UNIVERSITY OF DAR ES SALAAM

COMPUTING CENTRE

Diploma in Computing and Information Technology

UNIVERSITY OF DAR ES SALAAM COMPUTING CENTRE
One Stop Centre for ICT Solutions

CTT 06105: DESIGN AND IMPLEMENTATION OF DATABASE-DRIVEN WEBSITES

© 2016 University of Dar es Salaam Computing Centre

University Road
P.O Box 35062
Dar es Salaam
Tanzania

Tel: +255 (022) 2410645
Fax: +255 (022) 2410690
Email: **training@udsm.ac.tz**
Internet: **http://www.ucc.co.tz**

Introduction

Course Description

This course will enable you to build real-world, dynamic websites. If you've built websites using plain HTML, you realize the limitations of this approach. Static content from a pure HTML website is just that—static. It stays the same unless you physically update it. Your users can't interact with the site in any meaningful fashion.

Using a language such as PHP and a database such as MySQL allows you to make your sites dynamic: to have them be customizable and contain real-time information.

We have deliberately focused this course on real-world applications, even in the introductory chapters

Course Objectives

At the end of the course students should be able to

1. Identify different tools for developing database driven websites
2. Describe the structure of database driven websites
3. Describe what happens behind the browser in a web-based application
4. Design relational databases for real-world applications to be stored in database server
5. Apply server side scripting language and other tools to create web forms and pages that properly forms the user interface
6. Create server side script code that utilizes SQL language and built-in functions to perform data manipulations via the webpage
7. Apply cookies and session controls in implementing database driven websites
8. Describe different ways of identifying the visitor in web applications
9. Describe different ways of encrypting the password
10. Implement simple access controls and basic authentication
11. Utilize cookies and session controls to implement custom authentication

Delivery Methodology

The course will be delivered in form of lecturers in the classroom and in the Computer lab accordingly. Exercise with real life nature will be provided during and at the end of the class. The manual is also designed such that one can follow the course at own time and pace.

Chapter 1: Introduction to database driven website

1.1 Web development

Web development is the process of developing websites or webpages hosted on the Internet or intranet. Think about your favorite website; whether it’s an e-commerce store, blog, social network, online video streaming service, or any other type of Internet application, it all had to be built by a web developer.

But what does that look like? The web development process can be divided into three main components: server-side coding, client-side coding and database technology.

1.1.1 Client-Side Coding

When you are viewing or using a website, you are known as a ‘user’ or a ‘client.’ So web applications or computer programs executed by a user’s web browser are referred to as client-side scripts. That means the program requests any files it needs to run from the web server, and then runs within the client’s web browser.

This allows a webpage to have unique and alternating content depending on a user’s input or other variables. Ajax, Flash, JavaScript, jQuery, Microsoft Silverlight, HTML5, and CSS3 are some examples of popular languages, plugins and libraries used in client-side scripting

1.1.2 Server-Side Coding

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.

The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content. Example of web servers are Zeus, Apache, Glassfish,etc

A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Web servers are not only used for serving the World Wide Web. They can also be found embedded in devices such as printers, routers, webcams and serving only a local network.

HOW A WEB SERVER WORKS

1)The client’s browser divides the URL into diff. parts dividing including address, path name and protocol.

2)DNS translates the domain name into the corresponding IP address .the numeric combination represents the site's true address on the internet.

3)The browser now decides which protocol should be used .a protocol in common parlance is a language which the client's to communicate with the server. FTP,HTTP are some such protocols.

4)The server sends a GET request to the web server to retrieve the address it has been given. It verifies given address exists, finds necessary files ,runs appropriate scripts exchanges cookies if necessary and returns back to the browser.

5)The browser now converts the data into HTML and displays results to the user. If it does not locate it sends an error message to the browser and to the client.

In contrast to client-side scripts, server-side scripts are executed on the web server whenever a user requests a document or service. The server then produces the document, usually in the form of HTML, which can be read by the client's browser.

The document sent to the browser may often contain client-side scripts. ASP.NET, PHP, Java, ColdFusion, Perl, Python, and Ruby are examples of languages used for server-side coding.

1.1.3 Database Technology

A database server is a computer program that provides database services to other computer programs or computers, as defined by the client–server model. The term may also refer to a computer dedicated to running such a program. Database management systems frequently provide database server functionality, and some DBMSs (e.g., MySQL) rely exclusively on the client–server model for database access. Such a server is accessed either through a "front end" running on the user's computer which displays requested data or the "back end" which runs on the server and handles tasks such as data analysis and storage. Most of the Database servers works with the base of Query language. Each Database understands its query language and converts it to Server readable form and executes it to retrieve the results

For any website to function on the Internet, it must be hosted within a database on a webserver. The database contains all the files required for a website and its applications to function. Websites typically use some form of a relational database management system (RDBMS); the leading RDBMS options are Oracle, Microsoft SQL Server, Apache, and IBM. Open-source RDBMS are also very popular, led by MySQL, PostgreSQL, and MariaDB.

1.2 Web development language

When it comes to choosing the best web development language for your website, it's important to remember that there is no single best language.

Instead, a web developer will choose the option that best suits your project, based on the specific functionality or features you want. Which programming languages are most likely to come up in conversation?

While there are a couple of basic languages in common use, other languages are used specifically for client-side scripting or server-side scripting. Here is an overview of the more popular web development languages in use by the industry today.

1.2.1 Basic web development languages

HTML and CSS are the two most basic web development languages, and are used to build nearly all webpages on the Internet.

HTML

HTML is the standardized markup language that structures and formats content on the web. Page elements like the titles, headings, text and links are included in the HTML document. It is one of the core technologies in use on the Internet and serves as the backbone of all webpages.

CSS

CSS (Cascading Style Sheets) is a style-sheet language that basically allows web developers to "set it and forget it." Paired with HTML, CSS allows a programmer to define the look and format of multiple webpages at once; elements like color, layout and fonts are specified in one file that's kept separate from the core code of the webpage.

These two languages provide the basic structure and style information used to create a static webpage — a page that looks the same to everyone who visits it. Many webpages now are dynamic webpages, which are slightly tailored to each new visitor. To create these more complex webpages, you need to add more advanced client-side and server-side scripting.

1.2.2 Client-side scripting

Client-side scripting — which includes HTML and CSS — is any code that runs within a web browser. This means that the web browser temporarily downloads all the files from a web server and, in turn, displays a static web page; you would be able to view these files even if you lost your Internet connection (as long as you left your web browser open). JavaScript and ActionScript are the two most commonly used client-side scripts.

JavaScript

JavaScript is the programming language that brings animation, games, apps, interactivity and other dynamic effects to life. After HTML and CSS, it's the most ubiquitous of the client-side scripts. Some JavaScript applications can even run without connecting back to a web server, which means they'll work in a browser with or without an Internet connection.

ActionScript

ActionScript is the language used for Adobe Flash, which is especially well suited for rich Internet applications that use Flash animation and streaming audio and video.

Whether you use ActionScript or JavaScript is a matter of personal preference, but if you want to use the popular Adobe Flash Player software to share multimedia applications, ActionScript is a must.

1.2.3 Server-Side Scripting

All websites need to be hosted (i.e. stored) in a database on a web server. Server-side scripting simply refers to any code that facilitates the transfer of data from that web server to a browser. It also refers to any code used to build a database or manage data on the web server itself.

Server-side scripts run on the web server, which has the power and resources to run programs that are too resource intensive to be run by a web browser. Server-side scripts are also more secure, because the source code remains on the web server rather than being temporarily stored on an individual's computer.

PHP

Used by 75 percent of all web servers, PHP is a general-purpose server-side scripting language. The chief advantages of PHP are that it is open source, with a huge online community to support it, and that it's compatible across multiple platforms. PHP is most often used by websites with lower traffic demands.

Java

According to a study conducted by W3Tech, Java is the server-side language of choice for large-scale websites with a high volume of traffic. Sam's Club, Amazon andApple App Store use Java-based web frameworks.

One potential reason for its popularity among high traffic websites is that Java frameworks outperform other language frameworks in raw speed benchmark tests. That means faster server-based web applications for large scale websites. Java Servlets, JSP and WebObjects are examples of server-side solutions that use Java.

Python

Python is a general purpose, high-level programming language that puts an emphasis on code readability; for web developers, this means they can do more with fewer lines of code than other popular languages.

Python does this through the use of a large standard library, which keeps the actual code short and simple. This library is a file that contains pre-coded functions, provided by the community, which you can download to your server and use in your own code whenever a specific task appears. Like Java, Python was designed for web servers that deal with a large amount of traffic. Shopzilla, Yahoo Maps, and the National Weather Service are examples of sites that use Python.

1.3 Website

A website is a site (location) on the World Wide Web. Each Web site contains a home page, which is the first document users see when they enter the site. The site might also contain additional documents and files. Each site is owned and managed by an individual, company or organization.

1.3.1 How the website works

On the simplest level, the Web physically consists of the following components –

- **Your personal computer** – This is the PC at which you sit to see the web.
- **A Web browser** – A software installed on your PC which helps you to browse the Web.
- **An internet connection** – This is provided by an ISP and connects you to the internet to reach to any Website.
- **A Web server** – This is the computer on which a website is hosted.
- **Routers & Switches** – They are the combination of software and hardware who take your request and pass to appropriate Web server.

The Web is known as a client-server system. Your computer is the client and the remote computers that store electronic files are the servers.

How the website works

In summary a website works as follows:-

- A user enters a URL into a browser (for example, www.ucc.co.tz) This request is passed to a domain name server.
- The domain name server returns an IP address for the server that hosts the Website (for example, 196.44.165.1).
- The browser requests the page from the Web server using the IP address specified by the domain name server.
- The Web server returns the page to the IP address specified by the browser requesting the page. The page may also contain links to other files on the same server, such as images, which the browser will also request.
- The browser collects all the information and displays to your computer in the form of Web page.

1.4 Dynamic Vs. Static Websites

If you have tried to delve into data driven web design you must have come across these two terms. Here is a brief breakdown of what each entails:

- Static website: This website does not change every time the browser loads a page. If a user clicks a button, nothing changes on the layout and content of the page. The only changes occur when the user loads a new page or when the admin loads another page on the web browser. The content is stored on the web file system and it will always be presented in the same format.
- Dynamic website: As the name implies, these pages change every time they are loaded without the webmaster having to make the changes. If a user clicks on an image or text, some changes are observed on the particular page. The fact that the content is stored outside the web file system makes it easier to manage and you can manipulate data quickly.

1.5 Database driven web pages

One of the most common types of dynamic web pages is the database driven type. This means that you have a web page that grabs information from a database (the web page is connected to the database by programming,) and inserts that information into the web page each time it is loaded. The layout and the page content are created separately. The content is stored in a database while the layout is stored on the web server

If the information stored in the database changes, the web page connected to the database will also change accordingly (and automatically,) without human intervention.

This is commonly seen on online banking sites where you can log in (by entering your user name and password) and check out your bank account balance. Your bank account information is stored in a database and has been connected to the web page with programming thus enabling you to see your banking information.

1.6 Technologies for building a dynamic website

Database driven sites can be built using several competing technologies, each with its own advantages. Some of those technologies/tools include:

1. PHP
2. JavaScript is now ALSO being used on the server, with the engine called: Node.js
3. Java / JSP
4. Python

5. Ruby
6. ASP.NET - usually with the C# programming language

1.7 Methods for building dynamic website

You have basic knowledge of HTML, CSS and JS. That's good. But, for data-driven or dynamic site or a CMS (like zomato), you need to know a server side language, preferably PHP. The next thing is to know database manipulations. Also, it'll help if you know the basics of MySQL as it works better with PHP for data manipulation.

Now, here are two ways in which you can create your desired site:

1. Use one of the existing CMS platforms, like WordPress or Drupal. You can chose your desired theme from the avialable list. Since you have basic programming language, you might be able to make changes in the theme files according to your needs. If you're lucky, you won't have to, you'll find a theme which perfectly fits you needs. This is a fast process and helps you build a site in lesser time as compared to the general site building method. But, the negative point here will be, you might not get complete independence over the design.
2. The second way is to build your own website from scratch. Design the front end, since you already are familiar with HTML, CSS and JS, this will be easy. Then, move on tk the backend part, where actual work is to be done. Design your own methods of how the data should be used and displayed. This will be done using PHP and MySQL. This'll be time consuming and will be tiresome to write the whole code from scratch. But, you can take help of various problem solving platforms when you get stuck. The plus side here is, you have complete control over everything.

1.8 Examples of Database Driven Websites

If you are a web designer then you must have in mind the type of website you are building. This will help you determine whether a static website is best suited for your clients or not. Common database driven websites include:

- **E-Commerce platforms**: These businesses leverage data driven websites because of the expected changes in prices, offers and services. This guarantees the information internet users find is always fresh and up-to-date.

- **Content Management Systems (CMS):** If the website is going to use a CMS then it is database driven. Users can easily update content on the website even without the need for any specialized programming skills. These CMSs include WordPress and Joomla and they have an easy-to-use editor to allow publishing of content, editing and deleting.
- **Blogs:** Most blogs and online community forums are database driven because they involve regular updates by users. Whether people are leaving comments or liking a website there is immediate change on the page.

Exercises

1. What is a dynamic website?
2. Differentiate a server side from a client side scripting language
3. What is the function of a web server?
4. What is the main use of a CSS in web site development?
5. Explain how web servers work.

Chapter 2: MySQL Server Overview

From Web Technologies Course, you know already that PHP is a server-side scripting language that lets you insert into your Web pages instructions that your Web server software (be it Apache, IIS, or whatever) will execute before it sends those pages to browsers that request them.

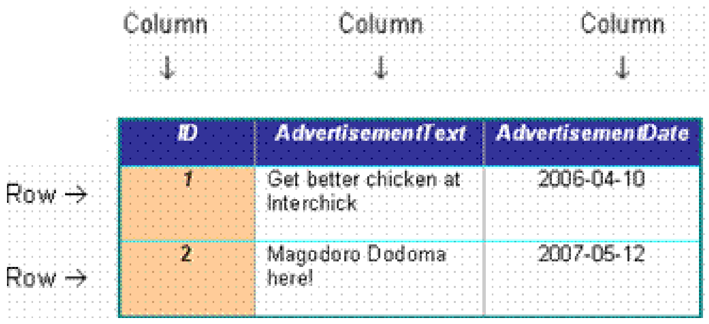
Now that's all well and good, but things really get interesting when a database is added to the mix. A database server (in our case, MySQL) is a program that can store large amounts of information in an organized format that's easily accessible through scripting languages like PHP.

For example, you could tell PHP to look in the database for a list of advertisements that you'd like to appear on your Website. In this example, the advertisements would be stored entirely in the database. The advantages of this approach would be twofold.

First, instead of having to write an HTML file for each of your advertisements, you could write a single PHP file that was designed to fetch any advertisement out of the database and display it.

Second, adding a advertisement to your Website would be a simple matter of inserting the advertisement into the database. The PHP code would take care of the rest, automatically displaying the new advertisement along with the others when it fetched the list from the database.

Let's run with this example as we look at how data is stored in a database. A database is composed of one or more **tables**, each of which contains a list of *things*. For our advertisement database, we'd probably start with a table called Advertisements that would contain a list of advertisements. Each table in a database has one or more **columns**, or **fields**. Each column holds a certain piece of information about each item in the table. In our example, our Advertisements table might have columns for the text of the advertisements, and the dates on which the advertisements were added to the database. Each advertisement that we stored in this table would then be said to be a **row** in the table. These rows and columns form a table that looks like Figure 2.1



ID	AdvertisementText	AdvertisementDate
1	Get better chicken at Interchick	2006-04-10
2	Magodoro Dodoma here!	2007-05-12

Figure 2.1. Structure of a typical database table

Notice that, in addition to columns for the advertisement text (AdvertisementText) and the date of the advertisement (AdvertisementDate), I included a column named ID. As a matter of good design, a database table should always provide a way to identify uniquely each of its rows. Since it's possible that a single advertisement could be entered more than once on the same date, the AdvertisementText and AdvertisementDate columns can't be relied upon to tell all the advertisements apart. The function of the ID column, therefore, is to assign a unique number to each advertisement, so we have an easy way to refer to them, and to keep track of which advertisement is which.

So, to review, the above is a three-column table with two rows, or entries. Each row in the table contains three fields, one for each column in the table: the advertisement's ID, its text, and the date of the advertisement. With this basic terminology under our belts, we're ready to get started with MySQL.

2.1. Logging On to MySQL

The standard interface for working with MySQL databases is to connect to the MySQL server software and type commands

To make this connection to the server, you'll need the MySQL client program. If you installed the MySQL server software yourself, either under Windows or under some brand of UNIX, you already have this program installed in the same location as the server program. Under Linux, for example, the program is called mysql and is located by default in the /usr/local/mysql/bin directory.

Under Windows, the program is called mysql.exe and is located by default in the C:\mysql\bin directory.

Whichever operating system you use, you'll end up at a command line, ready to run the MySQL client program and connect to your MySQL server. Here's what you should type:

```
mysql -h hostname –u username –p
```

You need to replace *hostname* with the host name or IP address of the computer on which the MySQL server is running. If the client program is run on the same computer as the server, you can actually leave off the -h *hostname* part of the command instead of typing -h localhost or –h 127.0.0.1. *username* should be your MySQL user name. If you installed the MySQL server yourself, this will just be root. If you're using your Web host's MySQL server, this should be the MySQL user name they assigned you.

The -p argument tells the program to prompt you for your password, which it should do as soon as you enter the command above. If you set up the MySQL server yourself, this password is the root password you chose during the installation. If you're using your Web host's MySQL server, this should be the MySQL password they gave you.

If you typed everything properly, the MySQL client program will introduce itself and then dump you on the MySQL command line:

```
mysql>
```

Now, the MySQL server can actually keep track of more than one database. This allows a Web host to set up a single MySQL server for use by several of its subscribers, for example. So your next step should be to choose a database with which to work. First, let's retrieve a list of databases on the current server. Type this command (don't forget the semicolon!), and press Enter.

```
mysql>SHOW DATABASES;
```

MySQL will show you a list of the databases on the server. If this is a brand new server (i.e. if you installed this server yourself), the list should looklike this:

```
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.11 sec)
```

The MySQL server uses the first database, called mysql, to keep track of users, their passwords, and what they're allowed to do. The second database, called test, is a sample database. You can actually get rid of this database. Deleting something in MySQL is called "dropping" it, and the command for doing so is appropriately named:

```
mysql>DROP DATABASE test;
```

If you type this command and press Enter, MySQL will obediently delete the database, saying "Query OK" in confirmation. Notice that you're not prompted with any kind of "are you sure?" message. You have to be very careful to type your commands correctly in MySQL because, as this example shows, you can obliterate your entire database—along with all the information it contains—with one single command!

Before we go any further, let's learn a couple of things about the MySQL command line. As you may have noticed, all commands in MySQL are terminated by a semicolon (;). If you forget the semicolon, MySQL will think you haven't finished typing your command, and will let you continue to type on another line:

```
mysql>SHOW
->DATABASES;
```

MySQL shows you that it's waiting for you to type more of your command by changing the prompt from mysql> to ->. For long commands, this can be handy, as it allows you to spread your commands out over several lines. If you get halfway through a command and realize you made a mistake early on, you may want to cancel the current command entirely and start over from scratch. To do this, type \c and press Enter:

```
mysql>DROP DATABASE\c
mysql>
```

MySQL will completely ignore the command you had begun to type, and will go back to the prompt to wait for another command. Finally, if at any time you want to exit the MySQL client program, just type quit or exit (either one will work). This is the only command that doesn't need a semicolon, but you can use one if you want to.

```
mysql>quit
Bye
```

The set of commands we are going to use to tell MySQL what to do for the rest of this course is part of a standard called Structured Query Language, or SQL (pronounced either "sequel" or "ess-cue-ell"). Commands in SQL are also called queries (I'll use these two terms interchangeably in this course).

So what's SQL?

SQL is the standard language for interacting with most databases, so even if you move from MySQL to a database like Microsoft SQL Server in the future, you'll find that most of the commands are identical. It's important that you understand the distinction between SQL and MySQL. MySQL is the database server software that you're using. SQL is the language that you use to interact with that database.

2.2. Creating a Database

Those of you running a MySQL server that you installed yourselves will need to create your own database. It's just as easy to create a database as it is to delete one:

```
mysql>CREATE DATABASE advertisements;
```

I chose to name the database advertisements, because that fits with the example we're using. Feel free to give the database any name you like.

Now that we have a database, we need to tell MySQL that we want to use it. Again, the command isn't too hard to remember:

```
mysql>USE advertisements;
```

You're now ready to use your database. Since a database is empty until you add some tables to it, our first order of business will be to create a table that will hold our advertisements.

2.2.1 Creating a Table

The SQL commands we've encountered so far have been reasonably simple, but as tables are so flexible, it takes a more complicated command to create them. The basic form of the command is as follows:

```
mysql>CREATE TABLE table_name (
-> column_1_name column_1_type column_1_details,
-> column_2_name column_2_type column_2_details,
-> ...
->);
```

Let's return to our example Advertisements table. Recall that it had three columns: ID (a number), AdvertisementText (the text of the advertisement), and AdvertisementDate (the date the advertisement was entered). The command to create this table looks like this:

```
mysql>CREATE TABLE Advertisements (
-> ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> AdvertisementText TEXT,
-> AdvertisementDate DATE NOT NULL
->);
```

It looks pretty scary, Let's break it down:

1. The first line is fairly simple: it says that we want to create a new table called Advertisements.
2. The second line says that we want a column called ID that will contain an integer (INT), that is, a whole number. The rest of this line deals with special details for this column. First, this column is not allowed to be left blank (NOT NULL). Next, if we don't specify any value in particular when we add a new entry to the table, we want MySQL to pick a value that is one more than the highest value in the table so far (AUTO_INCREMENT). Finally, this column is to act as a unique identifier for the entries in this table, so all values in this column must be unique (PRIMARY KEY).
3. The third line is super-simple; it says that we want a column called AdvertisementText, which will contain text (TEXT).
4. The fourth line defines our last column, called AdvertisementDate, which will contain data of type DATE, and which cannot be left blank (NOT NULL).

Note that, while you're free to type your SQL commands in upper or lower case, a MySQL server running on a UNIX-based system will be case-sensitive when it comes to database and table names, as these correspond to directories and files in the MySQL data directory. Otherwise, MySQL is completely case-insensitive, but for one exception: table, column, and other names must be spelled exactly the same when they're used more than once in the same command.

Note also that we assigned a specific type of data to each column we created. ID will contain integers, AdvertisementText will contain text, and AdvertisementDate will contain dates. MySQL requires you to specify a data type for each column in advance. Not only does this help keep your data organized, but it allows you to compare the values within a column in powerful ways, as we'll see later.

Now, if you typed the above command correctly, MySQL will respond with Query OK and your first table will be created. If you made a typing mistake, MySQL will tell you there was a problem with the query you typed, and will try to give you some indication of where it had trouble understanding what you meant. For such a complicated command, Query OK is a pretty boring response. Let's have a look at your new table to make sure it was created properly. Type the following command:

```
mysql>SHOW  TABLES;
```

The response should look like this:

```
 +-----------------+
 | Tables in advertisements |
 +-----------------+
 | Advertisements |
 +-----------------+
 1 row in set
```

This is a list of all the tables in our database (which I named advertisements above). The list contains only one table: the Advertisements table we just created. So far everything looks good. Let's have a closer look at the Advertisements table itself:

```
mysql>DESCRIBE  Advertisements;
 +----------+---------+------+-----+------------+----------------+
 | Field    | Type | Null | Key | Default | Extra |
 +----------+---------+------+-----+------------+----------------+
 | ID | int(11) | | PRI | NULL | auto_increment |
 | AdvertisementText | text | YES | | NULL | |
 | AdvertisementDate | date | | | 0000-00-00 | |
 +----------+---------+------+-----+------------+----------------+
 3 rows in set
```

As we can see, there are three columns (or fields) in this table, which appear as the 3 rows in this table of results. The details are somewhat cryptic, but if you look at them closely for a while you should be able to figure out what most of them mean. Don't worry about it too much, though. We've got better things to do, like adding some advertisements to our table! We need to look at just one more thing before we get to that, though:

2.2.2 Deleting a table

This task is as frighteningly easy as deleting a database. In fact, the command is almost identical:

```
mysql>DROP TABLE tableName;
```

2.3. Inserting Data into a Table

Our database is created and our table is built; all that's left is to put some actual advertisements into our database. The command for inserting data into our database is called, appropriately enough, INSERT. There are two basic forms of this command:

```
mysql>INSERT INTO table_name SET
-> columnName1 = value1,
-> columnName2 = value2,
-> ...
->;
```

```
mysql>INSERT INTO table_name
-> (columnName1, columnName2, ...)
-> VALUES (value1, value2, ...);
```

So, to add a advertisement to our table, we can choose from either of these commands:

```
mysql>INSERT INTO Advertisements SET
->AdvertisementText = "Get better Chicken in the InterChick!",
->AdvertisementDate = "2006-04-11";
```

```
mysql>INSERT INTO Advertisements
->(AdvertisementText, AdvertisementDate) VALUES (
->" Get better Chicken in the InterChick!",
->"2006-04-11"
->);
```

Note that in the second form of the INSERT command, the order in which you list the columns must match the order in which you list the values. Otherwise, the order of the columns doesn't matter, as long as you give values for all required fields. Now that you know how to add entries to a table, let's see how we can view those entries.

2.4. Viewing Stored Data

The command we use to view data stored in your database tables, SELECT, is the most complicated command in the SQL language. The reason for this complexity is that the chief strength of a database is its flexibility in data retrieval and presentation. As, at this point in our experience with databases, we need only fairly simple lists of results, we'll just consider the simpler forms of the SELECT command. This command will list everything stored in the Advertisements table:

```
mysql>SELECT  * FROM Advertisements;
```

Read aloud, this command says "select everything from Advertisements". If you try this command, your results will resemble this:

```
+----+--------------------------------------+-------------------+
| ID | AdvertisementText                    | AdvertisementDate |
+----+--------------------------------------+-------------------+
| 1  | Get better Chicken in the Interchick! | 2006-04-11        |
+----+--------------------------------------+-------------------+
1 row in set (0.05 sec)
```

It looks a little disorganized because the text in the AdvertisementText column is too long for the table to fit properly on the screen. For this reason, you might want to tell MySQL to leave out the AdvertisementText column. The command for doing this is as follows:

```
mysql>SELECT  ID, AdvertisementDate FROM Advertisements;
```

This time instead of telling it to "select everything", we told it precisely which columns we wanted to see. The results look like this:

```
+----+-------------------+
| ID | AdvertisementDate |
+----+-------------------+
| 1  | 2006-04-11        |
+----+-------------------+
1 row in set (0.00 sec)
```

Not bad, but we'd like to see at least some of the advertisement text, wouldn't we? In addition to listing the columns that we want the SELECT command to show us, we can modify those columns with functions. One function, called LEFT, lets us tell MySQL to display up to a specified maximum number of characters when it displays a column. For example, let's say we wanted to see only the first 18 characters of the AdvertisementText column:

```
mysql>SELECT ID, LEFT(AdvertisementText,18), AdvertisementDate FROM Advertisements;
 +----+----------------------------+-------------------+
 | ID | LEFT(AdvertisementText,18) | AdvertisementDate |
 +----+----------------------------+-------------------+
 | 1  | Get better Chicken         | 2006-04-11        |
 +----+----------------------------+-------------------+
 1 row in set (0.05 sec)
```

See how that worked? Another useful function is COUNT, which simply lets us count the number of results returned. So, for example, if we wanted to find out how many advertisements were stored in our table, we could use the following command:

```
mysql>SELECT  COUNT(*) FROM Advertisements;
 +----------+
 | COUNT(*) |
 +----------+
 | 1        |
 +----------+
 1 row in set (0.06 sec)
```

As you can see, we have just one advertisement in our table. So far, all our examples have fetched all the entries in the table. But if we add what's called a **WHERE clause** (for reasons that will become obvious in a moment) to a SELECT command, we can limit which entries are returned as results. Consider this example:

```
mysql>SELECT  COUNT(*) FROM Advertisements WHERE AdvertisementDate >= "2000-01-01";
```

This query will count the number of advertisements that have dates "greater than or equal to" January 1st, 2000. "Greater than or equal to", when dealing with dates, means "on or after".

Another variation on this theme lets you search for entries that contain a certain piece of text. Check out this query:

```
mysql>SELECT  AdvertisementText FROM Advertisements WHERE AdvertisementText LIKE  "%chicken%";
```

This query displays the text of all advertisements that contain the word "chicken" in their AdvertisementText column. The LIKE keyword tells MySQL that the named column must match the given pattern. In this case, the pattern we've used is "%chicken%". The % signs here indicate that the word "chicken" may be preceded and/or followed by any string of text.

Additional conditions may also be combined in the WHERE clause to further restrict results. For example, to display Lock advertisements from April 2000 only, we could use the following query:

```
mysql>SELECT  AdvertisementText FROM Advertisements WHERE
    ->AdvertisementText  LIKE "%Lock%" AND
    ->AdvertisementDate  >= "2000-04-01" AND
    ->AdvertisementDate  < "2000-05-01";
```

Enter a few more advertisements into the table and experiment with SELECT statements a little. A good familiarity with the SELECT statement will come in handy later in this course. There's a lot more you can do with the SELECT statement, but we'll save looking at some of its more advanced features for later, when we need them.

2.5. Modifying and Deleting Stored Data

2.5.1 Modifying Stored Data

Having entered your data into a database table, you might like to change it. Whether you want to correct a spelling mistake, or change the date attached to a advertisement, such alterations are made using the UPDATE command. This command contains elements of the INSERT command (that set column values) and of the SELECT command (that pick out entries to modify). The general form of the UPDATE command is as follows:

```
mysql>UPDATE table_name SET
    -> col_name = new_value, ...
    ->WHERE conditions;
```

So, for example, if we wanted to change the date on the advertisement we entered above, we'd use the following command:

```
mysql>UPDATE  Advertisements SET AdvertisementDate="1990-04-01" WHERE ID=1;
```

Here's where that ID column comes in handy. It allows us to easily single out a advertisement for changes. The WHERE clause here works just like it does in the SELECT command. This next command, for example, changes the date of all entries that contain the word "chicken":

```
mysql>UPDATE  Advertisements SET AdvertisementDate="1990-04-01"
    ->WHERE AdvertisementText  LIKE "%chicken%";
```

2.5.2 Deleting Stored Data

The deletion of entries in SQL is dangerously easy, which, if you haven't noticed yet, is a recurring theme. Here's the command syntax:

```
mysql>DELETE  FROM table_name WHERE conditons;
```

So to delete all chicken advertisements from your table, you'd use the following query:

```
mysql>DELETE    FROM    Advertisements    WHERE    AdvertisementText    LIKE
"%chicken%";
```

One thing to note is that the WHERE clause is actually optional. You should be very careful; however, if you leave it off, as the DELETE command will then apply to all entries in the table. This command will empty the Advertisements table in one fell swoop:

```
mysql>DELETE  FROM Advertisements;
```

Exercises

1. The integer 56678685 could be which data type(s)?
2. How would you define a field that could contain only the following strings: apple, pear, banana, cherry?
3. How would you formulate a string comparison using LIKE to match first names of "John" or "Joseph"?
4. How would you explicitly refer to a field called id in a table called table1?

Chapter 3: PHP and HTML Form Overview

PHP is a server-side scripting language that enables you to build dynamic web pages. PHP pages may contain text, HTML, and script blocks. When a browser requests a PHP page, the PHP script is executed on the web server, and the resulting HTML is displayed in the browser.

A PHP page must have a PHP-supported extension. Typically, a PHP file ends with .php, although other PHP extensions such as .php4 and .phtml also exist. However, .php is the most common extension.

A PHP script block can appear anywhere in a .php page, and each page can contain more than one PHP script block. A PHP script block must begin with <?php and end with ?>.

The following is an example of a PHP script block:

```
<?php
echo "Hello World";
?>
```

With PHP onboard you can do the following on the websites

1. Create custom content based on different variables
2. Write information to the database or read information from the database. For this case PHP is commonly used to process data posted via HTML Forms.
3. Tracking user information or activities in the database

3.1 HTML Forms

Forms are used to collect data inputted by a user. They can be used as an interface for a web application, for example, or to send data across the web.

On their own, forms aren’t usually especially helpful. They tend to be used in conjunction with a programming language to process the information inputted by the user. These scripts take all manner of guises that are largely outside of the remit of this website because they require languages other than HTML and CSS.

The basic element used in the actual HTML forms are form, input, textarea, select and option.

3.1.1 Form

Form defines the form and within this tag, if you are using a form for a user to submit information (which we are assuming at this level), an action attribute is needed to tell the form where its contents will be sent to.
The method attribute tells the form how the data in it is going to be sent and it can have the value get, which is default, and latches the form information onto a web address, or post, which (essentially) invisibly sends the form's information.
Get is used for shorter chunks of non-sensitive information - you might see the information you have submitted in a web site's search to appear in the web address of its search results page, for example.post is used for lengthier, more secure submissions, such as in contact forms.
So a form element will look something like this:

```
<form action="processingscript.php" method="post">
</form>
```

3.1.2 Input

The input tag is the daddy of the form world. It can take a multitude of guises, the most common of which are outlined below :

- <input **type="text"**> or simply <input> is a standard textbox. This can also have a value attribute, which sets the initial text in the textbox.
- <input **type="password"**> is similar to the textbox, but the characters typed in by the user will be hidden.
- <input **type="checkbox"**> is a checkbox, which can be toggled on and off by the user. This can also have a checked attribute (<input type="checkbox" **checked**> - the attribute doesn't require a value), and makes the initial state of the check box to be switched on, as it were.
- <input **type="radio"**> is similar to a checkbox, but the user can only select one radio button in a group. This can also have a checked attribute.
- <input **type="submit"**> is a button that when selected will submit the form. You can control the text that appears on the submit button with thevalue attribute, for example <input type="submit" value="Text on a button">.

3.1.3 TextArea

textarea is basically, a large, multi-line textbox. The anticipated number of rows and columns can be defined with rows and cols attributes, although you can manipulate the size to your heart's content using CSS.

```
<textarea rows="5" cols="20">A big load of text</textarea>
```

Any text you choose to place between the opening and closing tags (in this case "a big load of text") will form the initial value of the text area.

3.1.4 Select

The select tag works with the option tag to make drop-down select boxes.

```
<select>
   <option value="option 1">Option 1</option>
   <option value="option 2">Option 2</option>
   <option value="third option">Option 3</option>
</select>
```

When the form is submitted, the value of the selected option will be sent. This value will be an explicit value specified with the value attribute. So, in the above example, if the first item is selected, "Option 1" will be sent, if the third item is selected, "third option" will be sent.
Similar to the checked attribute of checkboxes and radio buttons, an option tag can also have a selected attribute, to start off with one of the items already being selected, eg. <option selected>Rodent</option> would pre-select "Rodent" from the items.

3.1.5 Name

All of the tags mentioned above will look very nice presented on the page but if you hook up your form to a form-handling script, they will all be ignored. This is because the form fields need names. So to all of the fields, the attribute name needs to be added, for example <input type="text" name="color">.

A simple application form, for example, might look something like the one below. (Note: this form will not work unless there is a “formp.php” file, which is stated in the action attribute of the form tag, to handle the submitted data). You should also note the use of table tags for proper placing of HTML form elements though in other examples in this chapter tables will not be used to reduce the size of the code.

```
<TABLE BORDER="0" ALIGN="CENTER" WIDTH="30%">
<CAPTION>APPLICATION FORM</CAPTION>
<FORM ACTION="formp.php" METHOD="POST">

<TD ALIGN="RIGHT"><LABEL>NAME</LABEL></TD>
<TD ALIGN="LEFT"><INPUT TYPE="TEXT" SIZE="20" NAME="name"></TD>
</TR>

<TD ALIGN="RIGHT"><LABEL>GENDER</LABEL></TD>
<TD ALIGN="LEFT">
        <INPUT TYPE="RADIO" NAME="gender" VALUE="male" CHECKED>Male
        <INPUT TYPE="RADIO" NAME="gender" VALUE="female">Female
        </TD>
</TR>

<TD ALIGN="RIGHT"><LABEL>COURSE</LABEL></TD>
<TD ALIGN="LEFT">
        <SELECT SIZE="1" NAME="course">
                <OPTION VALUE="diploma" SELECTED>Diploma</OPTION>
                <OPTION VALUE="certificate">Certificate</OPTION>
        </SELECT>
        </TD>
</TR>

<TD ALIGN="RIGHT">
        <INPUT TYPE="RESET" NAME="reset" VALUE="Clear"></TD>
<TD ALIGN="LEFT">
        <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Apply">
        </TD>
</TR>
</FORM>
</TABLE>
```

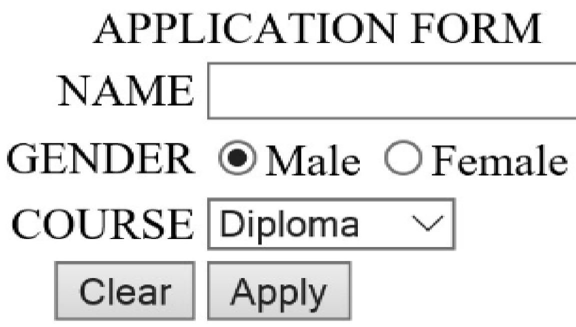


Figure 3.1: application form

3.2 Getting the form data

The previous part of this series explained how to create the HTML part of a web form (the client side). In order to make the form useful, we need to add server side processing support to the form.

There are scripting languages like PHP, ASP and Perl that can be used to write the server side form processing script. You have to find out which of them are supported on your web server. PHP is supported on almost all platforms. Due to those advantages, PHP has been taken onboard throughout of this manual.

You have already created a form that is displayed in figure 3.1 and now let us add some PHP to process this form:

```
<?php
 if($_POST['submit'] == "submit")
 {
   $name = $_POST['name'];
  $gender = $_POST['gender'];
   $course = $_POST['course'];
 }
?>

APPLICATION FORM
<FORM ACTION="formp.php" METHOD="POST">
NAME: <INPUT TYPE="TEXT" SIZE="20" NAME="name"><BR>
GENDER:
<INPUT TYPE="RADIO" NAME="gender" VALUE="male" CHECKED>Male
<INPUT TYPE="RADIO" NAME="gender" VALUE="female">Female <BR>
COURSE
        <SELECT SIZE="1" NAME="course">
                <OPTION VALUE="diploma" SELECTED>Diploma</OPTION>
                <OPTION VALUE="certificate">Certificate</OPTION>
        </SELECT>
        <INPUT TYPE="RESET" NAME="reset" VALUE="Clear">
        <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Apply">
</FORM>
```

First, remember that this form "submits to itself", meaning that the form variables will be sent to this page, and the page will load again from the top.

So, the first thing we should do is to check and see if a form was submitted or not. To do that, let's look at the value of the "submit" button. If its value is "submit", then we know a form has been submitted. If not, then the user is probably visiting this page for the first time. To access this value, use the$_POST['inputname'] array, where "inputname" is the name in the HTML element. We use "$_POST" since that is the form's method. If it was "get" instead, we'd use $_GET[].
Second, after we've determined that the form was submitted, let's get the value that the user typed or selected. Again, we use the $_POST array to get the value and put it into the respective variable.

3.3 Validating the input by using JavaScript

Validating form input with JavaScript is easy to do and can save a lot of unnecessary calls to the server as all processing is handled by the web browser. It can prevent people from leaving fields blank, from entering too little or too much or from using invalid characters.
When form input is important, it should always be verified using a secure server-side script. Otherwise a browser with JavaScript disabled, or a hacker trying to compromise your site, can easily submit invalid data.

3.3.1 Restricting input to alphanumeric characters

In the following example, the single input box, inputfield, must: a) not be empty; and b) contain only alphanumeric characters and spaces. Only if both tests are passed can the form be submitted (when the script returns a value of true).

```
<script type="text/javascript">
 function checkForm(form)
 {
  // validation fails if the input is blank
  if(form.inputfield.value == "") {
   alert("Error: Input is empty!");
   form.inputfield.focus();
return false
   }
  // regular expression to match only alphanumeric characters and spaces
  var re = /^[\w ]+$/;
  // validation fails if the input doesn't match our regular expression
  if(!re.test(form.inputfield.value)) {
   alert("Error: Input contains invalid characters!");
   form.inputfield.focus();
   return false;
  }
  // validation was successful
```

```
    return true;
  }
</script>
```

The pre-defined class \w represents any alphanumeric character or the '_' underscore.
The regular expression ^[\w]+$ will fail if the input is empty as it requires at least one character (because we used + instead of *). The first test in the example is therefore only necessary in order to provide a different error message when the input is blank.
The purpose of a form validation script is to return a boolean value (true or false) to the onsubmit event handler. A value of true means that form will be submitted while a false value will block the form from being submitting. The focus() command is used to set the focus to the problem element.
You can test the above script with different input values using this form:

Figure 3.2: name text box

The form is put together as follows:

```
<form method="POST" action="form-handler" onsubmit="return checkForm(this);">
<p>Name: <input type="text" size="32" name="inputfield">
<input type="submit" value="send"></p>
</form>
```

The name attribute of the input field is used to reference that field from within the checkForm function. With the advent of DHTML it's tempting to use id's to reference form fields, but that can lead to namespace conflicts and why make things more complicated than necessary.
When the form is submitted - either by hitting Enter or clicking on the Submit button - the onsubmithandler is triggered. This then calls our checkForm() function, passing a reference to itself (the form) as the only variable. This makes the value of the input box available within the function asform.input.value (the 'value' of the field called 'input' belonging to the form).
Other form values are available using a similar syntax, although this becomes more complicated if you're using SELECT lists, checkboxes or radio buttons (see below for examples).
The checkForm function tests the form input against our conditions, returning a value of true if the form is to be submitted (when all tests have been passed) or false to abort (cancel) the form submission. It's that simple.
In a real-life situation you will most likely have more fields to check, and more complicated conditions, but the principle remains the same. All you need to do is extend the checkForm function to encompass the new fields and conditions:

```
<script type="text/javascript">
```

```
      function checkForm(form)
 {
   if(!condition1) {
     alert("Error: error message");
     form.fieldname.focus();
     return false;
   }
   if(!condition2) {
     alert("Error: error message");
     form.fieldname.focus();
     return false;
   }
   ...
   return true;

    }


   </script>
```

When a return command is encountered, execution of the function is halted. In other words if the first condition fails, the second condition will not be tested and so forth. Only when all conditions have been satisfied do we reach the return true command, in which case the form will be submitted.
Note: Most modern browsers now support HTML5 Form Validation making it possible to validate form elements without (or before) any JavaScript is triggered.

3.3.2 Working with different types of FORM elements

Text/Textarea/Password boxes
The value of a text input box (or a textarea or password input) is available using the syntaxform.fieldname.value. This is not the case for other input types.
form.fieldname.value
To check whether two inputs have the same value is quite simple:

```
if(form.field1.value == form.field2.value) {
  // values are identical
}
```

Make sure to always use == for comparisons. If you use = (the assignment operator) instead then it can take a long time to debug.
and to see if they have different values we just reverse the logic:

```
if(form.field1.value != form.field2.value) {
  // values are different
}
```

If you want to test numeric values (or add or subtract them) then you first have to convert them from strings to numbers. By default all form values are accessed as as strings.

```
  var field1 = parseInt(form.field1.value);
  var field2 = parseInt(form.field2.value);
  if(field1 > field2) {
    // field1 as a number is greater than field2 as a number
  }
```

parseFloat is the same as parseInt except that it also works for floating point numbers.

Select/Combo/Drop-down boxes

The value of a SELECT input element is accessed using:

```
var selectBox = form.fieldname;
var selectedValue = selectBox.options[selectBox.selectedIndex].value
var selectedText = selectBox.options[selectBox.selectedIndex].text
```

where fieldname is the SELECT element, which has an array of options and a value selectedIndex that tells you which option has been selected. The illustration in figure 3.3 shows this relationship:

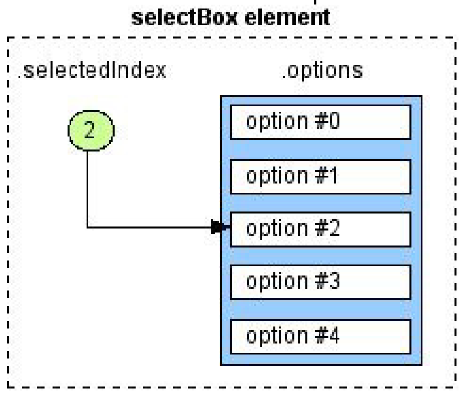


Figure 3.3: select element's array options

Note that the 'I' in selectedIndex needs to be capitalised - JavaScript functions and variables arealways case-sensitive.

If you define a value for the OPTION elements in your SELECT list, then .value will return that, while.text will return the text that is visible in the browser. Here's an example of what this refers to:

```
 <option value="value">text</option>
```

If you just want to check that an option has been chosen (ie. that the SELECT box is no longer in it's default state) then you can use:

```
 if(form.fieldname.selectedIndex > 0) {
   // an option has been selected
 } else {
   // no option selected
 }
```

Checkboxes

These really are simple:

form.checkboxfield.checked

will return a boolean value (true or false) indicating whether the checkbox is in a 'checked' state.

```
 function checkForm(form)
 {
   if(form.checkboxfield.checked) {
     alert("The checkbox IS checked");
   } else {
     alert("The checkbox IS NOT checked");
   }
   return false;
 }
```

You don't need to test using form.checkboxfield.checked == true as the value is already boolean.

Radio buttons

Radio buttons are implemented as if they were an array of checkboxes. To find out which value (if any) has been selected, you need to loop through the array until you find which one has been selected:

```
<script type="text/javascript">

function checkRadio(field)
{
  for(var i=0; i < field.length; i++) {
    if(field[i].checked) return field[i].value;
  }
  return false;
}

</script>
```

The form handler function is then the following:

```
  function checkForm(form)
  {
    if(radioValue = checkRadio(form.radiofield)) {
      alert("You selected " + radioValue);
      return true;
    } else {
      alert("Error: No value was selected!");
      return false;
    }

      }
```

Figure 3.4: radio buttons

Bottom of Form

Actually, there is a bug in the above function in that it doesn't handle the case when there is only a single radio button. In that case field.length is undefined and the function will always return false. Below is a patched version:

```
function checkRadio(field)
{
  if((typeof field.length == "undefined") && (field.type == "radio")) {
    if(field.checked) return field.value;
  } else {
    for(var i=0; i < field.length; i++) {
      if(field[i].checked) return field[i].value;
    }
  }
  return false;
}
```

In the case of a single radio button we have nothing to loop through so just return either it's value, if the radio button is checked, or false.

3.4 Form Validation with HTML5

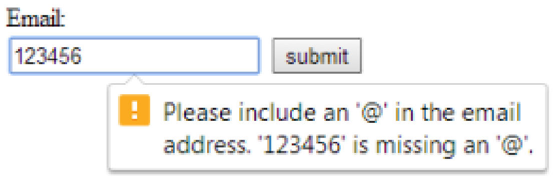
HTML5 includes a fairly solid form validation mechanism powered by the following `<input />` attributes: `type` and `require`. Thanks to these new attributes in HTML5, because validation can now be done without having to write complex JavaScript validation code.

Let's examine these new form attributes to see how they can aid form validation.

3.4.1 The type Attribute

This form attribute indicates what kind of input control to display such as the popular `<input type="text" />` for handling simple text data.

Some form controls inherit validation systems without having to write any code. For example, `<input type="email" />` validates the field to ensure the entered data is in fact a valid email address. If the field contains an invalid value, the form cannot be submitted for processing until it is corrected.



Please include an '@' in the email address. '123456' is missing an '@'.

Try the demo below by entering an invalid email:

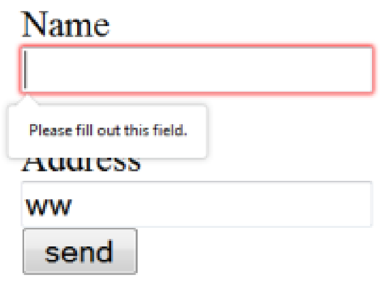
There is also `<input type="number" />`, `<input type="url" />` and `<input type="tel" />` for validating numbers, URLs, and telephone numbers respectively.

3.4.2 The required Attribute

This is a Boolean attribute used to indicate that a given input field's value is required in order to submit the form. By adding this attribute to a form field, the browser requires the user to enter data into that field before submitting the form.

This replaces the basic form validation currently implemented with JavaScript, making things a little more usable and saving us some development time.

Example: `<input type="text" name="my_name" required />` or `<input type="text" name="my_name" required="required" />` for XHTML compatibility.



All the demos embedded above use the `required` attribute, so you can test those by trying to submit any of the forms without entering anything in the field.

Browser support for form validation features is pretty strong, and you can easily polyfill them where necessary.

It is worth noting that relying solely on the browser (client-side) for validation can be dangerous because it can be circumvented by a malicious user or by computer bots.

Not all browsers support HTML5, and not all input sent to your script will come from the form. This means that server-side side validation should also be in place before the form data is sent to the server for processing.

3.5 Validating the input by using PHP Script

Validations can be done with simple PHP if statements. Take for example the below form which was presented figure 3.1

```
APPLICATION FORM
<FORM ACTION="formp.php" METHOD="POST">
NAME: <INPUT TYPE="TEXT" SIZE="20" NAME="name"><BR>
GENDER:
<INPUT TYPE="RADIO" NAME="gender" VALUE="male" CHECKED>Male
<INPUT TYPE="RADIO" NAME="gender" VALUE="female">Female <BR>
COURSE
        <SELECT SIZE="1" NAME="course">
                <OPTION VALUE="diploma" SELECTED>Diploma</OPTION>
                <OPTION VALUE="certificate">Certificate</OPTION>
        </SELECT>
        <INPUT TYPE="RESET" NAME="reset" VALUE="Clear">
        <INPUT TYPE="SUBMIT" NAME="submit" VALUE="Apply">
</FORM>
```

suppose on buldiing the processing file formp.php you want to validate whether user has typed his/her name

```
<?php
if($_POST['name'] == "" or !isSet($_POST['name']))
{
        echo "<p>There was an error with your form:</p>";
        echo "You forgot to enter a name!";
        include "formp.php";
}
else
{
        $name = $_POST['name'];
        $gender = $_POST['gender'];
   $course = $_POST['course'];
        echo "name: $name <br> gender: $gender <br> course: $course";
}

?>
```

In this example, the code makes some very basic checks to see that the user typed in something –anything–into the input field. If the user didn't, the check will print an error message and the form for the user to type the required name, otherwise, it will print the values the user selected or typed.

3.6 Understanding Object-Oriented Concepts

Modern programming languages usually support or even require an object-oriented approach to software development. Object-oriented development attempts to use the classifications, relationships, and properties of the objects in the system to aid in program development and code reuse.

3.6.1 Classes and Objects

In the context of OO software, an object can be almost any item or concept—a physical object such as a desk or a customer; or a conceptual object that exists only in software, such as a text input area or a file. Generally, you will be most interested in objects, including both real-world objects and conceptual objects that need to be represented in software.

Object-oriented software is designed and built as a set of self-contained objects with both attributes and operations that interact to meet your needs. *Attributes* are properties or variables that relate to the object. *Operations* are methods, actions, or functions that the object can perform to modify itself or perform for some external effect. (You will hear the term *attribute* used interchangeably with the terms *member variable* and *property*, and the term *operation* used interchangeably with *method*.)

Object-oriented software's central advantage is its capability to support and encourage *encapsulation*—also known as *data hiding*. Essentially, access to the data within an object is available only via the object's operations, known as the *interface* of the object.

An object's functionality is bound to the data it uses. You can easily alter the details controlling how the object is implemented to improve performance, add new features, or fix bugs *without having to change the interface*. Changing the interface could have ripple effects throughout the project, but encapsulation allows you to make changes and fix bugs without your actions cascading to other parts of the project.

In other areas of software development, object orientation is the norm, and procedural or function-oriented software is considered old fashioned. Most web scripts are still designed and written using an *ad hoc* approach following a function-oriented methodology.

A number of reasons for using this approach exist. Many web projects are relatively small and straightforward. You can get away with picking up a saw and building a wooden spice rack without planning your approach, and you can successfully complete the majority of web software projects in the same way because of their small size. However, if you picked up a saw and attempted to build a house without formal planning, you wouldn't get quality results, if you got results at all. The same is true for large software projects.

Many web projects evolve from a set of hyperlinked pages to a complex application. Complex applications, whether presented via dialog boxes and windows or via dynamically generated HTML pages, need a properly thought-out development methodology.

Object orientation can help you to manage the complexity in your projects, increase code reusability, and thereby reduce maintenance costs. In OO software, an object is a unique and identifiable collection of stored data and operations that operate on that data. For instance, you might have two objects that represent buttons. Even if both have a label "OK", a width of 60 pixels, a height of 20 pixels, and any other attributes that are identical, you still need to be able to deal with one button or the other. In software, separate variables act as *handles* (unique identifiers) for the objects.

Objects can be grouped into classes. Classes represent a set of objects that might vary from individual to individual, but must have a certain amount in common. A class contains objects that all have the same operations behaving in the same way and the same attributes representing the same things, although the values of those attributes vary from object to object. You can think of the noun *bicycle* as a class of objects describing many distinct bicycles with many common features or *attributes*—such as two wheels, a color, and a size—and operations, such as move.

My own bicycle can be thought of as an object that fits into the class bicycle. It has all the common features of all bicycles, including a move operation that behaves the same as most other bicycles' move—even if it is used more rarely. My bicycle's attributes have unique values because my bicycle is green, and not all bicycles are that color.

3.6.2 Polymorphism

An object-oriented programming language must support polymorphism, which means that different classes can have different behaviors for the same operation. If, for instance, you have a class car and a class bicycle, both can have different move operations. For real world objects, this would rarely be a problem. Bicycles are not likely to become confused and start using a car's move operation instead. However, a programming language does not possess the common sense of the real world, so the language must support polymorphism to know which move operation to use on a particular object.

Polymorphism is more a characteristic of behaviors than it is of objects. In PHP, only member functions of a class can be polymorphic. A real-world comparison is that of verbs in natural languages, which are equivalent to member functions. Consider the ways a bicycle can be used in real life. You can clean it, move it, disassemble it, repair it, or paint it, among other things. These verbs describe generic actions because you don't know what kind of object is being acted on. (This type of abstraction of objects and actions is one of the distinguishing characteristics of human intelligence.)

For example, moving a bicycle requires completely different actions from those required for moving a car, even though the concepts are similar. The verb *move* can be associated with a particular set of actions only after the object acted on is made known.

3.6.3 Inheritance

Inheritance allows you to create a hierarchical relationship between classes using subclasses. A subclass inherits attributes and operations from its superclass. For example, car and bicycle have some things in common. You could use a class vehicle to contain the things such as a color attribute and a move operation that all vehicles have, and then let the car and bicycle classes inherit from vehicle.

You will hear *subclass*, *derived class*, and *child* used interchangeably. Similarly, you will hear *superclass* and *parent* used interchangeably. With inheritance, you can build on and add to existing classes. From a simple base class, you can derive more complex and specialized classes as the need arises. This capability makes your code more reusable, which is one of the important advantages of an object-oriented approach.

Using inheritance might save you work if operations can be written once in a superclass rather than many times in separate subclasses. It might also allow you to more accurately model real-world relationships. If a sentence about two classes makes sense with "is a" between the classes, inheritance is probably appropriate. The sentence "a car is a vehicle" makes sense, but the sentence "a vehicle is a car" does not make sense because not all vehicles are cars. Therefore, car can inherit from vehicle.

3.7 Creating Classes, Attributes, and Operations in PHP

So far, we have discussed classes in a fairly abstract way. When creating a class in PHP, you must use the keyword class.

3.7.1 Structure of a Class

A minimal class definition looks like this:

```
class classname
 {
 }
```

To be useful, the classes need attributes and operations. You create attributes by declaring variables within a class definition using the keyword var. The following code creates a class called classname with two attributes, $attribute1 and $attribute2:

```
class classname
{
 var $attribute1;
 var $attribute2;
}
```

You create operations by declaring functions within the class definition. The following code creates a class named classname with two operations that do nothing. The operation operation1() takes no parameters and operation2() takes two parameters:

```
class classname
 {
 function operation1()
 {
  }
  function operation2($param1, $param2)
  {
  }
 }
```

3.7.2 Constructors

Most classes have a special type of operation called a constructor. A constructor is called when an object is created, and it also normally performs useful initialization tasks such as setting attributes to sensible starting values or creating other objects needed by this object. A constructor is declared in the same way as other operations, but has the special name __construct(). This is a change in PHP5. In previous versions, constructor functions had the same name as the class. For backward compatibility, if no function called __construct() is found in a class, PHP will search for a function with the same name as the class.

Although you can manually call the constructor, its main purpose is to be called automatically when an object is created. The following code declares a class with a constructor:

```
class classname
 {
 function __construct($param)
 {
 echo "Constructor called with parameter $param <br />";
 }
 }
```

PHP5 now supports function overloading, which means that you can provide more than one function with the same name and different numbers or types of parameters. (This feature is supported in many OO languages.) We will discuss this later in this chapter.

3.7.3 Destructors

The opposite of a constructor is a destructor. Destructors are new in PHP5. They allow you to have some functionality that will be executed just before a class is destroyed, which will occur automatically when all references to a class have been unset or fallen out of scope.

Similar to the way constructors are named, the destructor for a class must be named __destruct(). Destructors cannot take parameters.

3.7.4 Instantiating Classes

After you have declared a class, you need to create an object—a particular individual that is a member of the class—to work with. This is also known as creating an instance of or instantiating a class. You create an object by using the new keyword. When you do so, you need to specify what class your object will be an instance of and provide any parameters required by the constructor.

The following code declares a class called classname with a constructor and then creates three objects of type classname:

```
class classname
  {
  function __construct($param)
  {
  echo "Constructor called with parameter $param <br />";
  }
  }
  $a = new classname('First');
  $b = new classname('Second');
  $c = new classname();
```

Because the constructor is called each time you create an object, this code produces the following output:

```
Constructor called with parameter First
Constructor called with parameter Second
Constructor called with parameter
```

3.7.5 Using Class Attributes

Within a class, you have access to a special pointer called $this. If an attribute of your current class is called $attribute, you refer to it as $this->attribute when either setting or accessing the variable from an operation within the class.

The following code demonstrates setting and accessing an attribute within a class:

```
class classname
     {
     var $attribute;
     function operation($param)
     {
     $this->attribute = $param
     echo $this->attribute;
     }
     }
```

Whether you can access an attribute from outside the class is determined by access modifiers, discussed later in this chapter. This example does not restrict access to the attributes, so you can access them from outside the class as follows:

```
class classname
  {
  var $attribute;
  }
 $a = new classname();
 $a->attribute = 'value';
 echo $a->attribute;
```

It is not generally a good idea to directly access attributes from outside a class. One of the advantages of an object-oriented approach is that it encourages encapsulation. You can enforce this with the use of __get and __set functions. If, instead of accessing the attributes of a class directly, you write accessor functions, you can make all your accesses through a single section of code. When you initially write your accessor functions, they might look as follows:

```
        class classname
          {
          var $attribute;
          function __get($name)
          {
          return $this->$name;
          }
          function __set ($name, $value)
          {
          $this->$name = $value;
          }
          }
```

This code provides minimal functions to access the attribute named $attribute. The function named __get() simply returns the value of $attribute, and the function named __set() assigns a new value to $attribute. Note that __get() takes one parameter—the name of an attribute—and returns the value of that attribute. Similarly, the __set() function takes two parameters: the name of an attribute and the value shows that these functions have a special meaning in PHP, just like the __construct() and __destruct() functions.

How then do they work? If you instantiate the class $a = new classname(); you can then use the __get() and __set() functions to check and set the value of any attributes.

`If you type $a->$attribute = 5;`

This statement implicitly calls the __set() function with the value of $name set to "attribute", and the value of $value set to 5. You need to write the __set() function to do any error checking you want.

The __get() function works in a similar way. If, in your code, you reference `$a->attribute`

This expression implicitly calls the __get() function with the parameter $name set to "attribute". It is up to you to write the __get() function to return the value.

At first glance, this code might seem to add little or no value. In its present form, this is probably true, but the reason for providing accessor functions is simple: You then have only one section of code that accesses that particular attribute.

With only a single access point, you can implement validity checks to make sure that only sensible data is being stored. If it occurs to you later that the value of $attribute should only be between 0 and 100, you can add a few lines of code once and check before allowing changes. You could change the __set() function to look as follows:

```
function __set ($name, $value)
 {
 if( $name='attribute' && $value >= 0 && $value <= 100 )
 $this->attribute = $value;
}
```

With only a single access point, you are free to change the underlying implementation.
If, for some reason, you choose to change the way $attribute is stored, accessor functions allow you to do this and change the code in only one place.

You might decide that, instead of storing $attribute as a variable, you will retrieve it from a database only when needed, calculate an up-to-date value every time it is requested, infer a value from the values of other attributes, or encode the data as a smaller data type. Whatever change you decide to make, you can simply modify the accessor functions. Other sections of code will not be affected as long as you make the accessor functions still accept or return the data that other parts of the program expect.

3.7.6 Controlling Access with private and public

The access modifiers were introduced from PHP5. They control the visibility of attributes and methods, and are placed in front of attribute and method declarations. PHP5 and later versions support the following three different access modifiers:

(i) The default option is public, meaning that if you do not specify an access modifier for an attribute or method, it will be public. Items that are public can be accessed from inside or outside the class.

(ii) The private access modifier means that the marked item can be accessed only from inside the class. You might use it on all attributes if you are not using __get() and __set().You may also choose to make some methods private, for example, if they are utility functions for use inside the class only. Items that are private will not be inherited..

(iii) The protected access modifier means that the marked item can be accessed only from inside the class. It also exists in any subclasses; again, we return to this issue when we discuss inheritance later in this chapter. For now, you can think of protected as being halfway in between private and public.

To add access modifiers to the sample class, you can alter the code as follows:

```
class classname
      {
      public $attribute;
      public function __get($name)
      {
      return $this->$name;
      }
      public function __set ($name, $value)
      {
      $this->$name = $value;
      }
      }
```

Here, each class member is prefaced with an access modifier to show whether it is private or public. You could leave out the public keyword because it is the default, but the code is easier to understand with it in if you are using the other modifiers. Notice that we removed the var keyword in front of the attribute; it is replaced with the access modifier public. In this case, we made everything public.

3.7.7 Calling Class Operations

You can call class operations in much the same way that you call class attributes. Say you have the class

```
class classname
      {
      function operation1()
      {
      }
      function operation2($param1, $param2)
      {
      }
      }
```

and create an object of type classname called $a as follows:

```
$a = new classname();
```

You then call operations the same way that you call other functions: by using their name and placing any parameters that they need in brackets. Because these operations belong to an object rather than normal functions, you need to specify to which object they belong. The object name is used in the same way as an object's attributes, as follows:

```
$a->operation1();
 $a->operation2(12, ‘test’);
```

If the operations return something, you can capture that return data as follows:

```
$x = $a->operation1();
$y = $a->operation2(12, ‘test’);
```

5.8 Implementing Inheritance in PHP

If the class is to be a subclass of another, you can use the extends keyword to specify this use. The following code creates a class named B that inherits from some previously defined class named A:

```
class B extends A
     {
     var $attribute2;
     function operation2()
     {
     }
     }
```

If the class A was declared as

```
class A
     {
     var $attribute1;
     function operation1()
     {
     }
     }
```

all the following accesses to operations and attributes of an object of type B would be valid:

```
$b = new B();
  $b->operation1();
  $b->attribute1 = 10;
  $b->operation2();
  $b->attribute2 = 10;
```

Note that because class B extends class A, you can refer to operation1() and $attribute1, although they were declared in class A. As a subclass of A, B has all the same functionality and data. In addition, B has declared an attribute and an operation of its own.

It is important to note that inheritance works in only one direction. The subclass or child inherits features from its parent or superclass, but the parent does not take on features of the child. This means that the last two lines in this code are wrong:

```
$a = new A();
  $a->operation1();
  $a->attribute1 = 10;
  $a->operation2();
  $a->attribute2 = 10;
```

The class A does not have an operation2() or an attribute2.

3.8.1 Controlling Visibility through Inheritance with private and protected

You can use the access modifiers private and protected to control what is inherited. If an attribute or method is specified as private, it will not be inherited. If an attribute or method is specified as protected, it will not be visible outside the class (like a private element) but will be inherited.

Consider the following example:

```
<?php
  class A
  {
  private function operation1()
  {
  echo "operation1 called";
  }
  protected function operation2()
  {
  echo "operation2 called";
  }
  public function operation3()
  {
  echo "operation3 called";
  }
  }
  class B extends A
  {
  function __construct()
  {
  $this->operation1();
  $this->operation2();
  $this->operation3();
  }
  }
  $b = new B;
 ?>
```

This code creates one operation of each type in class A: public, protected, and private. B inherits from A. In the constructor of B, you then try to call the operations from the parent.

The line $this->operation1(); produces a fatal error as follows:

```
Fatal error: Call to private method A::operation1() from context 'B'
```

This example shows that private operations cannot be called from a child class. If you comment out this line, the other two function calls will work. The protected function is inherited but can be used only from inside the child class, as done here. If you try adding the line $b->operation2(); to the bottom of the file, you will get the following error:

```
Fatal error: Call to protected method A::operation2() from context ''
```

However, you can call operation3() from outside the class, as follows:

```
    $b->operation3();
```

You can make this call because it is declared as public.

3.8.2 Overriding

In this chapter, we have shown a subclass declaring new attributes and operations. It is also valid and sometimes useful to redeclare the same attributes and operations. You might do this to give an attribute in the subclass a different default value to the same attribute in its superclass or to give an operation in the subclass different functionality to the same operation in its superclass. This action is called overriding.

For instance, say you have a class A:

```
    class A
         {
         var $attribute = 'default value';
         function operation()
         {
         echo 'Something<br />';
         echo "The value of \$attribute is $this->attribute<br />";
         }
```

```
        }
```

If you want to alter the default value of $attribute and provide new functionality for operation(), you can create the following class B, which overrides $attribute and operation():

```
class B extends A
      {
      var $attribute = 'different value';
      function operation()
      {
      echo 'Something else<br />';
      echo "The value of \$attribute is $this->attribute<br />";
      }
      }
```

Declaring B does not affect the original definition of A. Now consider the following two lines of code:

```
$a = new A();
  $a -> operation();
```

These lines create an object of type A and call its operation() function. This produces

```
Something
     The value of $attribute is default value
```

Proving that creating B has not altered A. If you create an object of type B, you will get different output.

This code

```
$b = new B();
 $b -> operation();
```

produces

```
Something else
  The value of $attribute is different value
```

In the same way that providing new attributes or operations in a subclass does not affect the superclass, overriding attributes or operations in a subclass does not affect the superclass. A subclass will inherit all the attributes and operations of its superclass, unless you provide replacements. If you provide a replacement definition, it takes precedence and overrides the original definition.

The parent keyword allows you to call the original version of the operation in the parent class. For example, to call A::operation from within class B, you would use

```
parent::operation();
```

The output produced is, however, different. Although you call the operation from the parent class, PHP uses the attribute values from the current class. Hence, you get the following output:

```
Something
     The value of $attribute is different value
```

Inheritance can be many layers deep. You can declare a class imaginatively called C that extends B and therefore inherits features from B and from B's parent, A. The class C can again choose which attributes and operations from its parents to override and replace.

3.8.3 Preventing Inheritance and Overriding with final

PHP5 introduces the keyword final. When you use this keyword in front of a function declaration, that function cannot be overridden in any subclasses. For example, you can add it to class A in the previous example, as follows:

```
class A
      {
      var $attribute = 'default value';
      final function operation()
      {
      echo 'Something<br />';
      echo "The value of \$attribute is $this->attribute<br />";
      }
      }
```

Using this approach prevents you from overriding operation() in class B. If you attempt to do so, you will get the following error:

```
Fatal error: Cannot override final method A::operation()
```

You can also use the final keyword to prevent a class from being subclassed at all. To prevent class A from being subclassed, you can add it as follows:

```
final class A
      {...}
```

If you then try to inherit from A, you will get an error similar to

```
Fatal error: Class B may not inherit from final class (A)
```

Exercises

1. What is the difference between get and post?
2. What is does action attribute do in a form tag?
3. Why does a form element need a name?
4. Create a simple student application form for UCC diploma in CIT program and validate its user inputs by using PHP script.
5. How would you declare a class called emptyClass that has no methods or properties?
6. How would you choose a name for a constructor method?
7. Create a class called baseCalc() that stores two numbers as properties. Next, create a calculate() method that prints the numbers to the browser. Finally, create classes called addCalc(), subCalc(), mulCalc(), and divCalc() that inherit functionality from baseCalc() but override the calculate() method and print appropriate totals to the browser.

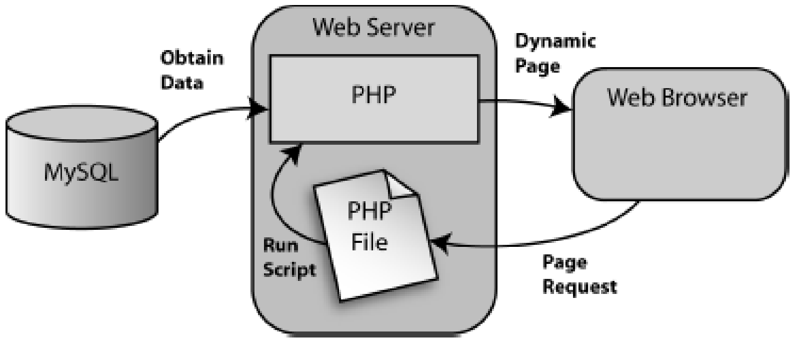
Chapter 4: Interacting with MySQL Using PHP

Now that you've learned the basics of PHP as well as the basics of working with MySQL, you're ready to make the two interact. Think of PHP as a conduit to MySQL, the commands you learned in the first chapter are the same commands that you will send to MySQL in this chapter, only this time you'll send them with PHP. In this chapter, you will learn

- How to connect to MySQL using PHP
- How to insert and select data through PHP scripts

Before we leap forward, it's worth a brief look back to remind you of our ultimate goal. We have two powerful tools at our disposal: the PHP scripting language, and the MySQL database engine. It's important to understand how these will fit together. The whole idea of a database-driven Website is to allow the content of the site to reside in a database, and for that content to be pulled from the database dynamically to create Web pages for people to view with a regular Web browser.

So, on one end of the system you have a visitor to your site who uses a Web browser to request a page, and expects to receive a standard HTML document. On the other end you have the content of your site, which sits in one or more tables in a MySQL database that understands only how to respond to SQL queries (commands).



As shown in the figure above, the PHP scripting language is the go-between that speaks both languages. It processes the page request and fetches the data from the MySQL database, then spits it out dynamically as the nicely-formatted HTML page that the browser expects.

This is what will happen when someone visits a page on your database-driven Website:

1. The visitor's Web browser requests the Web page using a standard URL.
2. The Web server software (Apache, IIS, or whatever) recognizes that the requested file is a PHP script, so the server interprets the file using its PHP plug-in before responding to the page request.
3. Certain PHP commands (which you have yet to learn) connect to the MySQL database and request the content that belongs in the Web page.
4. The MySQL database responds by sending the requested content to the PHP script.
5. The PHP script stores the content into one or more PHP variables, then uses the now-familiar echo statement to output the content as part of the Web page.

6. The PHP plug-in finishes up by handing a copy of the HTML it has created to the Web server.
7. The Web server sends the HTML to the Web browser as it would a plain HTML file, except that instead of coming directly from an HTML file, the page is the output provided by the PHP plug-in.

4.1 Formatting Strings for Database use

You often need to tidy up user strings (typically from an HTML form interface) before you can use them. This sections describe some of the functions you can use.

4.1.1 Trimming Strings:

The first step in tidying up is to trim any excess whitespace from the string. Although this step is never compulsory, it can be useful if you are going to store the string in a file or database, or if you're going to compare it to other strings. PHP provides two useful functions for this purpose which trim() and chop().

You can use the trim() function to tidy up your input data as follows:

```
    $name=trim($name);
$email=trim($email);
$feedback=trim($feedback);
```

The trim() function strips whitespace from the start and end of a string and returns the resulting string. The characters it strips by default are newlines and carriage returns (\n and \r), horizontal and vertical tabs (\t and \x0B), end-of-string characters (\0), and spaces. You can also pass it a second parameter containing a list of characters to strip instead of this default list. Depending on your particular purpose, you might like to use the ltrim() or rtrim() functions instead. They are both similar to trim(), taking the string in question as a parameter and returning the formatted string. The difference between these three is that trim() removes whitespace from the start and end of a string, ltrim() removes whitespace from the start (or left) only, and rtrim() removes whitespace from the end (or right) only.

4.1.2 Formatting Strings for database Storage

Certain characters are perfectly valid as part of a string but can cause problems, particularly when you are inserting data into a database because the database could interpret these characters as control characters. The problematic ones are quotation marks (single and double), backslashes (\), and the NULL character.

You need to find a way of marking or *escaping* these characters so that databases such as MySQL can understand that you meant a literal special character rather than a control sequence. To *escape* these characters, add a backslash in front of them. For example,

"(double quotation mark) becomes \"(backslash double quotation mark), and \(backslash) becomes \\(backslash backslash). (This rule applies universally to special characters, so if you have \\ in your string, you need to replace it with \\\\.)

PHP provides two functions specifically designed for escaping characters. Before you write any strings into a database, you should reformat them with addslashes(), as follows:

```
$feedback  = addslashes($feedback);
```

Like many of the other string functions, addslashes() takes a string as a parameter and returns the reformatted string.

so if the customer feedback was "Why you don't provide any guarantee ?"
customer feedaback before addslashes() would be :

```
Why you don't provide any guarantee?
```

customer feedaback after addslashes() would be :

```
Why you don\'t provide any guarantee?
```

before displaying this data you must remove slashes with stripslashes() function

```
$feedback  = stripslashes($feedback);
print $feedback
```

so, customer feedaback before stripslashes() would be :

```
Why you don\'t provide any guarantee?
```

customer feedaback after stripslashes() would be :

```
Why you don't provide any guarantee?
```

4.1.3 Using Magic Quotes

We have already looked at the use of addslashes() and stripslashes() that escape out any single quotation mark, double quotation mark, backslash, and NULL characters.

PHP has a useful capability to automatically or magically add and strip slashes for you. With two settings in your php.ini file, you can turn on or off magic quoting for GET, POST, cookie data, and other sources.

The value of the magic_quotes_gpc directive controls whether magic quoting is used for GET, POST, and cookie operations. The letters *gpc* stand for *GET, POST,* and *cookie*.This means that variables coming from these sources are automatically quoted.

With magic_quotes_gpc on, if somebody typed “Bob’s Auto Parts” into a form on your site, your script would receive “Bob\’s Auto Parts” because the quote is escaped for you. This behavior can be very handy, but you need to know that it is happening so you can remember to remove the slashes before echoing the data back to your users.

So before applying addslashes() function to the data submitted by user you must check first if magic_quotes_gpc is on to avoid characters been escaped twice.

The function get_magic_quotes_gpc() returns either 1(true) or 0 (false) , telling you the current value of magic_quotes_gpc. This is most useful for testing if you need to use addslashes() and stripslashes() on data received from the user. The value of magic_quotes_runtime controls whether magic quoting is used by functions that get data from databases and files. To get the value of magic_quotes_runtime, use the function get_magic_quotes_runtime(). This function returns either 1 or 0. Magic quoting can be turned on for a particular script using the function set_magic_quotes_runtime(). By default, magic_quotes_gpc is on and magic_quotes_runtime is off

Example

```
    if  (!get_magic_quotes_gpc()) //check if is set
     {
      $feedback  = addslashes($feedback);
     }
```

You also use stripslashes() on the data coming back from the database. If the magic quotes feature is turned on, the data will have slashes in it when it comes back from the database, so you need to take them out. also you can use htmlspecialchars() function to encode characters that have special meanings in HTML.

```
   if(get_magic_quotes_gpc()) //check if is set
   {
   $feedback  = stripslashes($feedback);
    $feedback  = htmlspecialchars($feedback);
   print $feedback
   }
```

4.2 Managing the Date and Time in Databases

Way back in " Web Technologies," we described using the date() function to get and format the date and time from PHP. Here, we going to discuss how to check and format the date and time and converting between date formats. These capabilities are especially important when you are converting between MySQL and PHP date formats, Unix and PHP date formats, and dates entered by the user in an HTML form.

4.2.1 Using the date() Function

As you might recall, the date() function takes two parameters, one of them optional. The first one is a format string, and the second, optional one is a Unix timestamp. If you don't specify a timestamp, date() will default to the current date and time. It returns a formatted string representing the appropriate date.

A typical call to the date() function could be

```
echo  date('jS F Y');
```

This call produces a date in the format 29th October 2000. The format codes accepted by date() are listed in Table below

Format Codes for PHP's date() **Function**

Code	Description
a	Morning or afternoon, represented as two lowercase characters, either am or pm..
A	Morning or afternoon, represented as two uppercase characters, either AM

Code	Description
	or PM.
B	Swatch Internet time, a universal time scheme. More information is available at http://www.swatch.com/.
c	ISO 8601 date. A date is represented as YYYY-MM-DD. An uppercase T separates the date from the time. The time is represented as HH:MM:SS. Finally, the time zone is represented as an offset from Greenwich Mean Time (GMT)—for example, 2004-03-26T21:04:42+11:00. (This format code was added in PHP5.)
d	Day of the month as a two-digit number with a leading zero. The range is from 01 to 31.
D	Day of the week in three-character abbreviated text format. The range is from Mon to Sun.
F	Month of the year in full text format. The range is from January to December.
g	Hour of the day in 12-hour format without leading zeros. The range is from 1 to 12.
G	Hour of the day in 24-hour format without leading zeros. The range is from 0 to 23.
h	Hour of the day in 12-hour format with leading zeros. The range is from 01 to 12.
H	Hour of the day in 24-hour format with leading zeros. The range is from 00 to 23.
i	Minutes past the hour with leading zeros. The range is from 00 to 59.
I	Daylight savings time, represented as a Boolean value. This format code returns 1 if the date is in daylight savings and 0 if it is not.
j	Day of the month as a number without leading zeros. The range is from 1 to 31.
l	Day of the week in full-text format. The range is from Monday to Sunday.
L	Leap year, represented as a Boolean value. This format code returns 1 if the date is in a leap year and 0 if it is not.
m	Month of the year as a two-digit number with leading zeros. The range is from 01 to 12.
M	Month of the year in three-character abbreviated text format. The range is from Jan to Dec.
n	Month of the year as a number without leading zeros. The range is from 1 to 12.
O	Difference between the current time zone and GMT in hours—for example, +1600.
r	RFC822-formatted date and time—for example, Wed, 9 Oct 2002 18:45:30 +1600. (This code was added in PHP 4.0.4.)
s	Seconds past the minute with leading zeros. The range is from 00 to 59.
S	Ordinal suffix for dates in two-character format. It can be st, nd, rd, or th, depending on the number it follows.
t	Total number of days in the date's month. The range is from 28 to 31.
T	Time zone setting of the server in three-character format—for example, EST.
U	Total number of seconds from January 1, 1970, to this time; also known as a *Unix timestamp* for this date.

Code	Description
w	Day of the week as a single digit. The range is from 0 (Sunday) to 6 (Saturday).
W	Week number in the year; ISO-8601 compliant. (This format code was added at PHP 4.1.0.)
y	Year in two-digit format—for example, 05.
Y	Year in four-digit format—for example, 2005.
z	Day of the year as a number. The range is 0 to 365.
Z	Offset for the current time zone in seconds. The range is -43200 to 43200.

4.2.2 Dealing with Unix Timestamps

The second parameter to the date() function is a Unix timestamp. In case you are wondering exactly what this means, most Unix systems store the current time and date as a 32-bit integer containing the number of seconds since midnight, January 1, 1970, GMT, also known as the *Unix Epoch*. This concept can seem a bit esoteric if you are not familiar with it, but it's a standard and integers are easy for computers to deal with.

Unix timestamps are a compact way of storing dates and times, but it is worth noting that they do not suffer from the year 2000 (Y2K) problem that affects some other compact or abbreviated date formats. They do have similar problems, though, because they can represent only a limited span of time using a 32-bit integer. If your software needs to deal with events before 1902 or after 2038, you will be in trouble.

On some systems including Windows, the range is more limited. A timestamp cannot be negative, so timestamps before 1970 cannot be used. To keep your code portable, you should bear this fact in mind.

You probably don't need to worry about your software still being used in 2038. Timestamps do not have a fixed size; they are tied to the size of a C long, which is at least 32 bits. If your software still happens to be in use in 2038, it is exceedingly likely that your compiler will be using a larger type.

Although this is a standard Unix convention, this format is still used by date() and a number of other PHP functions even if you are running PHP under Windows. The only difference is that, for Windows, the timestamp must be positive.

If you want to convert a date and time *to* a Unix timestamp, you can use the mktime() function. It has the following prototype:

```
int  mktime ([int hour[, int minute[, int second[, int month[,
 int day[, int year [, int is_dst]]]]]]])
```

The parameters are fairly self-explanatory, with the exception of the last one, *is_dst*, which represents whether the date was in daylight savings time. You can set this parameter to 1 if it was, 0 if it wasn't, or -1 (the default value) if you don't know. This parameter is optional, so you will rarely use it anyway.

The main trap to avoid with this function is that the parameters are in a fairly unintuitive order. The ordering doesn't lend itself to leaving out the time. If you are not worried about the time, you can pass in 0s to the *hour*, *minute*, and *second* parameters. You can, however, leave out values from the right side of the parameter list. If you don't provide parameters, they will be set to the current values. Hence, a call such as

```
$timestamp  = mktime();
```

returns the Unix timestamp for the current date and time. You could also get this result by calling

```
$timestamp  = time();
```

The time() function does not take any parameters and always returns the Unix timestamp for the current date and time.

Another option is the date() function, as already discussed. The format string "U" requests a timestamp. The following statement is equivalent to the two previous ones:

```
$timestamp  = date("U");
```

You can pass in a two- or four-digit year to mktime(). Two-digit values from 0 to 69 are interpreted as the years 2000 to 2069, and values from 70 to 99 are interpreted as 1970 to 1999.

Here are some other examples to illustrate the use of mktime():

```
$time  = mktime(12, 0, 0);
```

gives noon on today's date.

```
$time  = mktime(0,0,0,1,1);
```

gives the 1st of January in the current year.

You can also use mktime() for simple date arithmetic. For example,

```
$time  = mktime(12,0,0,$mon,$day+30,$year);
```

adds 30 days to the date specified in the components, even though ($day+30) will usually be bigger than the number of days in that month.

To eliminate some problems with daylight savings time, use hour 12 rather than hour 0. If you add (24 * 60 * 60) to midnight on a 25-hour day, you'll stay on the same day. Add the same number to midday, and it'll give 11am but will at least be the right day.

4.2.3 Using the getdate() Function

Another date-determining function you might find useful is getdate(). This function has the following prototype:

```
array  getdate ([int timestamp])
```

It takes an optional timestamp as a parameter and returns an array representing the parts of that date and time, as shown in table below

Key	**Value**
seconds	Seconds, numeric
minutes	Minutes, numeric
hours	Hours, numeric
mday	Day of the month, numeric
wday	Day of the week, numeric
mon	Month, numeric
year	Year, numeric
yday	Day of the year, numeric
weekday	Day of the week, full-text format
month	Month, full-text format
0	Timestamp, numeric

After you have these parts in an array, you can easily process them into any required format. The 0 element in the array (the timestamp) might seem useless, but if you call getdate() without a parameter, it will give you the current timestamp.

4.2.4 Validating Dates

You can use the checkdate() function to check whether a date is valid. This capability is especially useful for checking user input dates. The checkdate() function has the following prototype:

```
int  checkdate (int month, int day, int year)
```

It checks whether the *year* is a valid integer between 0 and 32,767, whether the *month* is an integer between 1 and 12, and whether the *day*

given exists in that particular month. The function also takes leap years into consideration when working out whether a day is valid.
For example,

```
checkdate(9,  18, 1972)   returns true, whereas

checkdate(9,  31, 2000)    does not.
```

4.2.5 Converting Between PHP and MySQL Date Formats

Dates and times in MySQL are handled in ISO 8601 format. Times work relatively intuitively, but ISO 8601 requires you to enter dates with the year first. For example, you could enter August 29, 2005, either as 2005-08-29 or as 05-08-29.

Dates retrieved from MySQL are also in this format by default. Depending on your intended audience, you might not find this function very user friendly. To communicate between PHP and MySQL, then, you usually need to perform some date conversion. This operation can be performed at either end.

When putting dates into MySQL from PHP, you can easily put them into the correct format by using the date() function, as shown previously. One minor caution if you are creating them from your own code is that you should store the day and month with leading zeros to avoid confusing MySQL. You can use a two-digit year, but using a four-digit year is usually a good idea.

If you want to convert dates or times in MySQL, two useful functions are DATE_FORMAT() and UNIX_TIMESTAMP(). The DATE_FORMAT() function works similarly to the PHP function but uses different formatting codes. The most common thing you want to do is format a date in normal American format (MM-DD-YYYY) rather than in the ISO format (YYYY-MM-DD) native to MySQL. You can do this by writing your query as follows:

```
SELECT  DATE_FORMAT(date_column, ‘%m %d  %Y’)
FROM tablename;
```

The format code %m represents the month as a two-digit number; %d, the day as a two-digit number; and %Y, the year as a four-digit number. A summary of the more useful MySQL format codes for this purpose is shown in the table below

Code	Description
%M	Month, full text
%W	Weekday name, full text

Code	Description
%D	Day of month, numeric, with text suffix (for example, 1st)
%Y	Year, numeric, four digits
%y	Year, numeric, two digits
%a	Weekday name, three characters
%d	Day of month, numeric, leading zeros
%e	Day of month, numeric, no leading zeros
%m	Month, numeric, leading zeros
%c	Month, numeric, no leading zeros
%b	Month, text, three characters
%j	Day of year, numeric
%H	Hour, 24-hour clock, leading zeros
%k	Hour, 24-hour clock, no leading zeros
%h or %I	Hour, 12-hour clock, leading zeros
%l	Hour, 12-hour clock, no leading zeros
%i	Minutes, numeric, leading zeros
%r	Time, 12-hour (hh:mm:ss [AM|PM])
%T	Time, 24-hour (hh:mm:ss)
%S or %s	Seconds, numeric, leading zeros
%p	AM or PM
%w	Day of the week, numeric, from 0 (Sunday) to 6 (Saturday)

The UNIX_TIMESTAMP function works similarly but converts a column into a Unix timestamp. For example,

```
SELECT  UNIX_TIMESTAMP(date_column)
FROM tablename;
```

returns the date formatted as a Unix timestamp. You can then do as you want with it in PHP.

You can easily perform date calculations and comparisons with the Unix timestamp. Bear in mind, however, that a timestamp can usually represent dates only between 1902 and 2038, whereas the MySQL date type has a much wider range. As a rule of thumb, use a Unix timestamp for date calculations and the standard date format when you are just storing or showing dates.

4.2.6 Date Calculations in PHP

A simple way to work out the length of time between two dates in PHP is to use the difference between Unix timestamps. We used this approach in the script shown below

calc_age.php—Working Out a Person's Age Based on Birthdate

```
<?php
```

```
  // set  date for calculation
    $day =  18;
    $month  = 9;
     $year  = 1972;
     //  remember you need bday as day month and year
     $bdayunix  = mktime (0, 0, 0, $month, $day, $year); // get ts for
then
    $nowunix  = time(); // get unix ts for today
    $ageunix  = $nowunix - $bdayunix; // work out the difference
   $age =  floor($ageunix / (365 * 24 * 60 * 60)); // convert from seconds
   to years
  echo “Age is  $age”;
?>
```

This script sets the date for calculating the age. In a real application, it is likely that this information might come from an HTML form. The script begins by calling mktime() to work out the timestamp for the birthday and for the current time:

```
$bdayunix  = mktime (0, 0, 0, $month, $day, $year);
$nowunix  = mktime(); // get unix ts for today
```

Now that these dates are in the same format, you can simply subtract them:

```
 $ageunix  = $nowunix - $bdayunix;
```

Now, the slightly tricky part: converting this time period back to a more human-friendly unit of measure. This is not a timestamp but instead the age of the person measured in seconds. You can convert it back to years by dividing by the number of seconds in a year. You then round it down by using the floor() function because a person is not said to be, for example, 20, until the end of his twentieth year:

```
$age =  floor($ageunix / (365 * 24 * 60 * 60)); // convert from seconds to
years
```

Note, however, that this approach is somewhat flawed because it is limited by the range of Unix timestamps (generally 32-bit integers). Birthdates are not an ideal application for timestamps. This example works on all platforms only for people born from 1970 onward. Windows cannot manage timestamps prior to 1970. Even then, this calculation is not always accurate because it does not allow for leap years and might fail if midnight on the person’s birthday is the daylight savings switchover time in the local time zone.

4.2.7 Date Calculations in MySQL

PHP does not have many date manipulation functions built in. Obviously, you can write your own, but ensuring that you correctly account for leap years and daylight savings time can be tricky. Another option is to download other people's functions. You can find many as user-contributed notes in the PHP manual, but only some of them are wellthought out.

An option that may not seem immediately obvious is using MySQL. MySQL provides an extensive range of date manipulation functions that work for times outside the reliable range of UNIX timestamps. You need to connect to a MySQL server to run a MySQL query, but you do not have to use data from the database.
The following query adds one day to the date February 28, 1700, and returns the resulting date:

```
select  adddate('1700-02-28',  interval 1 day)
```

The year 1700 is not a leap year, so the result is 1700-03-01.

4.3 Connecting to the MySQL Database

To successfully use the PHP functions to talk to MySQL, you must have MySQL running at a location to which your web server can connect (not necessarily the same machine as your web server). You also must have created a user (with a password), and you must know the name of the database to which you want to connect.

Before you can access and work with data in a database, you must create a connection to the database.

In PHP, this is done with the mysql_connect() function.

Syntax

```
mysql_connect(servername,username,password);
```

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
 $con = mysql_connect("localhost","peter","pass");
  if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
?>
```

The MySQL extension was deprecated in 2012. Starting from PHP 7 the MySQL extension is no longer supported.

PHP 5 and later versions can work with a MySQL database using:

- MySQLi extension (the "i" stands for improved)
- PDO (PHP Data Objects)

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, where as MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

The rest of this manual will demonstrate both MySQLi extension (Object oriented and procedural) and PDO extension. The first thing is to make sure you have enabled those extensions in php.ini configuration file.

You use the following line in the script to connect to the MySQL server using MySQLi extension in object oriented fashion:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

This line instantiates the mysqli class and creates a connection to host ‘localhost’ with username ‘peter’, and password ‘pass’.

Using this object-oriented approach, you can now invoke methods on this object to access the database.

If you prefer a procedural approach, mysqli allows for this, too. To connect in a procedural fashion, you would use the following code:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

This function returns a resource rather than an object. This resource represents the connection to the database, and if you are using the procedural approach, you will need to pass this resource in to all the other mysqli functions. This is very similar to the way the file-handling functions, such as fopen(), work.

Most of the mysqli functions have an object-oriented interface and a procedural interface. Generally, the differences are that the procedural version function names start with mysqli_ and require you to pass in the resource handle you obtained from mysqli_connect(). Database connections are an exception to this rule because they can be made by the mysqli object’s constructor.

The result of your attempt at connection is worth checking because none of the rest of code will work without a valid database connection. You do this using the following code:

```
     if  (mysqli_connect_errno())
     {
     echo ‘Error:  Could not connect to database. Please try again later.’;
      exit;
    }
```

This code is the same for the object-oriented and procedural versions. But from PHP 5.2.9 and later you can use object oriented method $conn->connect_error as shown in the above examples.

The mysqli_connect_errno() returns an error number on error, or zero on success.

Note that when you connect to the database, if you begin the line of code with the error suppression operator @, you can handle any errors gracefully. (This could also be done with exceptions, which we have not used in this simple example.)

Bear in mind that there is a limit to the number of MySQL connections that can exist at the same time. The MySQL parameter max_connections determines what this limit is. The purpose of this parameter and the related Apache parameter MaxClients is to tell the server to reject new connection requests instead of allowing machine resources to be completely used up at busy times or when software has crashed.

You can alter both of these parameters from their default values by editing the configuration files. To set MaxClients in Apache, edit the httpd.conf file on your system.

To set max_connections for MySQL, edit the file my.conf.

Now, connection to the MySQL database using PDO will take the following fashion:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";

try {
   $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
// set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   echo "Connected successfully";
   }
catch(PDOException $e)
   {
   echo "Connection failed: " . $e->getMessage();
   }
?>
```

Notice that in the PDO example above we have also specified a database (myDB). PDO require a valid database to connect to. If no database is specified, an exception is thrown.

A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented)
$conn->close();

Example (MySQLi Procedural)
mysqli_close($conn);

Example (PDO)
$conn = null;

4.4 Working with Database

4.4.1 Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.
Syntax

CREATE DATABASE database_name

To get PHP to execute the statement above we must use the mysqli_query() function. This function is used to send a query or command to a MySQL connection.

In the following examples we create a database called "mydb":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
```

```
$username = "peter";
$password = "pass";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

The following PDO example create a database named "myDBPDO":

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";

try {
   $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $sql = "CREATE DATABASE myDBPDO";
   // use exec() because no results are returned
   $conn->exec($sql);
   echo "Database created successfully<br>";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }

$conn = null;
?>
```

Contrary to how mysql_query() and mysqli_query() work, there are two kinds of queries in PDO: ones that return a result (e.g. select and show), and ones that don't (e.g. insert, delete, and so on). The exec() method is used to execute query that does not return results while query() method is used to execute query that return results.

4.4.2 Choosing a Database to Use

Remember that when you are using MySQL from a command-line interface, you need to tell it which database you plan to use with a command such as

```
use  books;
```

You also need to do this when connecting from the Web. The database to use can directly be specified as a parameter to the mysqli constructor or the mysqli_connect() function as follow

```
 @$db  = new mysqli('localhost', 'peter', 'pass', 'books');
or
 @$db  = mysqli_connect('localhost', 'peter', 'pass', 'books');
```

If you want to change the default database, you can do so with the mysqli_select_db() function. It can be accessed as either

 $db->select_db(*dbname*)
 or as
 mysqli_select_db(*db_resource, db_name*)

Here, you can see the similarity between the functions that we described before: The procedural version begins with mysqli_ and requires the extra database handle parameter.

4.4.3 Create a Table

The CREATE TABLE statement is used to create a database table in MySQL.

Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
.......
)
```

We must add the CREATE TABLE statement to the mysqli_query() function to execute the command. Alternatively you can use query() method or exec() method

The following example shows how you can create a table named "MyGuests", with five columns. The column names will be "id", "firstname", "lastname", "email" and "reg_date":

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
```

```
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
sex VARCHAR(5),
email VARCHAR(50),
birth_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(30).

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
sex VARCHAR(5),
email VARCHAR(50),
birth_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   // sql to create table
   $sql = "CREATE TABLE MyGuests (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   firstname VARCHAR(30) NOT NULL,
   lastname VARCHAR(30) NOT NULL,
  sex VARCHAR(5),
 email VARCHAR(50),
 birth_date TIMESTAMP
   )";

   // use exec() because no results are returned
   $conn->exec($sql);
   echo "Table MyGuests created successfully";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }

$conn = null;
?>
```

4.4.4 Primary Keys and Auto Increment Fields

Each table should have a primary key field. A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The primary key field is always indexed. There is no exception to this rule! You must index the primary key field so the database engine can quickly locate rows based on the key's value.

The above example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

4.3.5 Retrieving Error Messages

Take some time to familiarize yourself with the mysqli_error() function it will become your friend. When used in conjunction with the PHP die() function, which simply exits the script at the point at which it appears, the mysqli_error() function will return a helpful error message when you make a mistake.

For example, now that you have created a table called MyGuests, you won't be able to execute that script again without an error. Try to execute the script again; when you execute the script, you should see something like the following in your web browser:

```
Could not create table: Table ' MyGuests' already exist
```

PDO has exception class to handle any problems that may occur when query is executed. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block. In the catch block the error message is stored in $e attribute and can be displayed using getMessage() method.

4.5 Working with MySQL Data

Inserting, updating, deleting, and retrieving data all revolve around the use of the mysqli_query() function or query() method to execute the basic SQL queries. In PDO we use query() method in retrieving while exec() method is used for Inserting, updating and deleting.

4.5.1 Insert Data into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

Syntax

```
INSERT INTO table_name
 VALUES (value1, value2,....)
```

You can also specify the columns where you want to insert the data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Note: SQL statements are not case sensitive. INSERT INTO is the same as insert into.

To get PHP to execute the statements above we must use the mysqli_query() function. This function is used to send a query or command to a MySQL connection in procedural way. Alternatively you can use $con->query() in object oriented way.

Example

In the previous section we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Gayo', 'john@ucc.com')";

if (mysqli_query($conn, $sql)) {
   echo mysqli_affected_rows($conn)." record(s) added";
} else {
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

mysqli_affected_rows() returns the number of rows affected by the last INSERT, UPDATE, or DELETE query associated with the provided link parameter. If the last query was invalid, this function will return -1. The object oriented style is $conn->affected_rows

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
```

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Gayo', 'john@ucc.com')";

if ($conn->query($sql) === TRUE) {
echo $conn->affected_rows." record(s) added";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email) VALUES ('John', 'Gayo',
'john@ucc.com')";
    // use exec() because no results are returned
   $affected_rows = $conn->exec($sql);
   echo $affected_rows." record(s) added";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }
$conn = null;
?>
```

4.5.2 Insert Data into a Database from a HTML Form

Now we will create an HTML form that can be used to add new records to the "MyGuests" table.

Here is the HTML form:

```
<html>
<body>
 <form action="insert.php" method="post">
   Firstname: <input type="text" name="firstname" />
     Lastname: <input type="text" name="lastname" />
   Email: <input type="text" name="email" />
  <input type="submit" />
</form>
```

```
</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php". The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables. Then, the mysqli_query() function executes the INSERT INTO statement, and a new record will be added to the database table.

Below is the code in the "insert.php" page:

```
<?php
  // Create short variable names
   $firstname=$_POST[‘firstname’];
   $lastname=$_POST[‘lastname’];
   $email=$_POST[‘email’];
   //Check if all data have been typed
    if  (!$firstname || !$lastname || !$email)
         {
        echo ‘You  have not entered all the required details.<br />’
         .’Please  go back and try again.’;
         exit;
         }
//remove space
                $firstname  = trim($firstname);
         $lastname  = trim($lastname);
         $email  = trim($email);

         if  (!get_magic_quotes_gpc())
         {
         $firstname  = addslashes($firstname);
         $lastname  = addslashes($lastname);
         $email  = addslashes($email);
        }
//make connection

    $con = mysqli_connect("localhost","peter","pass");
   if (!$con)
  {
  die('Could not connect: ' . mysqli_error());
  }
     mysqli_select_db($con, " MyGuests");
     $sql="INSERT INTO MyGuests(FirstName, LastName, Email)
     VALUES('$firstname','$lastname','$email')";
     if (!mysqli_query($con, $sql))
    {
  die('Error: ' . mysqli_error());
  }
  echo mysqli_affected_rows($con)." record(s) added";
 mysqli_close($con)
?>
```

Here, you check that all the form fields were filled in, and you format them correctly for insertion into the database (if required) with trim() and addslashes()

4.5.3 Insert Multiple Records into Database

Multiple SQL statements must be executed with the mysqli_multi_query() function.

The following examples add three new records to the "MyGuests" table:

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Gayo', 'john@ucc.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Momba', 'mary@ucc.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julieth', 'Donald', 'julieth@ucc.com')";

if (mysqli_multi_query($conn, $sql)) {
   echo mysqli_affected_rows($conn)." record(s) added";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note that each SQL statement must be separated by a semicolon

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Gayo', 'john@ucc.com');";
```

```
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Momba', 'mary@ucc.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julieth', 'Donald', 'julieth@ucc.com')";

if ($conn->multi_query($sql) === TRUE) {
   echo $conn->affected_rows." record(s) added";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Example (PDO)

The PDO way is a little bit different:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

 // begin the transaction
    $conn->beginTransaction();
 // our SQL statements
 $conn->exec("INSERT INTO MyGuests (firstname, lastname, email) VALUES ('John',
'Gayo', 'john@ucc.com')");
 $conn->exec("INSERT INTO MyGuests (firstname, lastname, email) VALUES ('Mary',
'Momba', 'mary@ucc.com')");
 $conn->exec("INSERT INTO MyGuests (firstname, lastname, email) VALUES ('Julieth',
'Donald', 'julieth@ucc.com')");
    // commit the transaction
    $conn->commit();
    echo "New records created successfully";
    }
catch(PDOException $e)
    {
    // roll back the transaction if something failed
    $conn->rollback();
    echo "Error: " . $e->getMessage();
    }
$conn = null;
?>
```

4.5.4 Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have two main advantages:

- Prepared statements reduces parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

The following example uses prepared statements and bound parameters in MySQLi:

Example (MySQLi with Prepared Statements)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?,
?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Gayo";
$email = "john@ucc.com";
```

```
$stmt->execute();

$firstname = "Mary";
$lastname = "Momba";
$email = "mary@ucc.com";
$stmt->execute();

$firstname = "Julieth";
$lastname = "Donald";
$email = "julieth@ucc.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

Code lines to explain from the example above:

```
"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"
```

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind_param() function:

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

Example (PDO with Prepared Statements)

The following example uses prepared statements and bound parameters in PDO:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
   // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // prepare sql and bind parameters
   $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
   VALUES (:firstname, :lastname, :email)");
   $stmt->bindParam(':firstname', $firstname);
   $stmt->bindParam(':lastname', $lastname);
   $stmt->bindParam(':email', $email);

   // insert a row
   $firstname = "John";
   $lastname = "Gayo";
   $email = "john@ucc.com";
   $stmt->execute();

   // insert another row
   $firstname = "Mary";
   $lastname = "Momba";
   $email = "mary@ucc.com";
   $stmt->execute();

   // insert another row
   $firstname = "Julieth";
   $lastname = "Donald";
   $email = "julieth@ucc.com";
   $stmt->execute();

   echo "New records created successfully";
   }
catch(PDOException $e)
   {
   echo "Error: " . $e->getMessage();
   }
$conn = null;
?>
```

In PDO, the SQL statement can use named placeholder (:name) or question mark (?) in parameter markers for which real values will be substituted when the statement is executed. You cannot use both named and question mark parameter markers within the same SQL statement

bindParam() method take three parameters. First parameter of the method is name of the parameters, second is the value we want to replace with and the last is the datatype using the PDO::PARAM_* constants. The list of constants which can be used are:

PDO::PARAM_BOOL - Represents a Boolean data type.

PDO::PARAM_NULL - Represents the SQL NULL data type.

PDO::PARAM_INT - Represents the SQL INTEGER data type.

PDO::PARAM_STR - Represents the SQL CHAR, VARCHAR, or other string data type.

When the third parameter is ommited, it is set to PDO::PARAM_STR by default. If we use question mark(?) instead of named placeholder, then the first parameter should indicate the position of the parameter marker starting from 1. See the example below:-

```
.
.
$query = $conn->prepare('INSERT INTO Users (username, name, age) VALUES (?, ?, ?)');
 $query->bindParam(1, 'admin');
  $query->bindParam(2, 'John');
 $query->bindParam(3, 17, PDO::PARAM_INT);
$query->execute();
```

4.5.5 Retrieving Data from the Database

The SELECT statement is used to retrieve data from a database.

Syntax

```
SELECT column_name(s) FROM table_name
```

or we can use the * character to select ALL columns from a table:

```
SELECT * FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

To get PHP to execute the statement above we must use the mysqli_query() function. This function is used to send a query or command to a MySQL connection in procedural approach. In object oriented approach use query() method.

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
```

```
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows>0) {
   // output data of each row
   while($row = $result->fetch_assoc()) {
      if  (!get_magic_quotes_gpc())
   {
       echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";

}
  echo  "id:  "  .  stripslashes($row["id"]).  "  -  Name:  "  .  stripslashes($row["firstname"]).  "  "  .
stripslashes($row["lastname"]).                                                                    "<br>";
   }

echo $result->num_rows.” row(s) retrieved”;
} else {
   echo "0 results";
}
$conn->close();
?>
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function num_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the method fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

The output of the code above will be:

```
id: 1 - Name: John Gayo
id: 2 - Name: Mary Momba
id: 3 - Name: Julieth Donald
```

The following example shows the same as the example above, in the MySQLi procedural way:

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
```

```
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result)>0) {
    // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
  if  (!get_magic_quotes_gpc())
    {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";

}
  echo "id: " . stripslashes($row["id"]). " - Name: " . stripslashes($row["firstname"]). " " .
stripslashes($row["lastname"]). "<br>";
    }

echo mysqli_num_rows($result).” row(s) retrieved”;
} else {
    echo "0 results";
}

mysqli_close($conn);
?>
```

The example above stores the data returned by the mysqli_query() function in the $result variable. Next, we use mysqli_num_rows() function to check number of rows returned, if it is greater than zero then mysqli_fetch_assoc() function is used to return the first row from the recordset as an array. Each subsequent call to mysqli_fetch_assoc() returns the next row in the recordset. Then we use the while loop to loop through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['id’], $row['firstname'] and $row['lastname']).

Several variations can be used to get results from a result identifier. Instead of an array with named keys, you can retrieve the results in an enumerated array with mysqli_fetch_row(), as follows:

```
 $row =  $result->fetch_row($result);
  or
 $row =  mysqli_fetch_row($result);
```

Here, the attribute values are listed in each of the array values $row[0], $row[1], and so on. (The mysqli_fetch_array() function allows you to fetch a row as either or both kinds of array.)

You could also fetch a row into an object with the mysqli_fetch_object() function:

```
$row =  $result->fetch_object();
 or
$row =  mysqli_fetch_object($result);
```

You can then access each of the attributes via $row->FirstName, $row->LastName, and so on.

You can free up your resultset by calling either

```
 $result->free();
   Or
  mysqli_free_result($result);
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$stmt =$conn->query("SELECT id, firstname, lastname FROM MyGuests");
$row_count = $stmt->rowCount();

 while($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
            }
        echo $row_count." row(s) retrieved";

   }
   catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
 }
$conn = null;
?>
```

query() method returns a PDOStatement object. You can also fetch results this way:

```
<?php
foreach($db->query('SELECT * FROM table') as $row) {
    echo $row['field1'].' '.$row['field2']; //etc...
}
```

or

```
<?php
$stmt = $db->query('SELECT * FROM table');
$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
//use $results
```

Note the use of PDO::FETCH_ASSOC in the fetch() and fetchAll() code above. This tells PDO to return the rows as an associative array with the field names as keys. Other fetch modes like PDO::FETCH_NUM returns the row as a numerical array. The default is to fetch with PDO::FETCH_BOTH which duplicates the data with both numerical and associative keys. It's recommended you specify one or the other so you don't have arrays that are double the size! PDO can also fetch objects with PDO::FETCH_OBJ.

Using the fetch() method will return ONE result from your query, unless you couple it with a while loop, as shown above. fetchAll() method is similar to fetch(), however it returns ALL the results.

4.5.6 Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<!DOCTYPE html>
<html>
<head>
<style>
table, th, td {
    border: 1px solid black;
}
</style>
</head>
<body>

<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows>0) {
    echo "<table><tr><th>ID</th><th>Name</th></tr>";
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["id"]. "</td><td>" . $row["firstname"]. " " .
$row["lastname"]. "</td></tr>";
    }
    echo "</table>";
} else {
    echo "0 results";
}
```

```
$conn->close();
?>

</body>
</html>
```

The output of the code above will be:

ID	Name
1	John Gayo
2	Mary Momba
3	Julieth Donald

4.4.7 Using WHERE clause

To select only data that matches a specific criteria, add a WHERE clause to the SELECT statement

Syntax

```
SELECT column FROM table
WHERE column operator value
```

The following operators can be used with the WHERE clause:

Operator	Description
=	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: SQL statements are not case sensitive. WHERE is the same as where.

To get PHP to execute the statement above we must use the mysqli_query() function or query() method.

Example

The following example will select all rows from the "MyGuests" table, where firstname='John':

```
<?php
```

```
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

  $result = mysqli_query($con, "SELECT firstname,lastname FROM MyGuests WHERE
firstname='John'");
  while($row = mysqli_fetch_assoc($result))
  {
  echo $row['firstname'] . " " . $row['lastname'];
  echo "<br />";
  }
  mysqli_free_result($result);
 mysqli_close($con);
?>
```

The output of the code above will be:

```
John Gayo
```

As it has narrated earlier, the usage of Prepared Statements is very important for web applications security as it eliminates the possibility of SQL injection

The following examples use prepared statements with where clause

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql="SELECT lastname, email FROM MyGuests WHERE id<? AND firstname =?";
$id = 5;
$firstname = 'John';

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  die("Wrong SQL: ".$sql. " Error: ".$conn->error);
}

/* Bind parameters types: s = string, i = integer, d = double, b = blob */
$stmt->bind_param('is',$id,$firstname);

/* Execute statement */
$stmt->execute();
//Iterate over results
$stmt->bind_result($lastname, $email);
while($stmt->fetch()) {
  echo $lastname . ", " . $email . "<br>";
}
//close statement and connection
$stmt->close();
$conn->close();
?>
```

Note: bind_result() Binds variables to a prepared statement for result storage. Instead of bind_result() you can use get_result() as follow $result = $stmt->get_result(); and fetch the results into an array using fetch_assoc() as normal.

Example (PDO)

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$id = 5;
$firstname = 'John';
//prepare statement
$stmt = $conn->prepare("SELECT lastname, email FROM MyGuests WHERE id<? AND
firstname =?");
// Bind parameters & types
$stmt->bindParam(1, $id, PDO::PARAM_INT);
$stmt->bindParam(2, $firstname, PDO::PARAM_STR);
// Execute statement
$stmt->execute();
//display
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
}
catch(PDOException $e) {
   echo "Error: " . $e->getMessage();
 }
$conn = null;
?>
```

4.5.8 Sorting Data

The ORDER BY keyword is used to sort the data in a recordset.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
```

Note: SQL statements are not case sensitive. ORDER BY is the same as order by.

Example

The following example selects all the data stored in the "MyGuests" table, and sorts the result by the "firstname" column:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$result = mysqli_query($conn, "SELECT * FROM MyGuests ORDER BY firstname");
 while($row = mysqli_fetch_Assoc($result))
 {
  echo $row['firstmame'];
  echo " " . $row['lastname'];
  echo " " . $row['Email'];
  echo "<br />";
  }
  mysqli_free_result($result);
 mysqli_close($conn);
?>
```

The output of the code above will be:

```
John Gayo john@ucc.com
Julieth Donald julieth@ucc.com
Mary Momba mary@ucc.com
```

Sort Ascending or Descending

If you use the ORDER BY keyword, the sort-order of the recordset is ascending by default (1 before 9 and "a" before "z").
Use the DESC keyword to specify a descending sort-order (9 before 1 and "z" before "a"):

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name DESC
```

Order by Two Columns

It is possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are identical:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name1, column_name2
```

4.5.9 Update Data

The UPDATE statement is used to modify data in a database table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Note: SQL statements are not case sensitive. UPDATE is the same as update.

To get PHP to execute the statement above we must use the mysqli_query() function or query() method.

Example

Earlier we created a table named "MyGuests" in a myDB database . Here is how it looks:

ID	FirstName	LastName	Email
1	John	Gayo	john@ucc.com
2	Mary	Momba	mary@ucc.com
3	Julieth	Donald	julieth@ucc.com

The following example uses prepared statement to update some data in the "MyGuests" table:

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";
```

```
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
$sql="UPDATE MyGuests SET firstname = ?, Email = ? WHERE id = ?";
$firstname = 'Peter';
$email = 'peter@ucc.com'
$id = 1;

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  die("Wrong SQL: ".$sql. " Error: ".$conn->error);
}

/* Bind parameters and types: s = string, i = integer */
$stmt->bind_param('ssi',$firstname, $email, $id);

/* Execute statement */
$stmt->execute();

echo $stmt->affected_rows." rows affected";

$stmt->close();
$conn->close();
?>
```

After the update, the "MyGuests" table will look like this:

ID	FirstName	LastName	Email
1	Peter	Gayo	peter@ucc.com
2	Mary	Momba	mary@ucc.com
3	Julieth	Donald	julieth@ucc.com

4.5.10 Delete Data in a Database

The DELETE FROM statement is used to delete records from a database table.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

Note: SQL statements are not case sensitive. DELETE FROM is the same as delete from.

To get PHP to execute the statement above we must use the mysqli_query() function query() method. This function is used to send a query or command to a MySQL connection.

Example

Earlier we created a table named "MyGuests" in a myDB database . Here is how it looks:

ID	FirstName	LastName	Email
1	John	Gayo	john@ucc.com
2	Mary	Momba	mary@ucc.com
3	Julieth	Donald	julieth@ucc.com

The following example uses prepared statement to delete all the records in the "MyGuests" table where firstname ='Julieth':

```
<?php
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$sql="DELETE FROM MyGuests WHERE firstname = ?";
$first = 'Julieth';

/* Prepare statement */
$stmt = $conn->prepare($sql);
if($stmt === false) {
  die("Wrong SQL: ".$sql. " Error: ".$conn->error);
}

/* Bind parameters and types: s = string */
$stmt->bind_param('s',$first);

/* Execute statement */
$stmt->execute();

echo $stmt->affected_rows. " rows affected" ;

$stmt->close();
$conn->close();
```

After the deletion, the table will look like this:

ID	FirstName	LastName	Email
1	John	Gayo	john@ucc.com
2	Mary	Momba	mary@ucc.com

Exercises

1. What is the primary function used to make the connection between PHP and MySQL, and what information is necessary?
2. Using PHP, how do you acquire a UNIX time stamp that represents the current date and time?
3. Which PHP function accepts a time stamp and returns an associative array that represents the given date?
4. Which PHP function do you use to format date information?
5. Which PHP function could you use to check the validity of a date?
6. Why MySQL extension is not recommended to be used in a new project?
7. Provide the differences between MySQLi extension and PDO
8. Why prepared statements are recommended in Web Applications
9. Which PHP function retrieves the text of a MySQL error message?
10. How the error messages are handled in PDO
11. Which PHP function is used to count the number of records in a resultset?

Chapter 5: Session Management

You might have heard people say, “HTTP is a stateless protocol.” This means that the protocol has no built-in way of maintaining state between two transactions. When a user requests one page, followed by another, HTTP does not provide a way for you to tell that both requests came from the same user.

The idea of session management is to be able to track a user during a single session on a website. If you can do this, you can easily support logging in a user and showing content according to her authorization level or personal preferences. You can track the user’s behavior, and you can implement shopping carts.

5.1 Introduction to Sessions

A session is a period of time a user interfaces with a given system. The user session begins when the user accesses the system and ends when the user quits from the system.

A web session is a sequence of network HTTP request and response transactions associated to the same user. Modern and complex web applications require the retaining of information or status about each user for the duration of multiple requests.

There are four ways in which sessions can be managed in a PHP application:-

1. URL rewriting
2. Hidden form elements
3. Cookies
4. Sessions

5.2 URL rewriting

When you submit a form using the GET method, its fields and values are URL encoded and appended to the URL to which the form is sent. They then become available to the server and to your scripts. Assuming a form with two fields, name and age, the query string should end up looking something like the following:

http://www.ucc.co.tz/appdipform.php?name=saguge&age=24

Each name and value is separated by an equals (=) sign, and each name/value pair is separated by an ampersand (&). PHP decodes this string and makes each of the pairs available in the $_GET associative array variable. It also creates a global variable for each name, populating with the corresponding value. So, to access the name GET variable, you could use:
$_GET[‘name’];
You are not limited, however, to using forms to send query strings. You can build your own relatively easily and in so doing pass substantial amounts of information from page to page.
A Query string is information that is appended to the end of a page URL. The query string is composed of a series of field-value pairs. Within each pair, the field name and value are separated by an equals sign, '=' and the series of pairs is separated by the ampersand, '&' or semicolon, ';' for URLs embedded in HTML and not generated by a form.
Therefore this method could also be called session management by using query string.

5.2.1 Creating a query string

To create a query string, you need to be able to URL encode the keys and values you want to include. Assume that we want to pass a URL to another page as part of a query string. The forward slashes and the colon in a full URL would create ambiguity for a parser. We must therefore convert the URL into hexadecimal characters. We can do this using PHP's urlencode() function. urlencode() accepts a string and returns an encoded copy:

```
print urlencode("http://80-www.corrosive.co.uk.proxy.lib.uiowa.edu");
// prints http%3A%2F%2Fwww.corrosive.co.uk
```

Now that you can URL encode text, you can build your own query string. The following fragment builds a query string from two variables:

```
<?php
$name="ng'itu";
$email="sagida@uccmail.co.tz";
$query="name=".urlencode($name);
$query.="&email=".urlencode($email);
?>
<A HREF= "t3.php?<?php echo $query; ?>" >Go</A>
```

The URL in the link will reach the browser including an encoded query string: t3.php?name=king%27ote&email=jumbo%40uccmail.co.tz . Name and email parameters will become available within t3.php as global variables.

This approach is clumsy, however. Because we have hard-coded variable names into the query string, we cannot reuse the code easily. To pass information effectively from page to page, we need to make it easy to embed names and values into a link and generate a query string automatically. This is especially important if we are to maintain the benefit of PHP that it is easy for a non-programmer to work around.

5.3 Hidden Form Element

Hidden input fields are form fields that are not visible. The user can't see or change these fields, and they are used to transmit state information between different pages. In this case user information is stored in hidden field value and retrieved from another PHP page.

Let's use hidden fields to transport our data across our form, to the final processing script.

We start with the form for step 1:

```
<form  method="post" action="form2.php">
   Name:<input type="text" name="name"><br>
   Email:<input type="text" name="email_address"><br>
   <input type="submit" value="Go To Step 2">
</form>
```

Ok, so nothing more than 2 input fields and a submit button to take us to step 2. In the following page, apart from the HTML form to gather membership data, we are going to need code to store the submitted data from step 1 in the session.

```
<form method="post" action="form3.php">
Gender<input type="radio" name=”gender” value="Male">Male

<input type="radio" name=”gender” value="Female">Female<br>

<input type="hidden" name="name"
value="<?php echo $_POST['name']; ?>">
```

```
<input type="hidden" name="email_address"
 value="<?php echo $_POST['email_address']; ?>">

<input type="submit" value="Go To Step 3">
</form>
```

Please note that although the hidden fields are not visible to the visitor, they are visible in source of the page, so you must not use them to store critical information.

The data in the fields can be processed as follows

```
<?php

//accessing data
$name = $_POST['name'];
$email = $_POST[‘email_address’];
$name = $_POST['gender’];
// then you can write code that make use of this data.
echo :Your name is $name”;

?>
```

5.4 Cookies

5.4.1 Introduction to Cookies

A cookie is a small piece of information that scripts can store on a client-side machine and they are kept of use tracking purpose.

When a browser connects to an URL, it first searches the cookies stored locally. If any of them are relevant to the URL being connected to, they will be transmitted back to the server.

5.4.2 Setting Cookies with PHP

You can set a cookie on a user’s machine by sending an HTTP header containing data in the following format:

```
Set-Cookie: NAME=VALUE; [expires=DATE;] [path=PATH;]
[domain=DOMAIN_NAME;] [secure]
```

This creates a cookie called *NAME* with the value *VALUE*. The other parameters are all optional. The *expires* field sets a date beyond which the cookie is no longer relevant. (Note that if no expiry date is set, the cookie is effectively permanent unless you or the user manually delete it). Together, the path and domain can be used to specify the URL or URLs for which the cookie is relevant. The *secure* keyword means that the cookie will not be sent over a plain HTTP connection.

You can manually set cookies in PHP using the setcookie() function. It has the following prototype:

```
bool setcookie(string name [,string value [,int expire [,string path [,string domain [,int secure]]]]);
```

The parameters correspond exactly to the ones in the Set-Cookie header mentioned previously. Here is the detail of all the arguments:

1. **Name -** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
2. **Value -**This sets the value of the named variable and is the content that you actually want to store.
3. **Expiry -** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
4. **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
5. **Domain -** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
6. **Security -** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

If you set a cookie as

```
<?php
$username = 'peter';
setcookie('username', $username);
?>
```

As this cookie does not have an expiry time, then the cookie will be deleted once the user closes their browser. This can be very useful for keeping a user logged in for an indefinite amount of time, however, as soon as they close their browser (hence, have left the site), the cookie is removed.

There are two **very** important things to abide by when using cookies. Firstly, there can be no HTML, text or white-space output before calling the **setcookie()** function. This is due to a 'protocol restriction' and you will find that **header()** and **session_start()** functions must also follow this rule.

5.4.3 Accessing the Cookie

To access any cookies set in the current domain you will have to use either $_COOKIE[‘cookieName’] or $HTTP_COOKIE_VARS[“cookieName’]. Or, if you have register_globals turned on, you will have access directly as $cookieName.

You can use **isset()** function to check if a cookie for previous example is set or not.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
 if(isset($_COOKIE["username"]))
   echo "Welcome " . $_COOKIE["username"] . "<br />";
   else
   echo "Sorry... Not recognized" . "<br />";
   ?>
</body>
</html>
```

5.4.4 Deleting Cookies

Cookies as very picky about how they are deleted and usually require that they be removed, using the same values as they were set with. Hence, if we have a domain or path specified, then we must specify this domain/path when deleting the cookie.

To delete a cookie, you simply set the value of the cookie to null, and also set the expiring time of the cookie in the past.

Ideally, one would set the expiry time to about a year in the past, so that if the system time is off by a few minutes or days even, then the cookie will still delete correctly.

```
<?php
setcookie('username', '', time()-60*60*24*365);
?>
```

5.4.5 Creating a Simple Cookies Example

To give a more detailed look into how to use cookies, I am going to show you how to create a little login form so that you can store the username and password in a cookie. I will start with form login.html which allow user to supply the credential for setting cookies and login

```
<html>
<head>
<title>User Logon</title> </head>
<body>
<h2>User Login </h2>
<form name="login" method="post" action="login.php">
Username: <input type="text" name="username"><br>
Password: <input type="password" name="password"><br>
Remember Me: <input type="checkbox" name="rememberme" value="1"><br>
<input type="submit" name="submit" value="Login!"> </form>
</body>
</html>
```

The output of this form will be as in figure 5.1

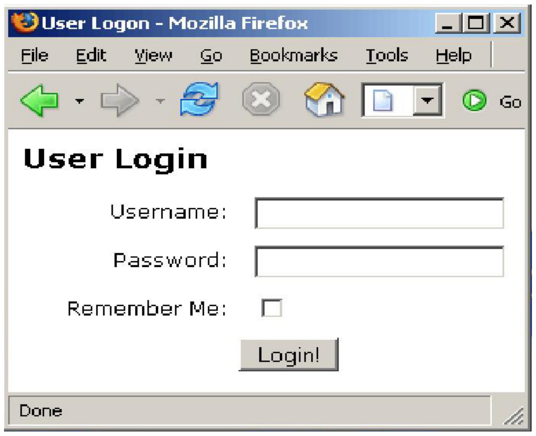

User Logon - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
User Login
Username:
Password:
Remember Me:
Login!
Done

figure 5.1 User Login form

Now that we have our form, we will create our login script. We must decide what restrictions we are going to place on the cookie. The cookies will be set when user select Remember Me checkbox, otherwise the user will be directed to index.php page

I have decided that this will only run on the www.ucc.com domain and in the /account directory only. Hence,

```
<?php
/* These are our valid username and passwords */
$user = 'peter';
$pass = 'pass';
if (isset($_POST['username']) && isset($_POST['password'])) {
if (($_POST['username'] == $user) && ($_POST['password'] == $pass)) {
if (isset($_POST['rememberme'])) {
 /* Set cookie to last 1 year */
setcookie('username', $_POST['username'], time()+60*60*24*365, '/account', '
www.ucc.com');
setcookie('password', md5($_POST['password']), time()+60*60*24*365, '/account',
'www.ucc.com');
} else {
/* Cookie expires when browser closes */
setcookie('username', $_POST['username'], false, '/account', 'www.ucc.com');
setcookie('password', md5($_POST['password']), false, '/account', 'www.exampl e.com');
 }
header('Location: index.php');
} else {
echo 'Username/Password Invalid';
}
} else {
echo 'You must supply a username and password.';
}
?>
```

The index.php will access the cookies set by Login.php. In this script, we just check that the cookie exists and is valid. If they aren't, then the user is redirected back to the login form.

Otherwise a welcome message is included. The only important thing to notice is how we have validated the password. Before on the login.php script, we have encrypted our password using md5() and as this encryption cannot be undone, we must compare encrypted versions. Hence, we encrypt our preset value and compare it to the already hashed cookie value. This way, there is no chance of the original password becoming available.

```
<?php
/* These are our valid username and passwords */
$user = 'peter';
$pass = 'pass';
if (isset($_COOKIE[['username']) && isset($_COOKIE['password')) {
if (($_POST['username'] != $user) || ($_POST['password'] != md5($pass))) {
header('Location: login.html');
} else {
echo 'Welcome back ' . $_COOKIE['username']. '<br>';
echo '<a href= logout.php>log out</a>';
}
} else {
header('Location: login.html');
}
?>
```

Now you can code the logout.php file to delete the cookies as follow

```
<?php

setcookie('username', '', time()-60*60*24*365, '/account', 'www.ucc.com');
setcookie('password', '', time()-60*60*24*365, '/account', 'www.ucc.com');

header('Location: login.html');

?>
```

5.5 Session

5.5.1 Introduction to Session

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer.

Sessions in PHP are driven by a unique session ID, a cryptographically random number. This session ID is generated by PHP and stored on the client side for the lifetime of a session. It can be either stored on a user's computer in a cookie or passed along through URLs.

The session ID acts as a key that allows you to register particular variables as so-called session variables. The contents of these variables are stored at the server. The session ID is the only information visible at the client side. If, at the time of a particular connection to your site, the session ID is visible either through a cookie or the URL, you can access the session variables stored on the server for that session. By default, the session variables are stored in flat files on the server.

Cookies have some associated problems: Some browsers do not accept cookies, and some users might have disabled cookies in their browsers. This is one of the reasons PHP sessions use a dual cookie/URL method.

When you are using PHP sessions, you do not have to manually set cookies. The session functions take care of this task for you.

You can use the function session_get_cookie_params() to see the contents of the cookie set by session control. It returns an array containing the elements lifetime, path, domain, and secure.

5.5.2 Implementing Simple Sessions

The basic steps of using sessions are

1. Starting a session
2. Registering session variables
3. Using session variables
4. Deregistering variables and destroying the session

Note that these steps don't necessarily all happen in the same script, and some of them happen in multiple scripts. Let's examine each of these steps in turn.

5.5.2.1 Starting a Session

Before you can use session functionality, you need to actually begin a session. There are two ways you can do this. The first, and simplest, is to begin a script with a call to the session_start()function:

```
session_start();
```

This function checks to see whether there is already a current session. If not, it will essentially create one, providing access to the superglobal $_SESSION array. If a session already exists, session_start() loads the registered session variables so that you can use them. It's a good idea to call session_start() at the start of all your scripts that use sessions.

The second way you can begin a session is to set PHP to start one automatically when someone comes to your site. You can do this by using the *session.auto_start* option in your php.ini file; this method has one big disadvantage: With *auto_start* enabled, you cannot use objects as session variables.

5.5.2.2 Registering Session Variables

The way you register session variables has recently changed in PHP. Session variables have been stored in the superglobal array $_SESSION since PHP 4.1, and also in the older $HTTP_SESSION_VARS. We recommend you use $_SESSION. To create a session variable, you simply set an element in this array, as follows:

```
$_SESSION['myvar'] = 5;
```

If you are using an older version of PHP, for a variable to be tracked from one script to another, you would have registered it with a call to session_register(). This use is now deprecated.

The session variable you have just created will be tracked until the session ends or until you manually unset it.

5.5.2.3 Using Session Variables

To bring session variables into scope so that they can be used, you must first start a session using *session_start()*.You can then access the variable via the $_SESSION superglobal array—for example, as $_SESSION['myvar'].

When you are using an object as a session variable, it is important that you include the class definition before calling session_start() to reload the session variables. This way, PHP knows how to reconstruct the session object.

If you have *register_globals* turned on, you can access session variables via their short form names—for example, $myvar—but this approach is not recommended. If you do have register_globals on, bear in mind that a session variable cannot be overridden by GET or POST data, which is a good security feature, but something to bear in mind when you're coding.

On the other hand, you need to be careful when checking whether session variables have been set (via, say, isset() or empty()). Remember that variables can be set by the user via GET or POST. You can check a variable to see whether it is a registered session variable by checking in $_SESSION. You can check this directly using the following

```
if (isset($_SESSION['myvar'])
```

5.5.2.4 Unsetting Variables and Destroying the Session

When you are finished with a session variable, you can unset it. You can do this directly by unsetting the appropriate element of the $_SESSION array, as in this example:

```
unset($_SESSION['myvar']);
```

Note that the use of session_unregister() and session_unset() is no longer required and is not recommended. These functions were used prior to the introduction of $_SESSION. You should not try to unset the whole $_SESSION array because doing so will effectively disable sessions. To unset all the session variables at once, use

```
$_SESSION = array();
```

When you are finished with a session, you should first unset all the variables and then call

session_destroy();

to clean up the session ID

5.5.2.3 Creating a Simple Session Example

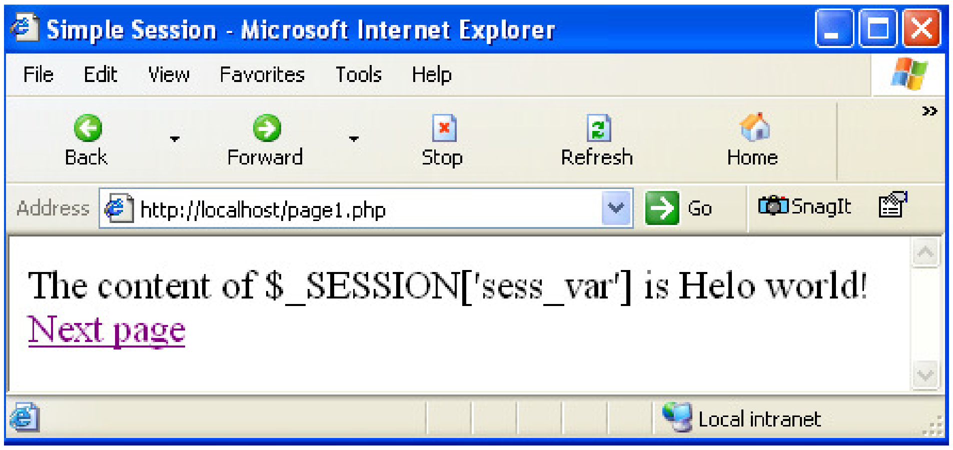
Some of this discussion might seem abstract, so let's look at an example. Here, you'll implement a set of three pages.

On the first page, start a session and create the variable $_SESSION['sess_var']. The code to do this is shown below

page1.php—Starting a Session and Creating a Session Variable

```
<?php
 session_start();
  echo "
  <html>
   <head> <title>Simple Session</title></head>
   <body>
   <font size = 4> ";
  $_SESSION['sess_var'] = 'Helo world!';
 echo 'The content of $_SESSION[\'sess_var\'] is '.$_SESSION['sess_var'].'<br />';
 echo "
  <a href=\"page2.php\">Next page</a>
   </font>
   </body>
  </html> ";
 ?>
```

This script creates the variable and sets its value. The output of this script is shown in figure 5.2


Simple Session - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://localhost/page1.php Go SnagIt
Local intranet

```
The content of $_SESSION['sess_var'] is Helo world!
Next page
```

Figure 5.2 simple session

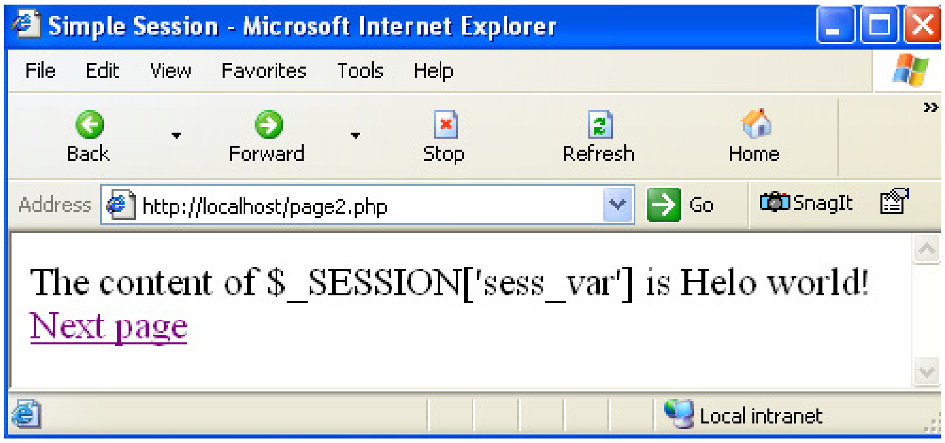
The final value of the variable on the page is the one that will be available on subsequent pages. At the end of the script, the session variable is serialized, or frozen, until it is reloaded via the next call to session_start().

You can therefore begin the next script by calling session_start(). This script is shown in below

page2.php—Accessing a Session Variable and Unsetting It

```
<?php
 session_start();
 Echo "
  <html>
   <head> <title>Simple Session</title></head>
   <body>
  <font size = 4>
 ";
 echo 'The content of $_SESSION[\'sess_var\'] is '.$_SESSION['sess_var'].'<br />';
  unset($_SESSION['sess_var']);
   Echo "
   <a href=\"page3.php\">Next page</a>
    </font>
    </body>
   </html>
  ";
 ?>
```

After you call session_start(), the variable $_SESSION ['sess_var'] is available with its previously stored value, as you can see in Figure 5.3.


Simple Session - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://localhost/page2.php Go SnagIt
Local intranet

The content of $_SESSION['sess_var'] is Helo world!
Next page

Figure 5.3 simple session 2

The value of the session variable is passed along via the session ID to page2.php.

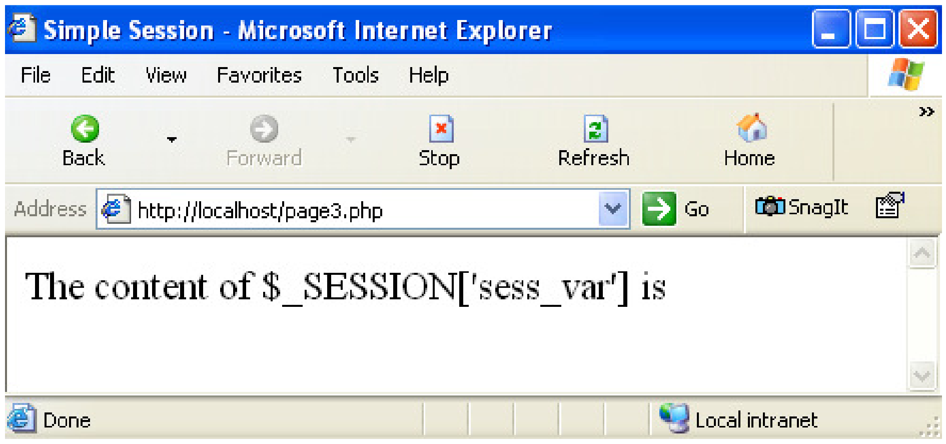
After you have used the variable, you unset it. The session still exists, but the variable $_SESSIONx['sess_var'] no longer exists.

Finally, you pass along to page3.php, the final script in the example. The code for this script is shown below

page3.php—Ending the Session

```
<?php
 session_start();
  Echo "
   <html>
    <head> <title>Simple Session</title></head>
   <body>
  <font size = 4>
 ";
 echo 'The content of $_SESSION[\'sess_var\'] is '.$_SESSION['sess_var'].'<br />';
  session_destroy();
   Echo "
    </font>
    </body>
   </html>
  ";
?>
```

As you can see in the figure 5.4, you no longer have access to the persistent value of $_SESSION['sess_var'].


Simple Session - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://localhost/page3.php Go SnagIt
Done Local intranet

```
The content of $_SESSION['sess_var'] is
```

Figure 5.4 simple session 3

With some PHP versions prior to 4.3, you might encounter a bug when trying to unset elements of $HTTP_SESSION_VARS or $_SESSION. If you find that you are unable to unset elements (that is, they stay set), you can revert to using session_unregister() to clear these variables. You finish by calling session_destroy() to dispose of the session ID.

5.6 Using HTTP Headers

HTTP Headers can provide dynamic content according to browser type, randomly generated numbers or User Input. It can also redirect the client browser to another location.

5.6.1 Identifying client Browser, Platform, computer and IP address

PHP creates some useful environment variables that can be seen in the phpinfo.php page that was used to setup the PHP environment.

One of the environment variables set by PHP is HTTP_USER_AGENT which identifies the user's browser and operating system.

PHP provides a function getenv() to access the value of all the environment variables. The information contained in the HTTP_USER_AGENT environment variable can be used to create dynamic content appropriate to the browser.

Following example demonstrates how you can identify a client browser and operating system.

```
<html>
<body>
<?php
$viewer=getenv("HTTP_USER_AGENT");
$browser = "An unidentified browser";
if(preg_match("/MSIE/i", "$viewer"))
{
$browser = "Internet Explorer";
}
else if(preg_match("/Netscape/i", "$viewer"))
{
$browser = "Netscape";
}
else if(preg_match("/Mozilla/i", "$viewer"))
{
$browser = "Mozilla";
}
$platform = "An unidentified OS!";
if( preg_match("/Windows/i","$viewer"))
{
$platform = "Windows!";
}
else if (preg_match("/Linux/i", "$viewer"))
{
$platform = "Linux!";
}
echo("You are using $browser on $platform");
?>
</body>
</html>
```

The result for these scripts will vary depending on what your computer is using. if your computer is using Mozilla as browser and Windows as operating system, then the results will be:

```
You are using Mozilla! on Windows!
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.
Table 5.1 shows PHP superglobal variables that can be used to retrieve various client information.

Table 5.1: superglobal variables

variable	Uses
$_SERVER['REMOTE_ADDR']	It can be used to retrieve the IP address from where the use is viewing the current page
$_SERVER['REMOTE_HOST']	It will return the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	It will return the port being used on the user's machine to communicate with the

	web server

5.6.2 Browser Redirection

The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

Following example demonstrates how you can redirect a browser request to another web page. Try out this example by putting the source code in test.php script.

```
<?php
 if( $_POST["location"])
 {
   $location = $_POST["location"];
   header("Location:$location");
   exit();
 }
?>
<html>
<body>
  <p>Choose a site to visit :</p>
  <form action="<?php $_PHP_SELF ?>" method="POST">
  <select name="location">
    <option value="http://w3c.org"> World Wise Web Consortium
    </option>
    <option value="http://www.google.com"> Google Search Page
    </option>
  </select>
  <input type="submit" />
  </form>
</body>
</html>
```

5.6.3 Retaining form data value after posting

Usualy, after the user fills the form and submits it, the page is reloaded and the form is empty. In case of an error or form validation fails, the user need to write again all those data to submit the form.

It is user-friendly if we keep the data that the user has added in form fields before submitting it, so that he can correct or write only data in the fields with errors. The submitted data fields can be retained either by using Query String(URL rewriting) or Session. PHP header redirected method is then used to post back those data to the submitted form.

Session control retain data by assigning them to the registered session variables. Data can then be accessed in directed page by using $_SESSION method. Therefore, session provides a safe way of transferring retained data because data will not be visible to the client.

We will first demonstrate how to retain the data for various form elements using query string and thereafter we will give practical example using Session.

Remember that data is treated differently in form elements. For text box and textarea elements user input is entered by typing. Therefore, to take care of some characters like & we have to use urlencode and urldecode to first encode the data before sending through the address bar and then after receiving at main page again we will decode it to get the original data entered by the visitor.

Period or radio buttons are used where the visitor has to make selection out of some given choice. The user can only make one choice out of the given options. So radio buttons are grouped in a common choice and one of them is selected by the user. The difference between radio button and check box is user can make multiple selections in case of check boxes.

From a drop down list box or combo box visitor selects one option and submits the form.If the form validation fails then we have to show again the same form asking for changes in input data and submit again. In this process we have to keep our drop down box selected to the option previously selected by the user.

Here is the code of the form which will post the data.

```
<?php
//get data from url/query string
$t1v=$_GET['t1v'];
$r1=$_GET['r1'];
$ch1=$_GET['ch1'];
$ch2=$_GET['ch2'];
$s1=$_GET['s1'];

//insert the data to the form elements
$t1v=urldecode($t1v); //text box
switch($r1)
{
case "Male":
$r1v="checked";
$r2v="";
break;

case "Female":
$r1v="";
$r2v="checked";
break;

default: // By default the 1st selection is selected
$r1v="checked ";
$r2v="";
break;
} // radio button

if($ch1=="yes"){$ch1v="checked";}
else{$ch1v=="";}
if($ch2=="yes"){$ch2v="checked";}
else{$ch2v=="";}  // checkboxes

switch ($s1)
{
case "rice":
$rice="selected";
break;
case "pizza":
$piz="selected";
break;
} //drop down box

//load form
echo "
<form method=post action=process.php>
Your Name: <input type=text name=t1 value='$t1v'><br>
Sex:
<input type=radio name=r1 value='Male' $r1v>Male |
<input type=radio name=r1 value='Female' $r2v>Female |<br>
Hobbies:
<input type=checkbox name=ch1 value=yes $ch1v>Sporting
<input type=checkbox name=ch2 value=yes $ch2v>Surfing<br>
Delicious Food:
<select name=s1>
<option value=rice $rice>Rice</option>
<option value=pizza $piz>Pizza</option>
</select>
<input type=submit value=Submit>
";
?>
```

Now we will go to the second page were the form date is collected and posted back to main page. Here is the code of process.php

```
<?php
//receive data from a form
 $t1=$_POST['t1'];
  $r1=$_POST['r1'];
 $ch1=$_POST['ch1'];
 $ch2=$_POST['ch2'];
$s1=$_POST['s1'];
// encode text box dat
$t1=urlencode($t1);
//return data to the main form
header("Location:p_form.php?t1v=$t1&r1=$r1&ch1=$ch1&ch2=$ch2&s1=$s1");
?>
```

The output form will look like figure 5.5

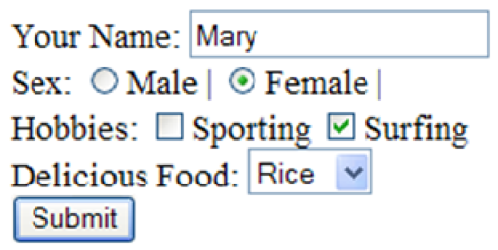


Figure 5.5 A simple Form

Try to build these Scripts and try to test by entering some values in a form.

Now, we are going to build a real example using session control. Earlier in chapter 3 we created a table named "MyGuests" in a myDB database with the following columns ID, firstname, lastname, sex, email and birth_date. We are going to create a form will be used to enter data into MyGuests table. When validation fails, the form will retain all filled data together with error message. If the data is successfully entered into table, the empty form is returned together with successfully message. All the user inputs are going to be validated before connecting to myDB database.

Here is a form page fdata.php:

```
<?php
 session_start();
//get data from session variables
$fname = $_SESSION['fnamev'];
$lname = $_SESSION['lnamev'];
$sex = $_SESSION['sexv'];
$email = $_SESSION['emailv'];
// load form
?>
<html>
   <head> <title>Input Data Form </title></head>
  <body>
<form method=post action=fprocess.php>
<table><tr>
<td>First Name:</td><td> <input type="text" name="fname" value="<?php echo
$fname;?>"> </td></tr><tr>
<td>Last Name: </td><td> <input type="text" name="lname" value="<?php echo
$lname;?>"> </td></tr><tr>
<td>Sex:</td><td>
<input type="radio" name="sex" value="female"
<?php if(isset($sex) && $sex=="female") echo "checked";?>>Female
<input type="radio" name="sex" value="male"
<?php if(isset($sex) && $sex=="male") echo "checked";?>>Male </td></tr><tr>

<td>E-mail:</td><td><input type="text" name="email" size = 20 value="<?php echo
$email;?>"> </td></tr><tr>
```

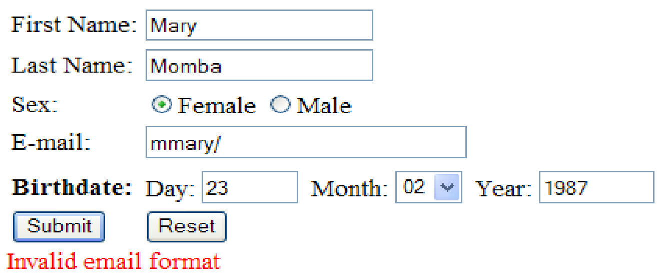
```
<tr><td><b>Birthdate: </td><td>
Day: <input type="text" name='Day' size = '6' value = '<?php echo $_SESSION['dayv'];
?>'>&nbsp;
Month: <select name="Month">
<option value="<?php echo $_SESSION['monthv']; ?>"><?php echo $_SESSION['monthv'];
?></option>
<option value="01">01</option>
<option value="02">02</option>
<option value="03">03</option>
<option value="04">04</option>
<option value="05">05</option>
<option value="06">06</option>
<option value="07">07</option>
<option value="08">08</option>
<option value="09">09</option>
<option value="10">10</option>
<option value="11">11</option>
<option value="12">12</select>&nbsp;
Year: <input type="text" name='Year' size = '8' value = '<?php echo $_SESSION['yearv'];
?>'></td></tr>
<tr><td><input type="submit" value ='Submit'></td><td>
<input type="reset" value ='Reset'></td></tr></table>
<?php echo $_SESSION['msg'];
unset($_SESSION['msg']);
?>
</form></body></html>
```

Here is form processing file fprocess.php

```
<?php
session_start();
//acquire data from the form
$fname = $_POST['fname'];
$lname = $_POST['lname'];
$sex = $_POST['sex'];
$email = $_POST['email'];
$day = $_POST['Day'];
$month = $_POST['Month'];
$year = $_POST['Year'];
//register session variables and assign posted data to them
$_SESSION['fnamev'] =$fname;
$_SESSION['lnamev'] =$lname;
$_SESSION['emailv'] =$email;
$_SESSION['sexv'] =$sex;
$_SESSION['dayv'] =$day;
$_SESSION['monthv'] =$month;
$_SESSION['yearv'] =$year;
//Check whether all inputs have been provided
if(!$fname||!$lname||!$email||!$sex||!$day||!$month||!$year) {
$_SESSION['msg'] ="<font color = red> Please Fill All Inputs </font>";
header("Location: fdata.php");
exit;
}
```

```
//Validate email format
if(!filter_var($email, FILTER_VALIDATE_EMAIL)) {
$_SESSION['msg'] ="<font color = red>Invalid email format</font>";
header("Location: fdata.php");
exit;
}

//Check whether day and year are numeric
if(!is_numeric($day)||!is_numeric($year)) {
$_SESSION['msg'] ="<font color = red> Day and Year Should be in numerical form</font>";
header("Location: fdata.php");
exit;
}
//Check whether the value for year has been entered in four digits
if(strlen($year)<>4) {
$_SESSION['msg'] ="<font color = red> Year Should be entered in four digits form</font>";
header("Location: fdata.php");
exit;
}
//Check whether the date is valid
if(!checkdate($month,$day,$year)) {
$_SESSION['msg'] ="<font color = red>Date is not valid!!, Please correct</font>";
header("Location: fdata.php");
exit;
}
//prepare birth date in Time stamp format
$bdate = date("Y-m-d", mktime(0, 0, 0, $month,$day,$year));
//make connections
$servername = "localhost";
$username = "peter";
$password = "pass";
$dbname = "myDB";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
 $_SESSION['msg'] ="<font color = red>$conn->connect_error</font>";
 header("Location: fdata.php");
exit;
}

// prepare statement
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname,lastname,sex,
email,birth_date) VALUES (?,?,?,?,?)");

if(!$stmt) {
$_SESSION['msg'] ="<font color = red>$conn->error</font>";
 header("Location: fdata.php");
exit;
} else {
// bind parameters
$stmt->bind_param("sssss",$fname,$lname,$sex,$email,$bdate);
//execute query
$stmt->execute();
//close prepared statement
$stmt->close();
$_SESSION['msg'] ="<font color = blue>record successfully inserted</font>";
//destroy session variables & return back to the form
unset($_SESSION['fnamev']);
unset($_SESSION['lnamev']);
unset($_SESSION['sexv']);
unset($_SESSION['emailv']);
unset($_SESSION['dayv']);
unset($_SESSION['monthv']);
unset($_SESSION['yearv']);
header("Location: fdata.php");
exit;
}

//close connection
$conn->close();
?>
```

The output form tested with invalid email format will be as shown in figure 5.6



First Name:	Mary
Last Name:	Momba
Sex:	[✓ Female] Male
E-mail:	mmary/
Birthdate:	Day: 23 Month: 02 Year: 1987

Invalid email format

Figure 5.6 A tested Form

Exercises

1. What is a query string?
2. Which function would you use to start or resume a session within a PHP script?
3. How would you create and destroy cookies with PHP
4. How can you retain form data after posting
5. Which function is used for browser redirection?
6. How would you end a session and erase all traces of it for future visits?
7. Why PHP sessions use a dual cookie/URL method to store session ID?
8. Create a script that uses session functions to remember which pages in your environment the user has visited. Provide the user with a list of links on each page to make it easy for her to retrace her steps.

Chapter 6: Implementing Authentication with PHP and MySQL

In this chapter, we are going to discuss on how to implement various php and mysql techniques for authenticating users.

Key topics covered in this chapter include

- Identifying visitors
- Implementing access control
- Using basic authentication
- Using basic authentication in PHP
- Creating your own custom authentication

6.1 Identifying Visitors

The Web is a fairly anonymous medium, but it is often useful to know who is visiting your site. Fortunately for visitors' privacy, you can find out very little about them without their assistance. With a little work, servers can find out quite a lot about the computers and networks that connect to them, however. A web browser usually identifies itself, telling the server what browser, browser version, and operating system a user is running. You can often determine what resolution and color depth visitors' screens are set to and how large their web browser windows are by using JavaScript.

Each computer connected to the Internet has a unique IP address. From a visitor's IP address, you might be able to deduce a little about him/her. You can find out who owns an IP and sometimes make a reasonable guess as to a visitor's geographic location.

Some addresses are more useful than others. Generally, people with permanent Internet connections have a permanent address. Customers dialing into an ISP usually get only the temporary use of one of the ISP's addresses. The next time you see that address, it might be used by a different computer, and the next time you see that visitor, he/she will likely be using a different IP address. IP addresses are not as useful for identifying people as they might at first glance seem.

Fortunately for web users, none of the information that their browsers give out identifies them. If you want to know a visitor's name or other details, you will have to ask him/her.

Most e-commerce sites record their customers' details when they make their first order. This means that a customer is not required to type his/her details every time.

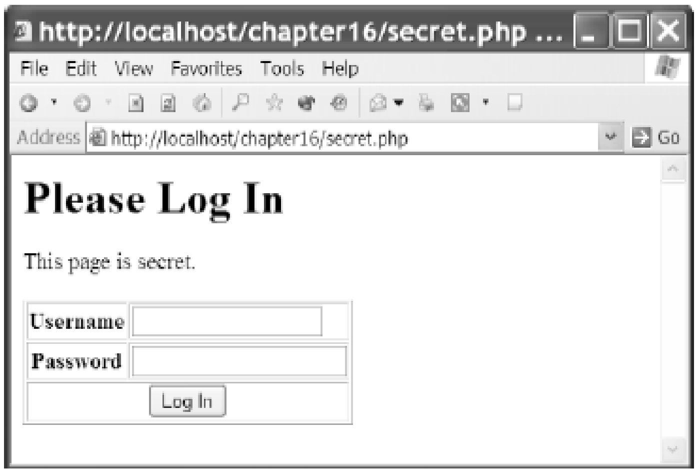
Having asked for and received information from your visitor, you need a way to associate the information with the same user the next time he/she visits. If you are willing to make the assumption that only one person visits your site from a particular account on a particular machine and that each visitor uses only one machine, you could store a cookie on the user's machine to identify the user.

This arrangement is certainly not true for all users. Many people share a computer, and many people use more than one computer. At least some of the time, you need to ask a visitor who she/he is again. In addition to asking who a user is, you also need to ask her to provide some level of proof that he/she is who he/she claims to be.

Asking a user to prove his/her identity is called *authentication*. The usual method of authentication used on websites is asking visitors to provide a unique login name and a password. Authentication is usually used to allow or disallow access to particular pages or resources, but can be optional, or used for other purposes such as personalization.

6.2 Implementing simple Access Control

Simple access control is not difficult to implement. The code shown below delivers one of three possible outputs. If the file is loaded without parameters, it will display an HTML form requesting a username and password. This type of form is shown in figure 6.1.


http://localhost/chapter16/secret.php ...
File Edit View Favorites Tools Help
Address http://localhost/chapter16/secret.php
Go

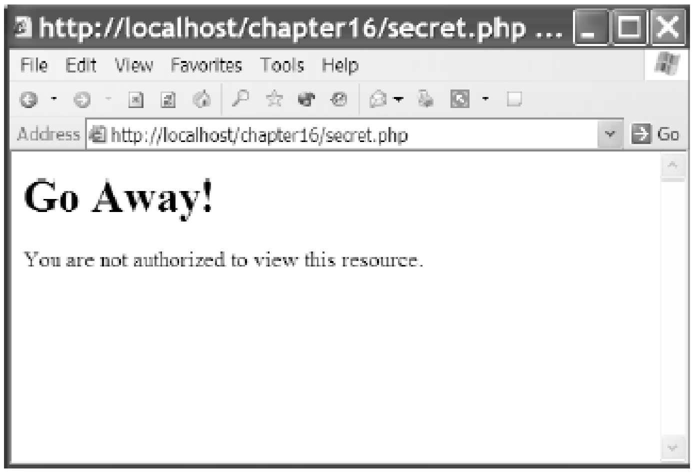
Please Log In

This page is secret.

Username	
Password	
Log In	

Figure 6.1 Login form

If the parameters are present but not correct, it will display an error message. A sample error message is shown in Figure 6.2. On a real site, you might want to give a somewhat friendlier message.

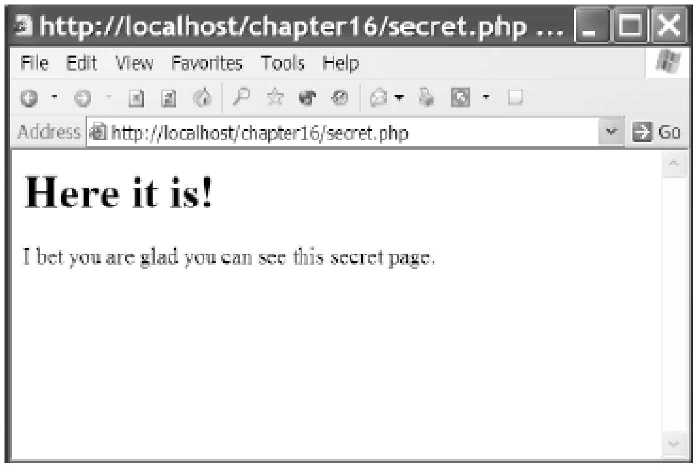

http://localhost/chapter16/secret.php ...
File Edit View Favorites Tools Help
Address http://localhost/chapter16/secret.php
Go

Go Away!

You are not authorized to view this resource.

Figure 6.2 Login error page

If these parameters are present and correct, it will display the secret content. The sample test content is shown in figure 6.3.


http://localhost/chapter16/secret.php ...
File Edit View Favorites Tools Help
Address http://localhost/chapter16/secret.php
Go

Here it is!

I bet you are glad you can see this secret page.

Figure 6.3 correct login page

The code to create the functionality shown in Figures 6.1, 6.2, and 6.3 is shown in

secret.php—PHP and HTML to Provide a Simple Authentication Mechanism

```
<?php
//create short names for variables
@ $name = $_POST['name'];
@ $password = $_POST['password'];
if(empty($name)||empty($password))
{
//Visitor needs to enter a name and password
?>
<h1>Please Log In</h1>
This page is secret.
<form method="post" action="secret.php">
<table border="1">
<tr>
<th> Username </th>
<td> <input type="text" name="name"> </td>
</tr>
<tr>
<th> Password </th>
<td> <input type="password" name="password"> </td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="Log In">
</td>
</tr>
</table>
</form>
<?php
}
```

```
else if($name=='user' && $password=='pass')
{
// visitor's name and password combination are correct
echo '<h1>Here it is!</h1>';
echo 'I bet you are glad you can see this secret page.';
}
else
{
// visitor's name and password combination are not correct
echo '<h1>Go Away!</h1>';
echo 'You are not authorized to view this resource.';
}
?>
```

The code above provides a simple authentication mechanism to allow authorized users to see a page, but it has some significant problems. This script

1. Has one username and password hard-coded into the script
2. Stores the password as plain text
3. Protects only one page
4. Transmits the password as plain text

These issues can all be addressed with varying degrees of effort and success.

6.2.1 Storing Passwords

There are many better places to store usernames and passwords than inside the script. Inside the script, modifying the data is difficult. It is possible, but a bad idea, to write a script to modify itself. Doing so would mean having a script on your server that is executed on your server but that can be written or modified by others. Storing the data in another file on the server lets you more easily write a program to add and remove users and to alter passwords.

Inside a script or another data file, you are limited to the number of users you can have without seriously affecting the speed of the script. If you are considering storing and searching through a large number of items in a file, you should consider using a database instead, as previously discussed. As a rule of thumb, if you want to store and search through a list of more than 100 items, they should be in a database rather than a flat file.

Using a database to store usernames and passwords would not make the script much more complex but would allow you to authenticate many different users quickly. It would also allow you to easily write a script to add new users, delete users, and allow users to change their passwords.

A script to authenticate visitors to a page against a database is shown below

secretdb.php—Using MySQL to Improve the Simple Authentication Mechanism

```
<?php
$name = $_POST['name'];
```

```
$password = $_POST['password'];
if(!isset($_POST['name'])&&!isset($_POST['password']))
{
//Visitor needs to enter a name and password
?>
<h1>Please Log In</h1>
 This page is secret.
<form method="post" action="secretdb.php">
<table border="1">
<tr>
<th> Username </th>
<td> <input type="text" name="name"> </td>
</tr>
<tr>
<th> Password </th>
<td> <input type="password" name="password"> </td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="Log In">
</td>
</tr>
</table>
</form>
<?php
}
else
{
// connect to mysql
$mysql = mysqli_connect('localhost', 'webauth', 'webauth');
if(!$mysql)
{
echo 'Cannot connect to database.';
exit;
}
// select the appropriate database
$selected = mysqli_select_db($mysql, 'auth');
if(!$selected)
{
echo 'Cannot select database.';
exit;
}
// query the database to see if there is a record which matches
$query = "select count(*) from authorized_users where name = '$name' and
password = '$password'";
$result = mysqli_query($mysql, $query);
if(!$result)
{
echo 'Cannot run query.';
exit;
}
$count = mysqli_num_rows($result)
if($count>0)
{
```

```
// visitor's name and password combination are correct
echo '<h1>Here it is!</h1>';
echo 'I bet you are glad you can see this secret page.';
}
else
{
// visitor's name and password combination are not correct
echo '<h1>Go Away!</h1>';
echo 'You are not authorized to view this resource.';
}
}
?>
```

You can create the database used here by connecting to MySQL as the MySQL root user and running the contents of the code below

createauthdb.sql— These MySQL Queries create the auth Database, the auth Table, and Two Sample Users

```
create database auth;
use auth;
create table authorized_users (name varchar(20),password varchar(40), primary key
(name));
insert into authorized_users values ('username','password');
insert into authorized_users values ('testuser',sha1('password'));
grant select on auth.* to 'webauth' identified by 'webauth';
flush privileges;
```

6.2.2 Encrypting Passwords

Regardless of whether you store your data in a database or a file, storing the passwords as plain text is an unnecessary risk. A one-way hashing algorithm can provide better security with very little extra effort.

PHP provides a number of one-way hash functions. The oldest and least secure is the Unix Crypt algorithm, provided by the function crypt(). The Message Digest 5 (MD5) algorithm, implemented in the function md5(), is stronger and available in most versions of PHP. If you do not require compatibility with old PHP versions, use Secure Hash Algorithm 1 (SHA-1).

The PHP function sha1() provides a strong, one-way cryptographic hash function. The prototype for this function is

```
string sha1(string str [,bool raw_output])
```

Given the string str, the function will return a pseudo-random 40-character string. If you set raw_output to be true, you will instead get a 20-character string of binary data.

For example, given the string "password", sha1() returns "5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8". This string cannot be decrypted and turned back into "password" even by its creator, so it might not seem very useful at first glance.

The property that makes sha1()useful is that the output is deterministic. Given the same string, sha1() will return the same result every time it is run.

Rather than having PHP code like

```
if($name == 'username' && $password == 'password')
{
//OK passwords match
}
```

you can have code like

```
if($name=='username' &&
 sha1($password)=='5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8'))
  {
  //OK passwords match
  }
```

You do not need to know what the password looked like before you used sha1() on it. You need to know only if the password typed in is the same as the one that was originally run through sha1().

As already mentioned, hard-coding acceptable usernames and passwords into a script is a bad idea. You should use a separate file or a database to store them.

If you are using a MySQL database to store your authentication data, you could either use the PHP function sha1() or the MySQL functions SHA1() and Password(). MySQL provides an even wider range of hashing algorithms than PHP, but they are all intended for the same purpose.

To use SHA1(), you could rewrite the SQL query in secretdb.php as

```
select count(*) from authorized_users where
name = '$username' and password = sha1('$password')
```

This query counts the number of rows in the table named authorized_users that have a name value equal to the contents of $name and a pass value equal to the output given by SHA1() applied to the contents of $password. Assuming that you force people to have unique usernames, the result of this query is either 0 or 1.

Keep in mind that the hash functions generally return data of a fixed size. In the case of SHA1, it is 40 characters when represented as a string. Make sure that your database column is this width.
Looking back at createauthdb.sql, you can see that we created one user ('username') with an unencrypted password and another user with an encrypted one ('testuser') to illustrate the two possible approaches.

6.2.3 Protecting Multiple Pages

Making a script like the ones in secret.php and secretdb.php protect more than one page is a little harder. Because HTTP is stateless, there is no automatic link or association between subsequent requests from the same person. This makes it harder to have data, such as authentication information that a user has entered, carry across from page to page.

The easiest way to protect multiple pages is to use the access control mechanisms provided by your web server. We look at these mechanisms shortly.

To create this functionality yourself, you could include parts of the script shown in secret.php in every page that you want to protect. Using auto_prepend_file and auto_append_file, you can automatically prepend and append the code required to every file in particular directories.

If you use this approach, what happens when your visitors go to multiple pages within your site? Requiring them to re-enter their names and passwords for every page they want to view would not be acceptable.

You could append the details the users entered to every hyperlink on the page. Because they might have spaces or other characters that are not allowed in URLs, you should use the function urlencode() to safely encode these characters.

This approach still has a few problems, though. Because the data would be included in web pages sent to the users and the URLs they visit, the protected pages they visit will be visible to anybody who uses the same computer and steps back through cached pages or looks at the browser's history list. Because you are sending the password back and forth to the browser with every page requested or delivered, this sensitive information is being transmitted more often than necessary.

There are two good ways to tackle these problems: HTTP basic authentication and sessions. Basic authentication overcomes the caching problem, but the browser still sends the password to the server with every request. Session control overcomes both of these problems. We look at HTTP basic authentication now and later session control

6.3 Using Basic Authentication

Fortunately, authenticating users is a common task, so authentication facilities are built into HTTP. Scripts or web servers can request authentication from a web browser. The web browser is then responsible for displaying a dialog box or similar device to obtain required information from the user.

Although the web server requests new authentication details for every user request, the web browser does not need to request the user's details for every page. The browser generally stores these details for as long as the user has a browser window open and automatically resends them to the web server as required without user interaction.

This feature of HTTP is called *basic authentication*. You can trigger basic authentication using PHP or using mechanisms built into your web server. We look first at the PHP method, then the Apache method, and finally the IIS method.

Basic authentication transmits a user's name and password in plain text, so it is not very secure. HTTP 1.1 contains a more secure method known as *digest authentication*, which uses a hashing algorithm (usually MD5) to disguise the details of the transaction.

Digest authentication is supported by many web servers and most current-version web browsers. Unfortunately, as with many recently implemented features, there are many older browsers still in use that do not support digest authentication and a version of the standard included in Microsoft IE and IIS that is not compatible with non-Microsoft products.

In addition to being poorly supported by installed web browsers, digest authentication is still not very secure. Both basic and digest authentication provide a low level of security. Neither gives the user any assurance that he is dealing with the machine he intended to access. Both might permit a cracker to replay the same request to the server. Because basic authentication transmits the user's password as plain text, it allows any cracker capable of capturing packets to impersonate the user for making any request.

Basic authentication provides a (low) level of security similar to that commonly used to connect to machines via Telnet or FTP, transmitting passwords in plain text. Digest authentication is somewhat more secure, encrypting passwords before transmitting them.

However, for many situations, a fast, but relatively insecure, method such as basic authentication is appropriate.

Basic authentication protects a named realm and requires users to provide a valid username and password. Realms are named so that more than one realm can be on the same server. Different files or directories on the same server can be part of different realms, each protected by a different set of names and passwords. Named realms also let you group multiple directories on the one host or virtual host as a realm and protect them all with one password.

6.3.1 Using Basic Authentication in PHP

PHP scripts are generally cross-platform, but using basic authentication relies on environment variables set by the server. For an HTTP authentication script to run on Apache using PHP as an Apache module or on IIS using PHP as an ISAPI module, it needs to detect the server type and behave slightly differently. The script below will run on Apache servers.

http.php—PHP Can Trigger HTTP Basic Authentication

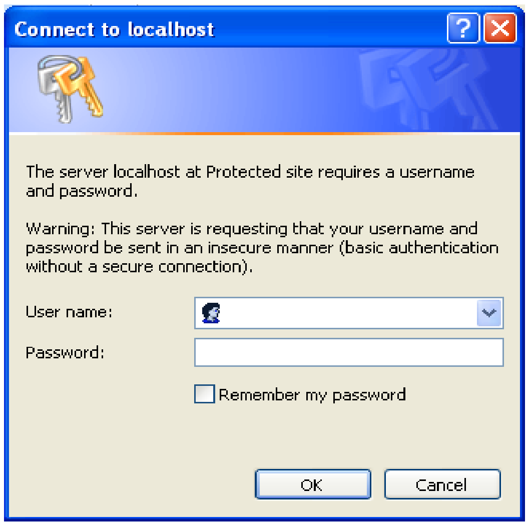
```
<?php
  /*
  ** Define a couple of functions for
  ** starting and ending an HTML document
  */
  function startPage()
  {
    print("<html>\n");
    print("<head>\n");
    print("<title>Listing 24-1</title>\n");
    print("</head>\n");
    print("<body>\n");
  }
  function endPage()
  {
    print("</body>\n");
    print("</html>\n");
  }
  /*
  ** test for username/password
  */
```

```
if((isset($_SERVER['PHP_AUTH_USER'])&&($_SERVER['PHP_AUTH_USER']=="peter"))
AND(isset($_SERVER['PHP_AUTH_PW'])&&($_SERVER['PHP_AUTH_PW']=="pass")))
   {
      startPage();
      print("You have logged in successfully!<br>\n");
      endPage();
   }
   else
   {
      //Send headers to cause a browser to request
      //username and password from user
      header("WWW-Authenticate: " .
         "Basic realm=\"Protected site\"");
      header("HTTP/1.0 401 Unauthorized");

      //Show failure text, which browsers usually
      //show only after several failed attempts
      print("This page is protected by HTTP " .
         "Authentication.<br>\nUse <b>peter</b> " .
         "for the username, and <b>pass</b> " .
         "for the password.<br>\n");
   }
?>
```

The code above acts similarly to the previous code in this chapter. If the user has not yet provided authentication information, it will be requested. If she/he has provided incorrect information, she/he is given a rejection message. If she/he provides a matching name and password pair, she/he is presented with the contents of the page.

In this case, the user will see an interface somewhat different from the previous code.

This script does not provide an HTML form for login information. The user's browser presents her/him with a dialog box. Some people see this as an improvement; others would prefer to have complete control over the visual aspects of the interface. The login dialog box that Internet Explorer provides is shown in Figure 6.4.


Connect to localhost

The server localhost at Protected site requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

User name:

Password:

Remember my password

Figure 6.4 login dialog box

Because the authentication is being assisted by features built into the browser, the browser chooses to exercise some discretion in how failed authorization attempts are handled. Internet Explorer lets the user try to authenticate three times before displaying the rejection page. Netscape Navigator lets the user try an unlimited number of times, popping up a dialog box to ask, “Authorization failed. Retry?” between attempts. Netscape displays the rejection page only if the user clicks Cancel.

As with the code given in secret.php and secretdb.php, you could include this code in pages you wanted to protect or automatically prepend it to every file in a directory.

For more flexibility and the ability to apply fine-grained control to parts of pages, you might want to implement your own authentication using PHP and MySQL.

6.4 Implementing Custom Authentication with Session Control

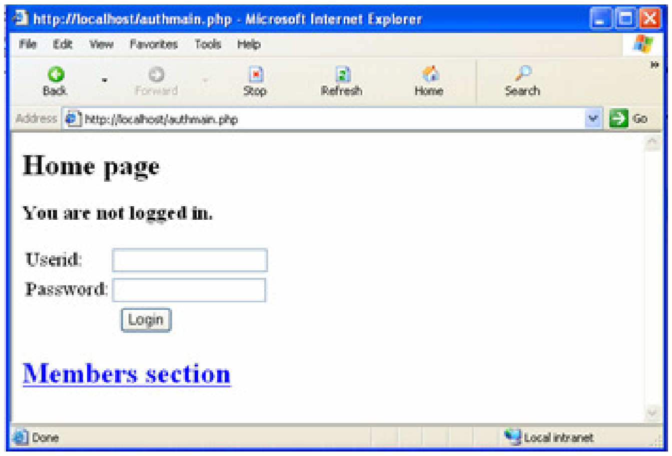
In the beginning of this chapter, you looked at creating your own authentication methods including some flaws and compromises and using built-in authentication methods, which are less flexible than writing your own code.

Now, we are going to look at a more substantial example using session control.

Possibly the most common use of session control is to keep track of users after they have been authenticated via a login mechanism. In this example, you combine authentication from a MySQL database with use of sessions to provide this functionality.

The example consists of three simple scripts. The first, authmain.php, provides a login form and authentication for members of the website. The second, members_only.php, displays information only to members who have logged in successfully. The third, logout.php, logs out a member.

To understand how this example works, look at Figure 6.5, which shows the initial page displayed by authmain.php.


http://localhost/authmain.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost/authmain.php
Go
Done
Local intranet

Home page

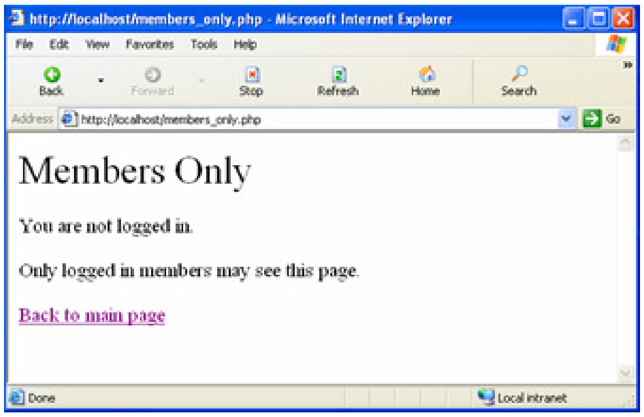
You are not logged in.

Userid:
Password:
Login

Members section

Figure 6.5 initial page

This page gives the user a place to log in. If she/he attempts to access the Members section without logging in first, she/he will get the message shown in Figure 6.6.


http://localhost/members_only.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost/members_only.php Go
Done
Local intranet

Members Only

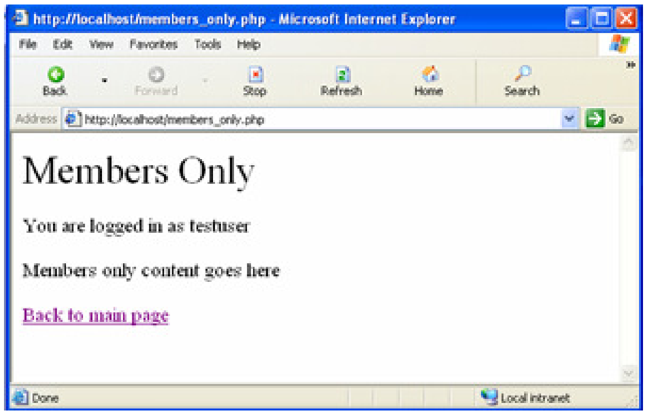
You are not logged in.

Only logged in members may see this page.

Back to main page

Figure 6.6 fail to login page

If the user logs in first (with username: testuser and password: password, as set up in system), however, and then attempts to see the Members page, she/he will get the output


http://localhost/members_only.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost/members_only.php Go
Done
Local intranet

Members Only

You are logged in as testuser

Members only content goes here

Back to main page

shown in Figure 6.7

Figure 6.7 proper login page

First, let's look at the code for this application. Most of the code is in authmain.php, shown below. Then we'll go through it bit by bit.

authmain.php—The Main Part of the Authentication Application

```
<?php
session_start();
if(isset($_POST['userid']) && isset($_POST['password']))
{
// if the user has just tried to log in
$userid = $_POST['userid'];
$password = $_POST['password'];
$db_conn = new mysqli('localhost', 'webauth', 'webauth', 'auth');
if(mysqli_conect_errno()) {
echo 'Connection to database failed:'.mysqli_conect_error();
exit();
}
$query = "select * from authorized_users "
."where name='$userid' "
." and password=sha1('$password')";
```

```
$result = $db_conn->query($query);
if ($result->num_rows>0 )
{
/ if they are in the database register the user id
$_SESSION['valid_user'] = $userid;
}
$db_conn->close();
}
?>
<html>
<body>
<h1>Home page</h1>
<?
if(isset($_SESION['valid_user'])
{
echo 'You are logged in as: '.$_SESION['valid_user'].' <br />';
echo '<a href="logout.php">Log out</a><br />';
}
else
{
if(isset($userid)
{
// if they've tried and failed to log in
echo 'Could not log you in.<br />';
}
else
{
// they have not tried to log in yet or have logged out
echo 'You are not logged in.<br />';
}
//provide form to log in
echo '<form method="post" action="authmain.php">';
echo '<table>';
echo '<tr><td>Userid:</td>';
echo '<td><input type="text" name="userid"></td></tr>';
echo '<tr><td>Password:</td>';
echo '<td><input type="password" name="password"></td></tr>';
echo '<tr><td colspan="2" align="center">';
echo '<input type="submit" value="Log in"></td></tr>';
echo '</table></form>';
}
?>
<br />
<a href="members_only.php">Members section</a>
</body>
</html>
```

Some reasonably complicated logic is included in this script because it displays the login form, is also the action of the form, and contains HTML for a successful and failed login attempt.

The script's activities revolve around the valid_user session variable. The basic idea is that if someone logs in successfully, you will register a session variable called $_SESSION['valid_user'] that contains her/his userid.

The first thing you do in the script is call session_start(). This call loads in the session variable valid_user if it has been created.

In the first pass through the script, none of the if conditions apply, so the user falls through to the end of the script, where you tell her/him that she/he is not logged in and provide her/him with a form to do so:

```
echo ‘<form method=”post” action=”authmain.php”>’;
echo ‘<table>’;
echo ‘<tr><td>Userid:</td>’;
echo ‘<td><input type=”text” name=”userid”></td></tr>’;
echo ‘<tr><td>Password:</td>’;
echo ‘<td><input type=”password” name=”password”></td></tr>’;
echo ‘<tr><td colspan=”2” align=”center”>’;
echo ‘<input type=”submit” value=”Log in”></td></tr>’;
echo ‘</table></form>’;
```

When the user clicks the submit button on the form, this script is reinvoked, and you start again from the top. This time, you will have a userid and password to authenticate, stored as $_POST[‘userid’] and $_POST[‘password’]. If these variables are set, you go into the authentication block:

```
if (isset($_POST[‘userid’]) & isset($_POST[‘password’]))
{
// if the user has just tried to log in
$userid = $_POST[‘userid’];
$password = $_POST[‘password’];
$db_conn = new mysqli(‘localhost’, ‘webauth’, ‘webauth’, ‘auth’);
if (mysqli_conect_errno()) {
echo ‘Connection to database failed:’.mysqli_conect_error();
exit();
}
$query = “select * from authorized_users “
.”where name=’$userid’ “
.” and password=sha1(‘$password’)”;
$result = $db_conn->query($query);
```

You connect to a MySQL database and check the userid and password. If they are a matching pair in the database, you create the variable $_SESSION[‘valid_user’], which contains the userid for this particular user, so you know who is logged in further down the track:

```
if ($result->num_rows >0 )
{
// if they are in the database register the user id
$_SESSION[‘valid_user’] = $userid;
}
$db_conn->close();
}
```

Because you now know who the user is, you don’t need to show her/him the login form again. Instead, you can tell her/him you know who she/he is and give her/him the option to log out:

```
if (isset($_SESSION[‘valid_user’]))
{
echo ‘You are logged in as: ‘.$_SESION[‘valid_user’].’ <br />’;
echo ‘<a href=”logout.php”>Log out</a><br />’;
}
```

logout.php— This Script Deregisters the Session Variable and Destroys the Session

```
<?php
session_start();
// store to test if they *were* logged in
$old_user = $_SESSION[‘valid_user’];
unset($_SESSION[‘valid_user’]);
session_destroy();
?>
<html>
<body>
<h1>Log out</h1>
<?php
if (!empty($old_user)
{
echo ‘Logged out.<br />’;
}
else
{
// if they weren’t logged in but came to this page somehow
echo‘You were not logged in,and so have not been logged out<br>’;
}
?>
<a href=”authmain.php”>Back to main page</a>
</body>
</html>
```

This code is simple, but you need to do a little fancy footwork. You start a session, store the user’s old username, unset the valid_user variable, and destroy the session. You then give the user a message that will be different if she/he was logged out or was not logged in to begin with.

Exercises

1. What are the advantages of database files over plain text files for storing user authentication information?
2. Can you name some disadvantages of HTTP basic authentication?
3. What PHP function is designed to allow you to set strong password encryptions?
4. Is there other encryption functions that are supported by PHP and MySQL ?
5. Practice using the various types of authentication both server-based and with PHP on your development server. Get a feel for the differences between basic HTTP authentication and something you devise on your own.

References

1. Fast Track Web Programming: A programmer's guide to Mastering Web Technologies, by D. Cintron (1101010), John Wiley and Sons, New York.
2. Core PHP Programming: Using PHP to Build Dynamic. by Leon Atkinson (2000). Web Sites (2nd Edition).
3. Creating Database Web Applications with PHP and ASP (Internet Series) by Jeanine Meyer, Publisher: Charles River Media; 1 edition (June 30, 2003) ISBN: 1584502649
4. Database-Driven Web Sites , By Kristin Antelman (Editor), Publisher: Haworth Information Press (September 2002) , ISBN: 0789017385