not connect");
```

3.5 Executing Queries

To fetch or alter data, SQL queries are to be executed. "SELECT" queries will obviously have result, and the description of how to deal with rowsets is provided later in this guide. Less obvious is, that "DELETE", "INSERT" and "UPDATE" queries will have result as well, however the result will contain no rowset, but different miscellaneous information, such as number of rows affected, etc. This section will explain how to execute queries and how to reclaim those "unobvious" results.

3.5.1 Executing Raw SQL

MySQL API for PHP contains function named *mysql_query*, which takes query string and connection handle as parameters. This function either returns rowset handle, miscellaneous info, or zero value (in case of errors). PHP itself doesn't differ rowset handles from miscellaneous info handles, and in terms of PHP returned value is called "resource handle"; however, different functions are used to fetch data from rowsets, than to get number of affected rows from miscellaneous handle, etc.

```
$res = mysql_query( SQL_query_string [, connection_handle] )
```

If *connection_handle* parameter is omitted, then the last connection is used. If you plan to run queries on multiple connections (e.g., copying data from one database server to another, or synchronizing two database servers), omitting *connection_handle* parameter can lead to confusion, so if you plan to handle two connections in the same PHP script, specify *connection_handle* parameters explicitly.

The *SQL_query_string* parameter specifies SQL query string to execute. Note, that no semicolon should be put to the end of string.

MySQL API for PHP contains function *mysql_affected_rows*, used to obtain number of rows affected by the query operation. Note, that this one doesn't work for SELECT statement queries – it works for data-modifying queries only; there's a different function to obtain number of rows attached to rowset handles (described later in this guide).

```
$nrows = mysql_affected_rows ( result_handle )
```

This function receives *result_handle*, returned by *mysql_query* function; *$nrows* – is an integer variable assigned to affected rows count.

Note, that when used with UPDATE statement queries, *mysql_affected_rows* can actually return different number of rows – those records, which already contained desired values would not be counted.

3.5.2 Formatting Data for Queries

In previous section, we went through description for raw SQL query execution. This would be enough, when all data for query is constant, pre-formatted and already prepared for execution. However, dynamically generated query strings may cause problems, if formatted incorrectly.

3.5.3 Working with Rowsets

It was mentioned earlier in this guide, that SELECT statement query returns some special kind of result – rowset handle. In this section, I will describe the common ways to pull actual data from rowset handles and how to deal with such rowset handles in some other aspects.

3.5.4 Buffered Queries

SELECT statement queries are executed like all other queries – with *mysql_query* function:

```
$res = mysql_query("SELECT xxxx FROM yyyy WHERE zzzz",
$connection_handle);
```

After such command execution, *$res* variable will contain rowset handle, to which buffered rowset is attached. By function *mysql_query*, the complete rowset is received from SQL server and stored in memory.

Normally, rows are extracted in sequential order, as further rows may not be ready, like in case of unbuffered queries (described in next section). Function *mysql_query* fully buffers received data, so rows can be accessed in random order. You can seek to some particular row by using *mysql_data_seek* function:

```
mysql_data_seek ( rowset_handle , row_number )
```

The first parameter is actually the result (e.g., *$res* variable) returned by *mysql_query* function. The second parameter means the destination row number, starting with 0.

Function returns logical value. *TRUE* will be returned, if row was successfully found and positioned, *FALSE* otherwise. All *mysql_fetch_xxxx* functions will start fetching from positioned row, or by default (if row pointer wasn't changed) from the beginning of rowset.

Total number of rows in rowset can be obtained by *mysql_num_rows* function:

```
$nrows = mysql_num_rows ( rowset_handle )
```

As *mysql_query* buffers all data received from MySQL, total number of rows can be queried simply by specifying *rowset_handle*, returned from earlier call to *mysql_query*.

Function *mysql_num_fields* queries number of fields per row:

```
$nfields = mysql_num_fields ( rowset_handle )
```

Function *mysql_num_fields* can be useful while retrieving information about table columns.

Those two functions query the dimensions of rowset, so by multiplying *$nfields* by *$nrows*, we get the total number of fields in rowset. This can be used in data size estimation for tables of equally-typed columns.

3.5.5 Un-buffered Queries

While *mysql_query* waits for the query to complete and buffers the result, *mysql_unbuffered_query* function returns as soon as query is passed to SQL server.

Sometimes, database size does not allow storing all queried results in memory (e.g., database is about 14GB), thus in these cases, for performance issues it's preferable to use *mysql _unbuffered_query* function instead of *mysql_query*.

```
$res = mysql_unbuffered_query ( query_string [,
                    connection_handle ] )
```

Un-buffered queries are especially useful when not each queried value is needed (like search engines).

3.5.6 Fetching Rows from Row sets

The essential way to extract data row, is to use *mysql_fetch_row* function:

```
$array = mysql_fetch_row( rowset_handle )
```

After such execution, $array will contain data row, which can be accessed by numeric index, starting with 0. E.g., if table contains eight fields: *customer_id, title, fname, lname, addressline, town, zipcode, phone* exactly in this order, *$array[0]* is assigned to value of field *customer_id*, *$array[1]* correspondingly to *fname* value, and so on.

3.5.7 Querying Information about Columns in Table

In some cases, it's necessary to execute queries on unknown tables (e.g., user-entered queries). For example, storing property values of complex structures sometimes will require different tables to be used with the same code. Thus it would be necessary to query information about table columns before actual data fetching. Earlier in this guide I described how to get number of rows in row set and how to seek pointer to a particular row. Different PHP function is used to operate with column pointer:

```
mysql_field_seek ( rowset_handle , column_offset )
```

The second parameter *column_offset* specifies default offset for column pointer. Field info fetching function can explicitly specify different field offset, but if not specified, the default pointer value will be used.

To fetch information of sequential column in a rowset, function *mysql_fetch_field* is used.

```
$object = mysql_fetch_field( rowset_handle [, field_offset ] )
```

The object value is returned. You can obtain the following object properties:

$object→name ..... the column/field name

$object→table ..... name of table, to which row set belongs

$object→max_length ..... maximum value length

$object→def ..... default field value

$object→not_null ..... 1 if column is forced to be not NULL, 0 otherwise

$object→primary_key ..... 1 if column is primary key for the table, 0 otherwise

$object→unique_key ..... 1 if column is unique key for the table, 0 otherwise

$object→multiple_key ..... 1 if column is non-unique key, 0 otherwise

$object→numeric ..... 1 if field is numeric, 0 otherwise

$object→blob ..... 1 if field type is BLOB, 0 otherwise

$object→type ..... the type of column/field

$object→unsigned ..... 1 if value stored in unsigned format, 0 otherwise

$object→zerofill ..... 1 if column is zero-filled, 0 otherwise

Of course, *mysql_fetch_field* function can't get any particular field value – it operates on column basis, not referring any particular row. This function affects neither row pointer, nor rowset data, but only gets basic information about table used in SELECT query, not depending on its contents.

For convenience, there are some PHP functions to obtain single property of a field:

```
$tablename   = mysql_field_table ( rowset_handle, field_offset)
$typestring  = mysql_field_type ( rowset_handle, field_offset)
$fieldlength = mysql_field_len ( rowset_handle, field_offset)
$sqlflags    = mysql_field_flags ( rowset_handle, field_offset)
```

All four functions receive *rowset_handle* and *field_offset* in parameters.

Note, that purpose of passing *field_offset* to *mysql_field_table* function is not obvious. Normally, all fields in a row set belong to the same table, however for some complex merged rowsets it's possible that single row will contain fields, which belong to different tables. Of course this is up to a particular PHP application, and if you don't ever plan to use merged rowsets, you can safely pass 0 as the second parameter for *mysql_field_table*.

3.5.8 Freeing Rowsets

All rowsets are automatically freed upon script execution completes. However, if PHP script is intended to receive some huge data arrays, and when especially doing this in cycle, memory space will be taken and not returned back until script completion.

Normally, if script contains only one call to query, freeing rowset is not necessary, however this should be done, if query resulting rowset is not necessary anymore before another query is about to start:

```
mysql_free_result ( rowset_handle )
```

Notice, that first parameter is a rowset handle, not result handle. This function should be used to free rowsets, obtained by SELECT statement queries, and should NOT be used for other types of query.

3.6 Error Handling

PHP provides two general functions for getting status information on the last MySQL API function execution. You can either get error number with *mysql_errno* function, or the full message text with *mysql_error*.

```
$errNr = mysql_errno()
$errtext = mysql_error()
```

Function *mysql_errno()* provides numerical status code of the last executed MySQL operation. If operation was successful, then zero value is returned; otherwise function will return error code, generated by MySQL.

Function *mysql_error()* returns error message string. If actual meaning of error is not important for script functionality (e.g. script simply quits on any error occurrence), while you still want to show the error message up to user, *mysql_error* function will be useful.

3.7 PEAR

PHP community made an attempt to create open source code repository (like CPAN is for Perl). As the result, PEAR appeared.

PEAR stands for PHP Extension and Application Repository, and represents a large collection of object oriented open source PHP classes.

For now, PEAR isn't documented well. Instead, some of information can be found on PHP official web site, and the other part – as comments in PEAR source code.

To use PEAR, you must have some knowledge of object oriented programming in relation to PHP. PHP object oriented programming techniques are described here:

http://www.php.net/manual/en/language.oop.php

Next sections will briefly describe how to perform database connections using PEAR, how to execute queries, fetch results and handle erroneous situations.

3.7.1 Getting PEAR to Work

Currently, PEAR is a part of PHP distribution. PEAR sources can be found under the directory *pear*, relative to PHP installation.

To enable PEAR functionality, PHP include path variable should be adjusted to include PEAR directory as well. For this, open *php.ini* file in any text editor and seek for *"include_path"* variable. Normally on fresh installations this variable will contain no value, so you just simply have to put full path to PEAR there. Otherwise, if some paths are already used, you can add PEAR's directory to the end of list. Place a colon to separate multiple path entries from each other.

In some cases you will need current directory to present in path (for including local files), so add `.' as additional path entry – this will allow inclusion of files from current script directory.

3.7.2 PEAR's Database Abstraction Interfaces

PEAR's DB abstraction class allows easy manipulation with databases, not depending on server type. No matter whether it would be MySQL or PostgreSQL – to port code from one database to another generally less lines of code will be changed, as opposed to full database-manipulation code reimplementation, when using traditional approach.

PEAR interface also adds convenient features to work with multiple results and advanced error handling, by providing corresponding classes and objects. In basic concept, PEAR's DB abstraction idea is much similar to Perl's DBI module.

The code flexibility is gained at a cost of performance. Typically software developer's time is more expensive than machine time, however some software solutions may require better performance, than PEAR can provide. And if application is planned to use MySQL-server databases only, it could be reasonable to use standard MySQL API, which provides generally better performance and is typically easier to use in simple projects.

The "Error Handling" section supplied basics of error handling and reporting.

Sequential guide part was dedicated to PEAR Database Abstraction interface. While PEAR is not widely used yet, some users may find it to be useful for portable database projects, such as web engines, etc.

Finally, this chapter ended with "Summary" part, which described in briefly, that in this chapter, the complete information… …this is possibly what you have read already.

4. MY SQL

4.1 Introduction

Since before the dawn of the computer age, people have been using databases. Before computers, a database may have been a Rolodex containing phone numbers of the important people you knew, or it was a filing cabinet that contained all the personnel records for the company. Today, databases are computer-based and are found virtually everywhere. From desktop databases of your record collection to Web-enabled databases that run large corporations. If MySQL is so good, why hasn't it already caught the attention of the industry? The answer is that until 1999, Linux and the Open Source movement were practically unknown. MySQL runs primarily on UNIX-based systems—though there are ports for almost every platform on the market. Until the Open Source movement and the availability of UNIX-based operating systems at affordable prices, no one really looked at MySQL as a contender.

Because of the recent success of Linux, MySQL has grown in popularity. Unfortunately, there is not much out there in the form of documentation.

MySQL, pronounced "my Ess Que El," is an open source, Enterprise-level, multi-threaded, relational database management system. That sounds like a lot of sales or marketing hype, but it truly defines MySQL.

You may not be familiar with some of these terms but, by the end of today, you will be. MySQL was developed by a consulting firm in Sweden called TcX. They were in need of a database system that was extremely fast and flexible. Unfortunately (or fortunately, depending on your point of view), they could not find anything on the market that could do what they wanted. So, they created MySQL, which is loosely based on another database management system called mSQL. The product they created is fast, reliable, and extremely flexible. It is used in many places throughout the world. Universities, Internet service providers and nonprofit organizations are the main users of MySQL, mainly because of its price (it is mostly free). Lately, however, it has begun to permeate the business world as a reliable and fast database system.

The reason for the growth of MySQL's popularity is the advent of the Open Source Movement in the computer industry. The Open Source Movement, in case you haven't heard about it, is the result of several computer software vendors providing not only a product but the source code as well. This allows consumers to see how their program operates and modify it where they see fit. This, and the popularity of Linux, has given rise the use of open source products in the business world. Because of Linux's skyrocketing popularity, users are looking for products that will run on this platform. MySQL is one of those products. MySQL is often confused with SQL, the structured query language developed by IBM. It is not a form of this language but a database system that uses SQL to manipulate, create, and show data. MySQL is a program that manages databases, much like Microsoft's Excel manages spreadsheets. SQL is a programming language that is used by MySQL to accomplish tasks within a database, just as Excel uses VBA (Visual Basic for Applications) to handle tasks with spreadsheets and workbooks. Other programs that manage databases include Microsoft's SQL Server, Sybase Adaptive Server, and DB2. Now that you know where MySQL came from, look at what it is. To begin with, start with the term database. What is a database? You have probably used one in your lifetime. If you've ever bought anything over the Internet or have a driver's license, you can be assured that you have used one. A *database* is a series of structured files on a computer that are organized in a highly efficient manner.

These files can store tons of information that can be manipulated and called on when needed. A database is organized in the following hierarchical manner, from the top down. You start with a database that contains a number of tables. Each table is made up of a series of columns. Data is stored in rows, and the place where each row intersects a column is known as a field. Figure 1.1 depicts this breakdown. For example, at your favorite online book store there is a database. This database is made up of many tables. Each table contains specific, common data. These tables are made up of named columns that tell what data is contained in them. When a record is inserted into a table, a row of data has been created. Where a row and a column intersect, a field is created. This how databases are broken down.

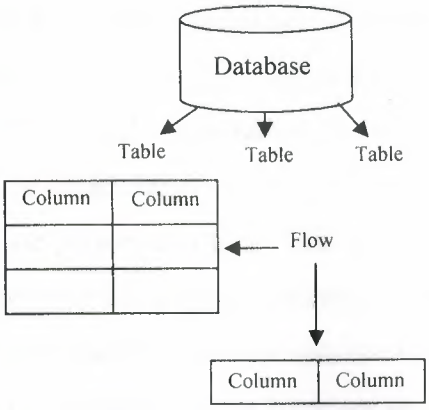

Figure 1.1 The anatomy of a database.

MySQL is more than just a database. It is a system that manages databases. It controls who can use them and how they are manipulated. It logs actions and runs continuously in the background. This is different from what you may be used to. Most people think about Microsoft Access or Lotus Approach when they think about databases. These are databases, but they are not management systems. A DBMS can contain many databases. Users connect to the database server and issue requests. The database server queries its databases and returns the requests to the issuers. Databases, such as Approach and Access, are a step down from this type of system. They share their files with multiple users, but there is no interface controlling the connections or answering requests.

There are many uses for a DBMS such as MySQL. Uses can range from help desk systems to Web site applications. The important thing to remember is that MySQL is large enough and quick enough to function in almost any situation. Where it finds itself most comfortable is the Enterprise.

1.2 What Is the Enterprise?

The Enterprise I'm referring to is not a starship or a space shuttle. The *Enterprise* is the area in the business world where many large systems interact with one another to accomplish a common goal. Some applications that are at this level of business include SAP, Microsoft SQL Server, Oracle 8i, and Sybase Adaptive Server.

The computer applications that exist at this level of business tend to have certain characteristics. They are usually multiuser in nature—many people can use the same application at the same time. Another characteristic is that they provide some sort of security mechanism. The final characteristic is that applications at this level have to be very flexible. The first characteristic of an Enterprise-level application is that it can be used by more than one person at a time. This is a requirement at this level of business. More than one person may need to have access to business information at a given time. This is critical for the business to function successfully.

MySQL meets this requirement. It can have up to 101 simultaneous connections. This doesn't mean that only 101 people can use this application. It means it can have 101 connections going on at the same time—which is a little different. A connection is the time it takes for a user to receive the data that he or she has requested. In the case of MySQL, this is hardly any time at all. Most database systems in the same class as MySQL allow fewer simultaneous connections. Currently, the only DBMS to offer more connections is Microsoft SQL Server.

The next characteristic that an Enterprise-level application must have is security. When dealing with mission-critical information, only people with the need to know should be allowed to view it. Security keeps malicious people at bay; without it, disasters can happen. MySQL meets this requirement. The security in MySQL is unparalleled. Access to a MySQL database can be determined from the remote machine that can control which user can view a table. The database can be locked down even further by having the operating system play a role in security as well. Very few databases in the same class as MySQL can compare to the level of security that MySQL provides.

One other characteristic of an Enterprise-level application is flexibility. How flexible is the application? Can it change to meet the ever-changing needs of business? How deep can you make those changes? How hard is it to change? MySQL answers these questions very well. It is extremely flexible and easy to use. MySQL can run on almost any platform. If a new CIO wants to change from Windows NT to Linux, fine—MySQL can adapt. MySQL also comes with the source code. If there are any deep-level changes that you need to make, you can edit the source and make these changes yourself. If MySQL is missing a feature that you can't live without, just add it yourself. No other database on the

market can offer you that kind of flexibility. MySQL also has several application-level interfaces in a variety of languages. If yours is mainly a Microsoft shop, you can use ODBC to interact with MySQL. If your company is a UNIX shop, you can use C, Perl, or JDBC. There is no end to the flexibility that MySQL has to offer. In addition to the previously discussed characteristics, databases at the Enterprise level must be able to work together. Data warehousing is a technique that combines all the data in a business. Because of the flexibility and speed that MySQL has to offer, it can work well in any situation.

The Internet has also become a piece of the Enterprise pie. No large corporation is without an Internet presence. These corporations need databases to sell and compete at this level of business. MySQL works well as an Internet-based database server. It has been proven in this arena and is the preferred database of many Internet service providers. Because of its speed and multiple application interfaces, MySQL is an ideal choice.

Enterprise applications are the crucial component to a business's decision-making power. Information must be timely and accurate for a business to perform effectively. To do this, applications must work quickly. An application is much like a car. It can look pretty on the outside, but the engine is what gives it its power. The same applies to an application; If its database engine is weak, so is the application. MySQL is clearly the choice for the Enterprise.

4.3 What Is a Relational Database?

A *relational database*, simply defined, is a database that is made up of tables and columns that relate to one another. These relationships are based on a key value that is contained in a column. For example, you could have a table called Orders that contains all the information that is required to process an order, such as the order number, date the item was ordered, and the date the item was shipped. You could also have a table called Customers that contains all the data that pertains to customers, such as a name and address. These two tables could be related to each other.

The relational database model was developed by E.F. Codd back in the early 1970s. He proposed that a database should consist of data stored in columns and tables that could be

related to each other. This kind of thinking was very different from the hierarchical file system that was used at the time. His thinking truly revolutionized the way databases are created and used.

A relational database is very intuitive. It mimics the way people think. People tend to group similar objects together and break down complex objects into simpler ones. Relational databases are true to this nature. Because they mimic the way you think, they are easy to use and learn. In later days, you will discover how easy a relational database is to design and learn.

Most modern databases use a relational model to accomplish their tasks. MySQL is no different. It truly conforms to the relational model. This further adds to the ease of use of MySQL.

4.4 The Client/Server Paradigm

The client/server paradigm or model has been around a lot longer than most people think. If you look back to the early days of programming, you remember or have heard or read about the large mainframe computer with many smaller "dumb" terminals. These terminals were called dumb for a reason. No logic or processing was done at the terminals. They were just receptacles for the output of the mainframe. This was the dawn of the client/server age, but the term client/server wasn't the buzzword it is today.

As the personal computer became more prevalent, giving rise to the local area network (LAN), the client/server model evolved. Now processing could be done at the client. Clients started sharing data.

This data was stored in sharable computers called file servers. Now, instead of all the processing being done at the server, it was all being done at the client. The server or centralized computer was just a large storage device. It did little or no processing—a complete reversal of earlier thinking.

After a couple of years, desktop applications became more powerful. People needed to share more information more quickly. This gave rise to the more powerful server machines. These machines answered requests from clients and processed them. These servers are what you know today as database servers, Web servers, and file servers. This is when people started calling it client/server computing. It is basically a two-tier design;

a client issues requests, and a server answers them. All the business logic is at the application level on the client. Two-tier design is still very prevalent today. This is also known as a *fat client* because all the application processing is done at the client level. After a couple of years, servers became the powerhouses of business organizations because of their duties. They were usually top-of-the-line systems with the best hardware and were tweaked for speed.

So, it was just a matter of time before someone came up with the idea of moving the guts of their programs to the server. The client would just be a graphical user interface (GUI) and the main application or business logic would be processed on the server. The server would then make the necessary calls to other servers, such as database servers or file servers, as needed. This gave birth to the three-tier or *thin client* design. In this design, all processing of the business logic is done at the server level. This allows the more powerful machine to handle the logic and the slower machines to display the output. Does this sound familiar? It should—we've come full circle. The heavy processing is again done on the more powerful, centralized machines, while all the client machines do is display the output.

The Internet is a prime example of thin client architecture. A very thin client—the browser—sends requests to a Web server, which sends a response back to the browser. The browser then displays the requested information—completely full circle.

Again, we are on the verge of a new era in computing. Applications are becoming more balanced across the network. Because of a decline in computer prices, very good machines are showing up on the desktop as clients. This allows applications to pick up the slack and perform some processing.

Server applications are becoming more advanced as well. You can now run functions remotely and accomplish distributed computing fairly easily. These advancements allow your applications to be more robust in nature and more useful to your business.

Distributed computing allows client programs to interact with multiple server processes, which, in turn, can interact with other servers. The server components can be spread across the resources of the network.

MySQL fits in very well in all these architectures. It performs extremely well in two-tier or three-tier architecture. It can also perform very well on its own.

4.5 Features of MySQL

MySQL is a full-featured relational database management system. It is very stable and has proven itself over time. MySQL has been in production for over 10 years. MySQL is a multithreaded server. *Multithreaded* means that every time someone establishes a connection with the server, the server program creates a thread or process to handle that client's requests. This makes for an extremely fast server. In effect, every client who connects to a MySQL server gets his or her own thread.

MySQL is also fully ANSI SQL92-compliant. It adheres to all the standards set forth by the American National Standards Institute. The developers at TcX take these standards seriously and have carefully adhered to them.

ANSI SQL92 is a set of standards for the Structured Query Language that was agreed on in 1992 by the American National Standards Institute.

Another valuable feature of MySQL is its online help system. All commands for MySQL are given at a command prompt. To see which arguments the commands take or what the utility or command does, all you have to do is type the command and include the -help or switch. This will display a slew of information about the command.

Yet another feature of MySQL is its portability—it has been ported to almost every platform. This means that you don't have to change your main platform to take advantage of MySQL. And if you do want to switch, there is probably a MySQL port for your new platform.

MySQL also has many different application programming interfaces (APIs). They include APIs for Perl, TCL, Python, C/C++, Java (JDBC), and ODBC. So no matter what your company's expertise is, MySQL has a way for you to access it.

MySQL is also very cheap. For an unlicensed, full version of MySQL, the cost is nothing. To license your copy will currently cost you $200. This is an incredible deal, considering what you are getting for your money. Database systems that provide half the features that MySQL has can cost tens of thousands of dollars. MySQL can do what they do better and for less.

5. CREATING DATABASE

5.1 Overview

Creating a database is probably one of the most important, yet least used, of all the MySQL functions. There are many ways to accomplish this task in MySQL. This chapter explains the following:

The CREATE and DROP commands

Using the mysqladmin utility

Adding users to your database

Creating the Meet-A-Geek database

5.2 The CREATE and DROP Commands

When you think of the CREATE and DROP commands, you should envision earthmoving equipment, dump trucks, and cranes, because these are the tools you use to create your database. These commands, though seldom used, are the most important. Hopefully, a lot of thought has gone into the decision making process before either of these commands is issued.

5.2.1 The CREATE Command

There are many different ways to create databases in MySQL. When you create a database, you usually will have the entire layout ready. Normally, you would add the tables immediately after creating the database, but, because this book is a training guide, you will take it one step at a time.

The first way to create a database in MySQL is to enter the SQL (Structured Query Language) command CREATE DATABASE>*databasename* in the MySQL monitor, where *databasename* is the name of the database you are creating. Perform the following steps to create this sample database: The process of creating a database is the same for most operating systems. When something cannot be done in a particular operating system, I will make note of that fact. You should have changed your root password for the MySQL database system. To use the mysqladmin command and to start the mysql

monitor, you will need to enter this password. For the sake of brevity, I have left that argument (-p) off my commands.

1. Open a terminal.

2. Change the directory to the mysql directory. If you created a symbolic link, you can enter

3. cd mysql If you did not create a symbolic link, you will have to enter the full path, as shown in the following:

cd /usr/local/mysql (assuming MySQL was installed to this default directory).

Note Symbolic links are generally used as shortcuts. They can take a long path name and condense it into one word, making it convenient for the user to use.

4. Ensure the mysqld daemon is running. To do this, enter the following:

5. bin/mysqladmin ping

6. After you are sure the monitor is running, start the mysql monitor by entering the following from the command line:

6. bin/mysql

8. At the monitor prompt, type the following:

9. CREATE DATABASE sample_db;

Be sure to type it exactly as it appears. Remember that it is necessary to end the line with a semicolon or a \g.

Your results should be similar to those in Figure 5.1.

Figure 5.1 Results of a Successful Database Creation.

The mysql monitor is not case sensitive when it comes to SQL commands. Thus, the following commands are all the same:

Create Database sample_db;

CrEaTe DaTaBaSe sample_db;

create database sample_db;

These commands will all create the same database named sample_db. It is a popular convention to capitalize all SQL commands—this book will follow that convention. An important point to remember is that capitalization does matter when it comes to objects within your database. For example, sample_db is not the same as Sample_DB.

After your database has been successfully created, you can begin to use it. "What Is MySQL?," the command to do this is USE. To use the sample_db, type the following from the MySQL monitor prompt: USE sample_db;

The results of your command should resemble Figure 5.2.

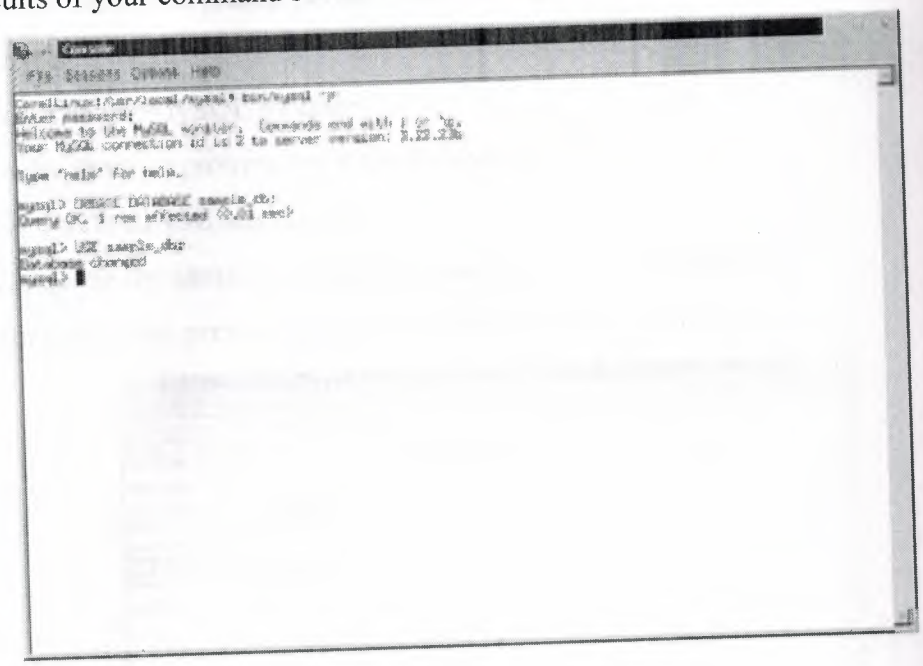

Figure 5.2 Using the New Database.

An important point to remember is that MySQL does not automatically make the database you just created the active database. You must implicitly state which database to activate with a USE statement.

5.2.2 The DROP Command

The DROP command is similar to the CREATE command. Where the latter creates a database, the former deletes one. A word of caution, the SQL DROP command is very unforgiving. There are no confirmation boxes asking if you are sure. The DROP command just deletes the database and all the data contained in it.

This shows some of the power of SQL commands. Once a command has been committed, there is no going back. (This is not entirely true—you can get your data back from a log file.) Use extreme caution when using the DROP command.

To use the DROP command, complete the following steps:

1. Make sure that the *mysqld* daemon is running and that you are in the *mysql* directory.
2. From the command prompt, type
3. bin/mysql

This will start the MySQL monitor.

4. From the monitor prompt, enter the following:
5. DROP DATABASE sample_db;

This will delete the sample_db database and ALL the data within it.

The output from the previous steps should look similar to Figure 6.3.

Figure 5.3 Dropping a Database.

mysqladmin

Like many things in the computer world, there is more than one way to accomplish a task in MySQL. MySQL offers a powerful utility that can help with the creating and dropping of a database—mysqladmin. This utility also provides many other useful functions; you will learn about some of those functions in later lessons. For now, you will create and drop a database using this utility. Creating a database with mysqladmin is very simple. To create the sample database do the following:

1. Make sure the mysqld daemon is running and that you are in the mysql directory.
2. Type the following command to create the sample database:
3. bin/mysqladmin –p CREATE sample_db

Your output should look like Figure 5.4.

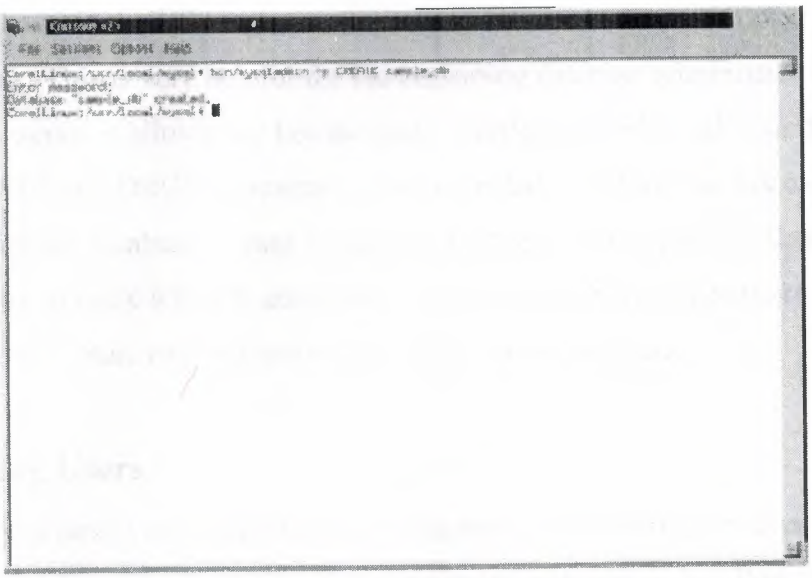

Figure 5.4 Creating a Database using ***mysqladmin***.

Dropping a database is just as easy. To delete the sample database, do the following:

1. Again, make sure the mysqld daemon is running and that you are in the mysql directory.
2. Enter the following command to *DROP* the database:
3. bin/mysqladmin –p DROP sample_db

Your output should resemble that shown in Figure 5.5.

Figure 5.5 Dropping a Database using ***mysqladmin***.

You may have noticed that when using mysqladmin, you are prompted before deleting the database. This is very helpful for the beginning database administrator, as well as the seasoned veteran. It allows one last moment of reflection before all your data is lost.

The CREATE and DROP arguments of the mysqladmin utility are not case sensitive, but the name of the database is case sensitive. Another notable point is that you must have the authority to use CREATE and DROP. As root, you have this authority, but if you are not an administrator, you will not be able to use these commands.

5.3 Adding Users

Now that you have your database up and running, you should give other users the ability to use the database. Today, you will learn how to add users; To allow a user from your local machine—referred to hereafter as *localhost*—to gain access to your database, the user must exist in several places. The MySQL RDBMS contains a database named mysql. This database holds all the permissions for all MySQL databases. This database consists of the following tables: User The table that holds all the names, passwords, hosts, and privileges of all the users of this MySQL RDBMS *db.* The table that contains all the users, databases, and hostnames for this MySQL RDBMS. *host* The table that contains all hostnames, databases, and privileges they hold for this MySQL RDBMS For

a person to use your database, the hostname of the machine from which he or she will be connecting must exist in the host table. The user must exist in the user table, and the database must exist in the db table. Complete the following steps to give another user the ability to use your database from the local machine.

1. First, make sure the daemon is running and that you are currently in the mysql directory.

2. Add the hostname and database to the host table. To do this, you must use the MySQL monitor.

3. bin/mysql –p

4. Next, you must make the mysql database the active database. To do this, type the following:

5. USE mysql;

Note Remember, commands are not case sensitive, but the database objects are.

6. To add the hostname/database combination to this MySQL RDBMS, you must use an SQL INSERT command. Type the following from the command line:

6. INSERT INTO mysql VALUES('localhost','sample_db',

8. 'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');

Remember that if you do not type a \g, the MySQL monitor will continue your statement on the following line. This is helpful because it allows for easily readable commands, and, if you make a mistake, you can use the history key to bring it back. Your output should look like that in Figure 5.6.

Figure 5.6 Adding a Host to the Host Table.

The next step is to make sure you have users to add to your database. You will add a user now.

```
INSERT INTO user VALUES('localhost','TestUser',
PASSWORD('pass123'),'Y','Y','Y','Y','Y','Y','Y' ,'Y','Y','Y','Y','Y','Y','Y')
```

The PASSWORD function is an *intrinsic function*, that is, a function that can be called from within MySQL. The password function takes a string as an argument and encrypts it. This encrypted word is stored in the database. This prevents prying eyes from easily discovering the passwords of all your users with a simple query to the mysql database. It's best to get in the habit of adding users in this manner. You are now ready to add your database and users to the mysql database. To do this, enter the following:

```
INSERT INTO db VALUES('localhost','sample_db',
'TestUser','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y', 'Y')
```

Let's review what you have done. To allow a person to use the sample_db database from the local machine, several things must be in place. You will need the hostname of the computer the user will be using to connect to your database. In the example, you are going to use the same machine that has the MySQL RDBMS installed. Your machine may have a really cool name, but MySQL only requires the name 'localhost' to describe a local machine. If you were connecting to another mysql database from your machine, your machine's name would have to be in that database. The second thing that needs to be in place is a user. You can add users at any time. Because I'm assuming that you have a fresh installation, I went through the process of adding a user. After the user is added, you could go ahead and give this user permission to use your database. You did this by adding the user to the db table.

5.4 Creating the Meet_A_Geek Database

You will create the Meet_A_Geek database using the mysqladmin utility. (You will add users in a later lesson.) You will use this database as an example throughout the book, building upon it in each lesson. To create the database, do the following:

1. Make sure the daemon is active and that you are in the mysql directory.
2. To create the database, enter the following:
3. bin/mysqladmin –p CREATE Meet_A_Geek

6. ADDING TABLES, COLUMNS, AND INDEXES TO YOUR DATABASE

6.1 Overview

Aside from creating the database, adding columns, tables, and indexes are the most important steps in the database creation process. Tables and their columns are what define a database. This chapter presents the following:

- How to create tables and columns
- How to edit existing columns
- What an index is and how it is used

6.2 Creating Tables

Creating in MySQL is a relatively easy task. Like so many other things, there is more than one way to perform this task. This chapter will cover two ways to add the tables you came up with in your design session. First, you will use the MySQL monitor. The monitor is the primary tool to use when interacting with your database. To create your first table, perform the following steps:

1. Ensure that the mysqld daemon is running (using mysqladmin ping) and that you are in the mysql directory. (pwd should return /usr/local/mysql, assuming you installed mysql in the default directory.)
2. Start the MySQL monitor by typing the following:
3. bin/mysql –u root –p Meet_A_Geek

You should be prompted for a password. After you enter your password, you will enter the MySQL monitor with the Meet_A_Geek database as the active database (see Figure 6.1).

Figure 6.1 Starting MySQL with an Active Database.

You will create the Customers table. To do this, enter the following commands exactly as they appear. Remember that pressing the Enter key does not execute the command unless the command ends with a semicolon or a \g.

```
CREATE TABLE Customers (Customer_ID INT NOT NULL
PRIMARY KEY AUTO_INCREMENT, First_Name VARCHAR(20)
NOT NULL, Last_Name VARCHAR(30) NOT NULL,
Address VARCHAR(50), City VARCHAR(20),
State VARCHAR(2), Zip VARCHAR(20),
E_Mail VARCHAR(20), Age INT, Race VARCHAR(20),
Gender ENUM('M', 'F') DEFAULT 'F',
Eye_Color VARCHAR(10), Hair_Color VARCHAR(10),
Favorite_Activity ENUM('Programming', 'Eating',
'Biking', 'Running', 'None') DEFAULT 'None',
Favorite_Movie VARCHAR(50),Occupation VARCHAR(30)
, Smoker CHAR(0));
```

Your output should like Figure 6.2.

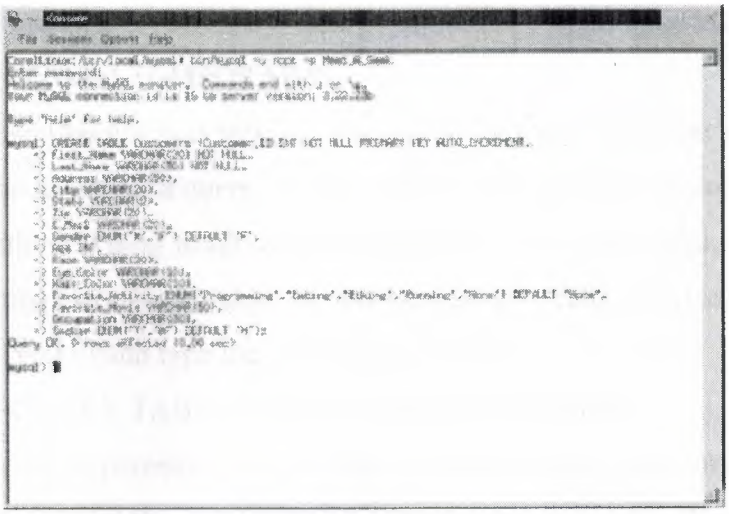

Figure 6.2 Creating a New Table for the ***Meet_A_Geek*** Database.

1. To verify your actions, type the following:
2. SHOW TABLES FROM Meet_A_GEEK; You should see a list of tables available in the Meet_A_Geek database. If you've been following along in the project, there should only be the Customers table.
3. To see a description of the table, you can type in either of the following:
4. SHOW COLUMNS FROM Customers; or DESCRIBE Customers;

Tip I prefer to use the second command only because there is less to type. Both of the commands return the same information. After you have verified your data, you can continue to add tables to the database. MySQL also enables you to create temporary tables. Temporary tables exist only for the current session and disappear when the connection is dropped. Temporary tables can only be seen by the connection that created them. So if I start up MySQL locally and create a temporary table, Joe, on a remote location, will not see or interact with this table in any way. Temporary tables are useful tools for storing data temporarily, or when you need to store the results of one query and compare them to the results of another. To create a temporary table, issue the following command:

CREATE TEMPORARY TABLE *tablename*(*columnsname* data type);

As you can see, creating a temporary table is almost like creating a true table, the only difference being the word TEMPORARY.

Another useful function that was recently introduced into MySQL is the ability to create a table based on the results of a query. This is a really nice feature because it allows you to create a table without typing in all the column data. It also allows you to easily create a copy of an existing permanent table. If you wanted to create a temporary copy of the Customers table, you would type the following statement:

CREATE TEMPORARY TABLE SELECT * FROM Customers;

If you wanted to create permanent copy of the Customers table, you could omit the word TEMPORARY and insert the new tablename after the word TABLE. The following is the syntax for this action: CREATE TABLE *tablename* SELECT * FROM Customers;

Another feature worth mentioning is the IF NOT EXISTS parameter. This statement can be used to check if a table exists before you actually create it. This is extremely helpful when you need to create a table, but don't know if it exists already. The syntax would look like the following:

CREATE TABLE IF NOT EXISTS *tablename* (*columnname* data type);

Remember that the conditional will only create the table if it doesn't exist. Otherwise it will do nothing.

Note Naming conventions are a necessary evil. They can help the entire project and bring new people up to speed faster.

A word needs to be mentioned about naming conventions. Naming conventions are a good thing. They enable you to have a standardized way of naming objects that you or others may use. By naming things in a certain way, new people can become familiar with a database schema quickly and easily. For example, if you named the table that holds all your customer data Customers and the table that holds all your product data Products, it is much easier for the new guy or girl to learn than if you named the same tables Table_01 and Table_02. The decision is up to you, the database designer. You can name the tables whatever you choose. I prefer to use the following conventions:

- Tables are plural and field names are singular. The table that holds all my customer data is *Customers*, not *Customer*. It just makes sense to me. My *Customers* table, or any table for that matter, holds many different types of the same object. A table is not just a repository for one of my customers but a repository for all my customers. It just makes sense to make tables plural.
- The first letter of a name is always capitalized. This just follows grammar rules. It also looks neater, in my opinion.
- Compound names are separated by an underscore, and the first letter of each name is capitalized (for example, Meet_A_Geek). It may be a pain to type, but it makes things easier to read. Also, spaces and dashes are not allowed in any database object name.
- Use descriptive names and be consistent. When you create a whole bunch of tables, it's nice to know that Last_Name will always be Last_Name, no matter in which table it exists. This is especially helpful when developing programs and queries that access a lot of tables repeatedly.

I will use this convention set throughout this book. Feel free to use whatever makes you comfortable.

The rules have been tried and tested over many databases and have proven time and again that naming conventions are nice—even if it means a few extra key strokes.

Entering the commands from the MySQL Monitor prompt is one way to create the schema of a database. Another way to create your schema is with a script. Scripts are text files that contain all the SQL commands required to build your database. This is probably the best way to create your database, because you have the ability to recreate your database (minus the data) at any given time. It also allows for code reuse—because, generally, computer people are a lazy bunch and the less work you have to do, the better.

To start this process, open your favorite text editor. In the text editor, type the following statements:

```
CREATE DATABASE Temp;
USE DATABASE Temp;
CREATE TABLE Test_Table
(Test_ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
```

```
Test_Name VARCHAR(30),
Test_Date DATETIME,
Test_Giver VARCHAR(30));
INSERT INTO Test_Table
(Test_ID, Test_Name, Test_Date, Test_Giver)
VALUES
(NULL, 'Test','2000-01-01','Glen');
```

It is common practice to format your SQL statements in this way. I'm not saying that it is bad or good. I like it because of the readability. It takes some getting used to, but it is really clear to see. Of course, you can enter your commands in any way that you like, as long as they are in the same order and end with a semicolon.

In this script, you create a database named Temp. You then make this the active database. After that, you create a table called Test_Table. Add four columns to your table. Then add one row of data. If you were to type this into a monitor session, it would take some time. And, when you end your session, all your statements would be gone. Save this file as Temp.sql. You could use any name here, I chose this name because it is easy to identify what the script does.

Before you can use this script, there are a few things you must do. First, make sure the mysqld daemon is running. Second, ensure that you are in the mysql directory. Finally, to process the script, type the following from the command line:

```
bin/mysql –p </complete path/Temp.sql
```

You will be prompted for a password. Use the root password. You must have CREATE authority to run this script. After your file is processed, start up the monitor using Temp as the active database. Execute a SHOW TABLE command. You should see the Test_Table table. Now type in the following command:

```
SELECT * FROM Test_Table;
```

6.3 Altering Existing Tables

Now that you have created your table, what if you need to go in and change something you have done?

Changing tables is just as easy as creating them, you just have to know what you want to change. The column name is very different from the table's name. Changing the column's type is different from changing a column's name. Check out the following examples to see how to alter a column's name, type, and the table's name.

6.4 Changing a Column Name

Sometimes you may need to change the name of one of your columns. Maybe you misspelled it when you created it and didn't notice until a colleague pointed it out. Or maybe your boss has a naming convention that you need to follow. Either way, changing a column name is pretty painless.

If you need to change the name of a column, do the following:

1. Make sure the mysqld daemon is running and that you are in the mysql directory.
2. Start up the MySQL monitor as you did before, using the Meet_A_Geek database as the active database.
3. To change the name of the First_Name column to FirstName in the Customers table, enter the following from the command line:
4. ALTER TABLE Customers
5. CHANGE First_Name FirstName VARCHAR(20);
6. DESCRIBE Customers;

You must specify the data type again, or you will get an error. I used the *DESCRIBE* command to verify the changes. It is not necessary to use that command after you change the table structure—I do it out of habit.

6.5 Changing a Column Type

Changing a column's type is similar to changing a column's name. You are going to change the *Last_Name* from a *VARCHAR(30)* to a *VARCHAR(50)*. Follow steps 1 and 2 of the previous example (changing a column name). Then, instead of typing what is in step 3, type the following:

ALTER TABLE Customers

CHANGE Last_Name Last_Name VARCHAR(50);

DESCRIBE Customers;

Notice that you must use the column name twice. The reason behind this is that MySQL creates a temporary table to hold your changes. This allows users to continue using the database as you make changes.

6.6 Renaming a Table

To change a table's name, make sure the *mysqld* daemon is running and that you are in the *mysql* directory. After you are sure that everything is up and running, start the MySQL monitor. From the monitor's command line, type the following:

ALTER TABLE Customers RENAME Customer_Table; SHOW TABLES FROM Meet_A_Geek;

Altering an existing table or column is pretty straightforward. A few syntactical gotchas are out there, but it is generally an easy process. The hardest part is in the design. Keep that in mind when you are planning or estimating the length of a job.

6.7 Deleting/Adding Columns and Tables

As you can see, when a table or column is created, it is not written in stone and can be changed easily. This even applies to adding columns to an existing table or deleting unwanted columns or tables. The process, again, is pretty straightforward.

6.7.1 Dropping Tables and Columns

To drop or delete tables or columns, make sure the mysqld process is running and that your current directory is the mysql directory. Start up the MySQL monitor with the database you need to make changes to as the active database. After you are up and running, enter the following commands:

To delete an existing table, type

DROP *tablename*;

Where *tablename* is the name of the table you want to delete. For example, to delete the Customers table from the Meet_A_Geek database, you would type the following command:

DROP Customers;

This will delete the entire table and all the data inside the table. Use caution when executing this command. Remember there are no warnings from the monitor. After you drop something, the only way to get it back is through a backup log.

If you need to delete a column from a table, enter the following command:

ALTER TABLE *tablename* DROP *columnname*;

Where *tablename* is the table that holds the column you want to delete, and *columnname* is the column you want to delete.

If you wanted to delete the Last_Name column of the Customers table, you would enter the following statement:

ALTER TABLE Customers DROP Last_Name;

This will delete the column and all the information that the column stored. Again, exercise caution when using this command.

6.7.2 Adding Columns

We have already covered adding tables to a database. You can only create and drop a table. To add a column, you have to use a variation of the ALTER TABLE command. For example, to add a column to an existing schema, execute the following statement:

ALTER TABLE *tablename* ADD *columnname* data type;

Where *tablename* is the table you need to add the column to, and *columnname* is the name of the column to be added. If you wanted to add the Last_Name column back to the Customer table, you would issue the following statement:

ALTER TABLE Customer ADD Last_Name VARCHAR(30);

This will add a column to your table. An important point to remember is that the column you add must have a default value. It cannot be a NOT NULL column. It must contain NULL or some other default value. The reason for this is fairly simple. If you add a column that is NOT NULL, how will MySQL know what value to store? It won't, so you must tell it what to store.

6.8 Using Indexes

An index is a structured file that facilitates data access. What this means to you as the database designer is this: An index on the correct column will increase a query's speed considerably. An index works much like alphabetic separator folders in a file cabinet. It allows you to skip to the part of the alphabet you're looking for. For example, suppose you needed Glen Brazil's record. You could go directly to the B section without going through every single record before you get to Mr. Brazil's. This makes your searches much easier to accomplish, and you're don't waste time looking at records that are not even close to what you need.

Indexes are wonderful things, but they do have some drawbacks. Too many indexes can have an adverse effect. In the example, you went directly to the B section. What if instead of just having letter separators you separated every name. There would be a ton of separators—almost as many as the number of people you were tracking. This would slow things down instead of speeding them up. So it is best not to have too many indexes.

Another adverse effect is that adding a row to an indexed table can be a little slower than adding it to a non-indexed table. Using the example, it takes a little time to put a record in the correct place. You have to go through the separators and then place it in the right order within the file drawer. This is much slower than throwing the record anywhere in the drawer. Retrieval from indexed columns is much quicker. It is up to you to decide if the good outweighs the bad.

Note Indexes speed up data access for SELECT queries, but they slow it down for INSERT, UPDATE, and DELETE queries.

6.8.1 Deciding Which Columns to Include in the Index

After you have decided to use indexes, you have to choose the column or columns you want to index. This can be a little tricky. You want to place an index on the column(s) that you will use most often as a filter in your queries. These are the columns mentioned after the WHERE clause. For example, in the SQL statement SELECT LAST_NAME FROM Customers WHERE Customer_ID < 10, a potential column to index would be the Customer_ID column. Remember, you are going to index the columns that you use most in your queries. If you perform a lot of queries in which you are looking for the last name of a customer, you might want to index the Last_Name column.

Indexes also work better on columns that contain unique data. That is one of the reasons that keys are usually your best choices for indexes. That could also be one of the reasons that people confuse keys and indexes. A key helps define the structure of a database, whereas an index just improves performance.

One index can be made up of one or more columns. For example, in the Meet_A_Geek project, you can have an index that is based on the Last_Name and First_Name columns. This would be useful if you use both of these as criteria in the WHERE clause of an SQL statement.

You can also have more than one index in a table. In fact, you can have up to 16 indexes in one table. You should never have to use that many indexes. If you do, take a serious look at your database design. You may have some problems. However, using a couple of indexes in a table, based on the criteria I stated previously, is not uncommon.

6.8.2 Creating an Index

By default, MySQL creates an index for you if you declare a column as a primary key. There is no need to create an index on this column; otherwise, you would have two indexes on the same column. The syntax for creating a column looks like the following:

CREATE INDEX *indexname* ON tablename(*columnnamelist*);

The *indexname* is anything you choose. Again, use something descriptive to help you remember what makes up this index. Notice the keyword ON. Make sure you don't forget this word when creating an index—you are sure to get a syntax error if you do. The keyword ON is followed by the name of the table that holds the column that is being indexed. The *columnnamelist* is a list of columns that will make up your index. Remember, an index can be made up of one or more columns.

You can also use the ALTER TABLE statement to add an index. For example, if you wanted to add an index to the Last_Name column of the Customers table, you would enter the following:

ALTER TABLE Customers ADD INDEX (IDX_Last_Name);

This same syntax is used if you want to add a primary key to a table that does not have one. That statement would look like the following:

ALTER TABLE Customers ADD PRIMARY KEY (Customer_ID);

Creating an index is a simple process. Indexes are one of the key factors to a fast database, and MySQL does a fantastic job with them. Remember not to overuse indexes because, as with all things, moderation is the key.

6.8.3 Deleting Indexes

Deleting an index is as simple as creating one. The syntax is the same as deleting a column or a table. You can use either of the following statements:

DROP INDEX *indexname* ON *tablename*;

or

ALTER TABLE *tablename* DROP INDEX *indexname*;

They both produce the same effect. Be aware that if you drop a column that makes up an index, that index may be dropped too. If one column of a multi-column index is dropped,

only the dropped column will be deleted from the index. If all the columns that make up an index are dropped, the entire index is dropped as well.

If you need to drop a PRIMARY KEY, use the following syntax:

```
ALTER TABLE tablename DROP PRIMARY KEY;
```

Remember that a table can only have one primary key. If you decide that a different column is better suited as a primary key, you must drop the original one first.

7. MAKING YOUR DATA NORMAL

7.1 Introduction

When structuring a database, putting the right columns in the right tables can be a daunting task. When you finally accomplish this task, you may find out that you have logic problems within your database, especially if you come from the old world of non-relational databases where everything was contained in the same file. Using the old idea of keeping all your data together in one table in a relational databases is a bad idea. It's almost sacrilegious. A set of rules was established to help database designers. These guidelines lead to the design of truly relational databases without logic flaws. Applying these rules to your database structure is referred to as *normalizing* your data, *normalization.* What normalization is and the benefits it can provide. The degrees of normalization.

7.2 What Is Normalization?

Normalization is a set of rules to help database designers develop a schema that minimizes logic problems. Each rule builds on the previous rule. Normalization was adapted because the old style of putting all the data in one place, such as a file or database table, was inefficient and led to logic errors when trying to manipulate the contained data. For example, look at the Meet_A_Geek database. If you stored all the data in the Customers table, the table would look like something like the following:

- Customers
- Customer_ID
- Last_Name
- First_Name
- Address
- Product_Name1
- Product_Cost1
- Product_Picture1

- Product_Name2
- Product_Cost2
- Product_Picture2
- Order_Date
- Order_Quantity
- Shipper_Name

The table has been abbreviated, but it still portrays the general idea. Now, in your Customers table, how could you add a new customer? You would have to add a product and an order as well. What if you wanted to run a report that shows all the products you sell? You could not easily separate products from customers in a simple SQL statement. The beauty of a relational database, if designed correctly, is that you can do just that.

Normalization also makes things easier to understand. Humans tend to break things down to the lowest common denominator. We do it with almost everything—from animals to cars. We look at a big picture and make it less complex by grouping similar things together. The guidelines that normalization provides create the framework to break down the structure. In your sample database. It is easy to see that you have three distinct groups: customers, products, and orders. Following normalization guidelines, you would create your tables based on these groups. The normalization process has a name and a set of rules for each phase of breakdown/grouping. This all may seem a little confusing at first, but I hope you will understand the process as well as the reasons for doing it this way. Most people are happy with a spreadsheet that holds all their pertinent data. The time it takes to break down your schema by going through the normalization process is well spent. It will require less time to go through the process than it would to cut and paste your columns of data so they fit the report the boss wants.

Another advantage to normalizing your database is space consumption. A normalized database will take up less space overall than one that is not normalized. There is less repetition of data, so the actual disk space that is consumed holding your data will be much smaller.

7.2.1 Degrees of Normalization

There are basically three steps of normalization. They are First Normal Form (1NF), Second Normal Form (2NF) and Third Normal Form (3NF). Each form has its own set of rules. After a database conforms to a level, it is considered normalized to that form. Say, for example, that your database conforms to all the rules of the second level of normalization. It is then considered to be in Second Normal Form. Sometimes it is not always the best idea to have a database conform to the highest level of normalization. It may cause an unnecessary level of complexity that could be avoided if it were at a lower form of normalization.

Note There are a total of nine different rules of normalization. They are First Normal Form, Second Normal Form, Third Normal Form, Boyce-Codd Normal Form, Fourth Normal Form, Fifth Normal Form or Join-Projection Normal Form, Strong Join-Projection Normal Form, Over-Strong Join-Projection Normal Form, and Domain Key Normal Form. This book will only cover the first three forms of normalization.

7.2.2 First Normal Form

The rule of First Normal Form states that all repeating columns should be eliminated and put into separate tables. This is a pretty easy rule to follow. Take a look at the schema for the Customers database in Table 7.1.

Table 7.1 Schema for *Customers* Database

Customers
Customer_ID
Last_Name
First_Name
Address
Product_Name1
Product_Cost1
Product_Picture1
Product_Name2
Product_Cost2
Product_Picture2
Order_Number
Order_Date
Order_Quantity
Shipper_Name

In Table 7.1, you have several repeating columns. They mostly deal with products. So, according to the rule, you must eliminate the repeaters and give them their own table. That's easy to do. The resulting database tables are shown in Table 7.2.

Table 7.2 Eliminating Data Repetition in a Database

Customers	Products
Customer_ID	Product_Name
Last_Name	Product_Cost
First_Name	Product_Picture
Address	
Order_Number	
Order_Date	
Order_Quantity	
Shipper_Name	

Now there are two tables. There still is a problem. There is no way currently to relate the data from the original table to the data in the new table. To do that, a key must be added to the second table to establish the relationship. To do this, add a primary key to the Products table called Product_ID, and add a key to Customers table that relates the Products table to the Customers table. The Product_ID field is an ideal candidate. The resulting tables resemble Table 7.3:

Table 7.3 First Normal Form

Customers	Products
Customer_ID	Product_Name
Last_Name	Product_Cost
First_Name	Product_Picture
Address	
Order_Number	
Order_Date	
Order_Quantity	
Shipper_Name	

Now, a one-to-many relationship has been established. This represents what the database will be doing in real life. The client will have many products to sell, regardless of how many customers there are to buy them. Also, a customer still needs to have ordered a product to be a customer. You are no longer obligated to add a new customer every time you add a new product to your inventory. Bringing a database to First Normal Form solves the multiple column heading problem. Too often, inexperienced database designers will do something similar to the non-normalized table in today's first example. They will create many columns representing the same data over and over again. In an electric company in the Northwest, there was a database that tracked nuclear power plant parts. The table in their database, which contained the part numbers, had a repeated column that numbered well into the 30s. Every time a new item was stored for this part, they created a new column to store the information. Obviously, this was a poorly designed database and a programmer's/administrator's nightmare. Normalization helps to clarify the database and break it down into smaller, more understandable pieces. Instead of having to understand a huge, monolithic table that has many different aspects, you only have to understand smaller, more tangible objects and the simple relationships they share with all the other smaller objects. Needless to say, a better understanding of how a database works leads to a better utilization of your assets.

7.2.3 Second Normal Form

The rule of Second Normal Form states that all partial dependencies must be eliminated and separated into their own tables. A ***partial dependency*** is a term to describe data that doesn't rely on the table key to uniquely identify it. In the sample database, the order information is in every record. It would be simpler to use just the order number. The rest of the information could reside in its own table. After breaking out the order information, your schema would resemble Table 7.4.

Table 7.4 Eliminating Partial Dependencies—Second Normal Form

Customers	Products	Ordres
Customer_ID	Product_ID	Order_Number
Product_ID	Order_Date	Product_Name
Order_Number	Product_Cost	Oreder_Quantity
Last_Name	Product_Picture	
First_Name		
Address		
Shipper_Name		

Again, by arranging the schema in this way, you have reflected the real world in your database. You would have to make some changes for your business rules to be applicable, but for illustrating normalization, this is okay.

By now you should be noticing some things. The table that was once hard to read and understand is now making more sense. Relationships between the information that is going to be stored is clearer and easier to understand. Things appear to be more logical. These are some of the advantages to normalizing a database.

One of the major disadvantages of normalization is the time it takes to do. Most people are busy enough, and to spend time making sure their data is normalized when it works just fine is perceived as a waste of time. This is not so. You will spend way more time fixing a broken, non-normalized database than you would a normalized, well-designed database.

By achieving the Second Normal Form, you enjoy some of the advantages of a relational database. For example, you can now add new columns to the Customers table without affecting the Products or the Orders tables. The same applies to the other tables. Getting to this level of normalcy allows data to fall naturally into the bounds for which it was intended.

After you have reached the level of Second Normal Form, most of the logic problems are taken care of.

You can insert a record without excess data in most tables. Looking closer at the Customers table, there is a Shipper_Name column. This column is not dependant on the customer. The next level of normalization will explain how to clear this up.

7.2.4 Third Normal Form

The rule of Third Normal Form is to eliminate and separate any data that is not a key. This column must depend on the key for its value. All values must be uniquely identified by the key. In the sample database, the Customers table contains the Shipper_Name column. The Shipper_Name is not uniquely identified by the key. You could separate this data from the current table and put it into its own table. Table 7.5 shows the resulting database schema:

Table 7.5 Eliminating Non-Key Data for Third Normal Form

Customers	Products	OrdreMaster	OrderDetail	Shipper
Customer_ID	Product_ID	Order_Number	Order_Detail_ID	Shipping
Product_ID	Order_Name	Product_Date	Order_Number	Shipping
Order_Number	Product_Cost	Oreder_Quantity	Oredr_Date	
Shipper_ID	Product_Picture		Order_Quantity	
Last_Name				
First_Name				
Address				

Now all your tables are in Third Normal Form. This provides the most flexibility and prevents any logic errors when inserting or deleting records. Each column in the table is uniquely identified by the key, and no data is repeated. This provides a clean, elegant schema that is easy to work with and easy to expand.

7.3 How Far to Take Normalization

The next decision is how far to go with normalization. Normalization is a subjective science. It is up to you to determine what needs to be broken down. If your database is just going to provide data to a single user for a simple purpose and there is little to no chance of expansion, taking your data to 3NF might be a little extreme. The rules of normalization exist as guidelines to create easily manageable tables that are flexible and efficient.

There are times when normalizing your data to the highest level doesn't make sense. For example, suppose you added another address column to your database. It is quite normal to have two lines for an address. The table schema might look like the following:

- Customer_ID
- Last_Name
- First_Name
- Address1
- Address2

According to the rules that would make this table compliant with First Normal Form, the address columns would be taken out and replaced with the key for the new table. The following is the resulting schema:

Customer_ID	Adderss_ID
Last_Name	Customer_ID
First_Name	Address

The database is now First Normal Form compliant. Your customers can have more than one address.

The problem that exists is that you have overcomplicated a simple idea because you were trying to follow the rules of normalization. In the example, the second address is totally optional. It is there just to collect information that might be used for contact information. There is really no need to break it into its own table and force the rules of normalization on it. In this instance, taking it to a form of normalcy defeats the purpose for which the data is used. It adds another layer of complexity that is not needed. A good way to determine if your normalizing is getting carried away is to look at the number of tables

you have. A large number of tables may indicate that you are normalizing too much. Take a step back and look at your schema. Are you breaking things down just to follow the rules, or is it a practical breakdown. These are the things that you, the database designer, need to decide. Experience and common sense will guide you to make the right decisions. Normalizing is not an exact science; It is a subjective one.

There are six more levels of normalization that have not been discussed so far. They are Boyce-Codd Normal Form, Fourth Normal Form (4NF), Fifth Normal Form (5NF), Strong Join-Protection Normal Form, Over-Strong Join-Protection Normal Form, and Domain Key Normal Form. These forms of normalization may take things further than they need to go. They exist to make a database truly relational. They mostly deal with multiple dependencies and relational keys. If you are familiar with this level of normalization .

REFERENCES

[1]. http://www.w3.org/TR/html401/struct/global.html#edef-DIV

[2]. http://www.w3.org/TR/html401/struct/global.html#edef-SPAN

[3]. The Web Wizard's Guide to XML Cheryl Hughes Addison-Wesley 2003 (covers both XML and CSS)

[4]. http://www.carlops.net

[5]. http://www.rfc-editor.org/rfc.html

[6]. http://www.w3.org

[7]. http://library.ukc.ac.uk/library/lawlinks

[8]. http://elj.warwick.ac.uk/jilt/cases/97_2pitm/default.htm

[9]. http://webjcli.ncl.ac.uk/articles1/widdis1.html

[10]. http://www.patent.gov.uk/

[11]. http://www.patent.gov.uk/copy/index.htm

[12]. http://www.wipo.int/

[13]. http://www.cyber-rights.org/reports/demon.htm

[14].http://www.law.ed.ac.uk/links/

[15].http://newsvote.bbc.co.uk/hi/english/uk/newsid_696000/696289.stm

[16]. http://newsvote.bbc.co.uk/hi/english/sci/tech/newsid_695000/695596.stm

[17].http://www.wired.com/news/politics/0,1283,20107,00.html

[18].http://www.wired.com/news/politics/0,1283,20107,00.html

[19]. http://www.wired.com/news/politics/0,1283,18764,00.html

[20]. http://www.cyber-rights.org/reports/demon.htm