ЎЗБЕКИСТОН РЕСПУБЛИКАСИ АЛОҚА, АХБОРОТЛАШТИРИШ ВА ТЕЛЕКОММУНИКАЦИЯ ТЕХНОЛОГИЯЛАРИ ДАВЛАТ ҚЎМИТАСИ
ТОШКЕНТ АХБОРОТТЕХНОЛОГИЯЛАРИ УНИВЕРСИТЕТИ
ФАРҒОНА ФИЛИАЛИ
«АХБОРОТ ТЕХНОЛОГИЯЛАРИ» КАФЕДРАСИ

«ҲИМОЯГА»

Кафедра мудири

_____

« _____ »_____2014 й

ВИДЕОМАЪЛУМОТ УЗАТИШ ТАРМОҚЛАРИДА MPLS ПРОТОКОЛИНИ ҚЎЛЛАШ

МАВЗУСИДА

МАЛАКАВИЙ БИТИРУВ ИШИ

БИТИРУВЧИ: А.Саматов
610-10 гуруҳ талабаси

Фарғона 2014

CONTENTS

ANNOTATION

The student of the 4th form of the of the direction of computer science and information technologies in Fergana branch of The Tashkent University of Information Technologies Turg'unliyev Bekzod used modern programming technologies in his diploma work on the theme about creating Interactive service for reception high educational establishments.

During the period of the preparation diploma work was used PHP5 and MySQL Diploma work consists of four part. The first part analytical part contains the about common interactive services and how to create that kind interactive services with PHP. Second part is project part work with database and send special queries from MySQL. The third part is called creation and implementing part has its safety and easy to use . The last part is consist of defence of labour part.

My goal of choosing this theme was creating useful reception of interactive service for applicants. They can see daily applicants tables and last year how many students had applied and how many students applied.

President Islam Karimov Addresses the International Conference in Samarkand

"The goal sought by our conference is to discuss and thoroughly perceive of the scientific legacy of the outstanding scholars and thinkers of the medieval East, evaluate their role and place in the history of the modern civilization.

I believe it is also important to afford a new potent impetus to the further intensive research works and the popularization of their genuinely invaluable scientific heritage, to reveal the urgency and demand in the discoveries made by them for the contemporary science and progress."

Introduction

If you've been watching the Web for a while you've probably noticed it is changing. When the Web first entered into the public consciousness, it was a way to distribute documents. These documents were pretty easy to make. Anybody with a weekend and a text editor could get a Web page up and running. Building a Web site in the early days was about making documents.

Today the Internet is much more than that. Interesting Web sites are not simply documents; they are applications. They have much more complexity and power. You might think the Web is no longer a place for individuals or beginning programmers. Many of the software development tools available are expensive and complicated.

PHP is a powerful programming language that lets you build dynamic Web sites. It works well on a variety of platforms, and it's reasonably easy to understand. MySQL is an impressive relational data management system used to build commercial quality databases. PHP and MySQL are such powerful and easy-to-use platforms that they make Web programming accessible even for beginners.

.

Programming on the Web Server

The Internet is all about various computers communicating with each other. The prevailing model of the Internet is the notion of clients and servers. You can understand this better by imagining a drive-through restaurant. As you drive to the little speaker, a barely intelligible voice asks for your order. You ask for your cholestoburger supreme and the teenager packages your food. You drive up, exchange money for the meal, and drive away. Meanwhile, the teenager waits for another customer to appear. The Internet works much like this model. Large permanent computers called *Web servers* host Web pages and other information. They are much like the drive-through restaurant. Users drive up to the Web server using a Web browser. The data is exchanged and the user can read the information on the Web browser. What's interesting about this model is the interaction doesn't have to stop there. Since the client (user's) machine is a computer, it can be given instructions. Commonly, the JavaScript language stores special instructions in a Web page. These instructions (like the HTML code itself) don't mean anything on the server. Once the page gets to the client machine, the browser interprets the HTML code and any other JavaScript instructions.

While much of the work is passed to the client, there are some disadvantages to this client-side approach. Programs designed to work inside a Web browser are usually greatly restricted in the kinds of things they can do. A client-side Web program usually cannot access the user's printer or disk drives. This limitation alone prevents such programs from doing much of the most useful work of the Internet, such as database connectivity and user tracking.

The server is also a computer; it's possible to write programs designed to operate on the server rather than the client. This arrangement has a number of advantages:

- Server-side programs run on powerful Web server computers.
- The server can freely work with files and databases.
- The code returned to the user is plain HTML, which can be displayed on any Web browser.

1.1 Adding basic PHP code to a Web page

All this HTML is nice, but presumably you're here to learn PHP, so it's high time to add PHP code to a page. PHP can be used to add characteristics to your page that aren't typically possible with normal HTML and CSS.

A page written in PHP can be identical to an HTML page. Both are written with a plain text editor, and stored on a Web server. A PHP program can have <script> elements embedded in the page. When the user requests a PHP page, the server first examines the page and executes any script elements before it sends the resulting HTML to the user. This will only work if the Web server has been configured to use the PHP language. You might need to check with your server administrator to see if this support is available. On a home computer, you can use the PHP Tripod software included on the CD-ROM that accompanies this book to set up all the necessary components.

The easiest way to determine if PHP exists on your server is to write a simple PHP program and see if it works. Here's a very simple PHP program.

```
<html>
<head>
<title>Hello in PHP</title>
</head>
<body>
<h1>Hello in PHP</h1>
<?
print "Hello, world!";
phpInfo();
?>
</body> </html>
```

A PHP program looks a lot like a typical HTML page. The only thing that's different is the special <? ?> tag. This tag specifies the existence of PHP code. Any code inside the tag will be read by the PHP interpreter, then converted into HTML

code. The code written between the <? and ?> symbols is PHP code. I added two commands to the page. Look at the output of the program shown in Figure 1.1, and you might be surprised:

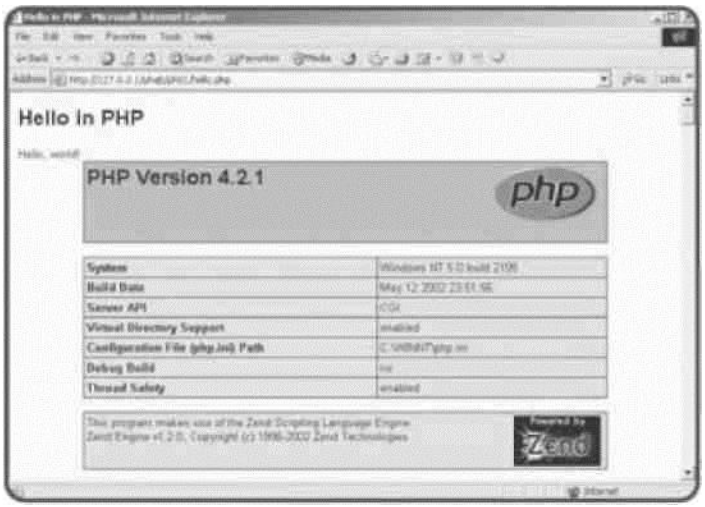


Figure 1.1: The page mixes HTML with some other things.

Examining the Results

There are three distinct types of text on this page. First, the "Hello in PHP" line is ordinary HTML. I wrote it just like a regular HTML page, and it was displayed just like regular HTML. The "Hello world" line is a little different though, because it was written by the PHP program embedded in the page. The rest of the page is a bit mysterious. It contains a lot of information about the particular PHP engine being used. It actually stretches on for several pages. All that code was generated by the phpInfo() command. This command is used to display information about the PHP installation. It isn't that important to understand all the information displayed by the phpInfo() command. It's much more critical to appreciate that when the user requests the hello.html Web page, the text of the page is first run through the PHP interpreter. This program scans for any PHP commands, executes the commands, and prints HTML code in place of the original commands. By the time a page gets to the user, all the PHP code is gone, because the server used the PHP to generate HTML code. For proof of this, point your browser at hello.php and

then view the source code. It will look something like this:

<html>

<head>

<title>Hello in PHP</title>

</head>

<body>

<h1>Hello in PHP</h1>

Hello, world!<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head><style type="text/css"><!--

a { text-decoration: none; }

a:hover { text-decoration: underline; }

h1 { font-family: arial, helvetica, sans-serif; font-size: 18pt; font-weight: bold

h2 { font-family: arial, helvetica, sans-serif; font-size: 14pt; font-weight: bold

body, td { font-family: arial, helvetica, sans-serif; font-size: 10pt; }

th { font-family: arial, helvetica, sans-serif; font-size: 11pt; font-weight: bold

//--></style>

<title>phpinfo()</title></head><body><table border="0" cellpadding="3" cellspacing=

width="600" bgcolor="#000000" align="center">

<tr valign="middle" bgcolor="#9999cc"><td align="left">

<img src="/phab/ph01/hello.php?=PHPE9568F34-D428-11d2

00AA001ACF42" border=0 align="right" alt="PHP Logo"><h1>PHP Version 4.2.1</h1>

Note that I showed only a small part of the code generated by the phpInfo () command, and that the details of the code might be different when you run the program on your own machine. The key point here is the PHP code to write "Hello World!" (print "Hello World!") is replaced with the actual text "Hello World!"

More significantly, the very simple phpInfo() command is replaced by a huge amount of HTML code.

A small amount of PHP code can very efficiently generate large and complex HTML documents. This is one significant advantage of PHP. Also, by the time the document gets to the Web browser, it's plain vanilla HTML code, which can be read easily by any browser. These two features are important benefits of server-side programming in general, and of PHP programming in particular. As you progress through this book, you'll learn about many more commands for producing interesting HTML, but the basic concept is always the same. Your PHP program is simply an HTML page that contains special PHP markup. The PHP code is examined by a special program on the server, and the results are embedded into the Web page before it is sent to the user.

Way back at the beginning of the part I promised you would be able to

write the "Tip of the day" program featured at the beginning of the part. This program requires HTML, Cascading Style Sheets, and one line of PHP code. The code shows a reasonably basic page.

```
<html>
<head>
<title>Tip of the day</title>
</head>
<body>
<center>
<h1>Tip of the day</h1>
<div style = "border-color:green; border-style:groove; border-width:2px">
<?
readfile("tips.txt");
?>
</div>
</center>
```

</body>

</html>

The page is basic HTML. It contains one div element with a custom style setting up a border around the tip of the day. Inside the div element, I added PHP code with the <? and ?> devices. This code calls one PHP function called readFile(). The readFile() command takes as an argument the name of some file. It reads the contents of that file and displays it onto the page as if it were HTML. As soon as that line of code stops executing (that is, the text in the tips.txt file has been printed to the Web browser) the ?> symbol indicates that the PHP coding is finished and the rest of the page will be typical HTML.

By the end of this part, you'll be able to write the program featured in Figures 1.2

Figures 1.2 and 1.3.

Figure 1.3: The program begins by asking the user to enter some information.

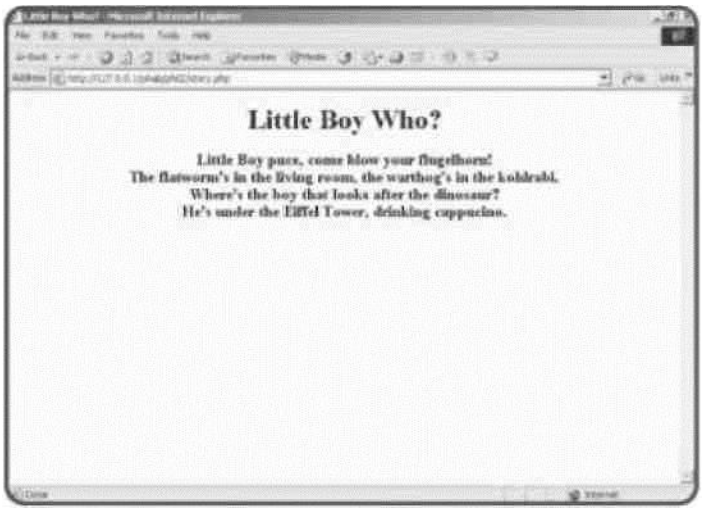


Figure 1.3: I hate it when the warthog's in the kohlrabi.

The program asks the user to enter some values into an HTML form, and then uses those values to build a custom version of a classic nursery rhyme. The story program works like most server-side programs. It has two distinctive parts. First, the user enters information into a plain HTML form and hits the submit button. The PHP program doesn't execute until after the user has submitted a form. The program takes the information from the form and does something to it. Usually the PHP program also returns an HTML page to the user.

Using Variables in Your Scripts

The most important new idea in this part is the notion of a variable. A variable is a container for holding information in the computer's memory. To make things easier for the programmer, every variable has a name. You can store information into a variable and get information out of a variable.

The program featured in Figure 1.4 uses a variable, although you might not be able to tell simply by looking at the output.

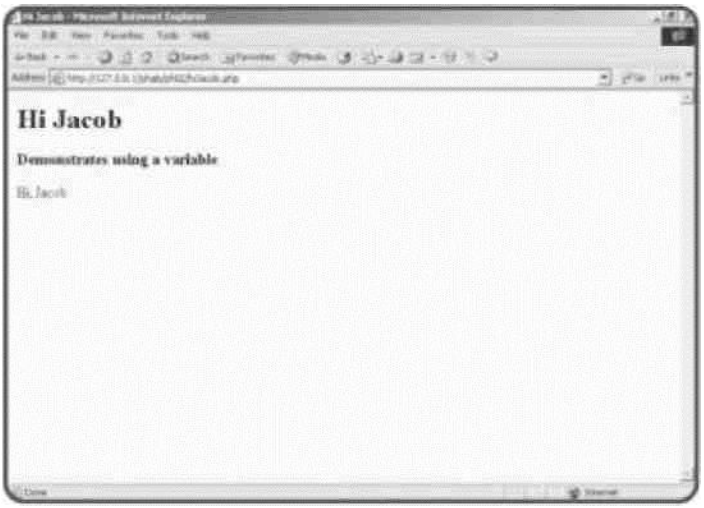


Figure 1.4: The word "Jacob" is stored in a variable in this page. You can't really see anything special about this program from the Web page itself (even if you look at the HTML source). To see what's new, look at the source code of hiJacob.php.

```
<html>
<head>
<title>Hi Jacob</title>
</head>
<body>
<h1>Hi Jacob</h1>
<h3>Demonstrates using a variable</h3>
<?
$userName = "Jacob";
print "Hi, $userName";
?>
</body>
</html>
```

The hi User page is mainly HTML with a small patch of PHP code in it. That code does a lot of very important work.

Creating a String Variable

The line $userName = "Jacob"; does a number of things. First, it creates a variable named $userName. In PHP, all variables begin with a dollar sign to distinguish them from other program elements. The variable's name is significant.

As a programmer, you will frequently get to name things. Experienced programmers have learned some tricks about naming variables and other elements.

- Make the name descriptive. It's much easier to figure out what $userName means than something like $myVariable or $r. When possible, make sure your variable names describe the kind of information they contain.
- Use an appropriate length. Your variable name should be long enough to be descriptive, but not so long that it becomes tedious to type.
- Don't use spaces. Most languages (including PHP) don't allow spaces in variable names.
- Don't use punctuation. Most of the special characters such as #, *, and / already have meaning in programming languages, so they can't be used in variable names. Of course, every variable in PHP begins with the $ character, but otherwise you should avoid using punctuation. One exception to this rule is the underscore (_) character, which is allowed in most languages, including PHP.
- Be careful about case. PHP is a case-sensitive language, which means that it considers $userName, $USERNAME, and $UserName to be three different variables. The convention in PHP is to use all lowercase except when separating words (note the uppercase "N" in $userName.) This is a good convention to follow, and it's the one I use throughout this book.
- Watch your spelling! Every time you refer to a variable, PHP checks to see if that variable already exists somewhere in your program. If so, it uses that variable. If not, it quietly makes a new variable for you. If you misspell a variable name, PHP will not catch it. Instead, it will make a whole new variable, and your

program probably won't work correctly.

It isn't necessary to explicitly create a variable. When you refer to a variable, it is automatically created by PHP.

Assigning a Value to a Variable

The equals sign (=) is special in PHP. It does not mean "equals" (at least in the present context.) The equals sign is used for assignment. If you read the equals sign as the word "gets," you'll be closer to the meaning PHP uses for

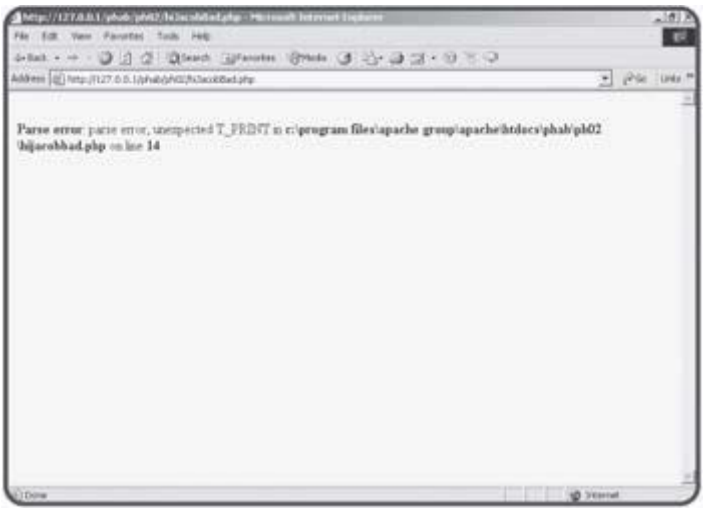


Figure 1.5 this is outputting new variables

this symbol. For example, the line

$userName = "Jacob"

If you wish to print the value of a variable, simply place th e variable name in the text you want printed. The line

print "Hi, $userName";

actually produces the output

Hi, Jacob

because when the server encounters the variable $userName, it replaces it with the

value of that variable, which is "Jacob." The output of the PHP program will be sent directly to the Web browser, so you can even include HTML tags in your output if you wish, simply by including them inside the quotes.

The ability to print the value of a variable inside other text is called string interpolation. That's not critical to know, but it could be useful information on a trivia show or something.

If you look back at the complete code for the hiJacob program, you can see that it has two lines of code inside the PHP block. Each line of PHP code ends with a semicolon. PHP is a more formal language than HTML and, like most programming languages, has some strict rules about the syntax used when writing a page.

Each unique instruction is expected to end with a semicolon. You'll end most lines of PHP code with a semicolon. If you forget to do this, you'll get an error that looks like Figure 1.6

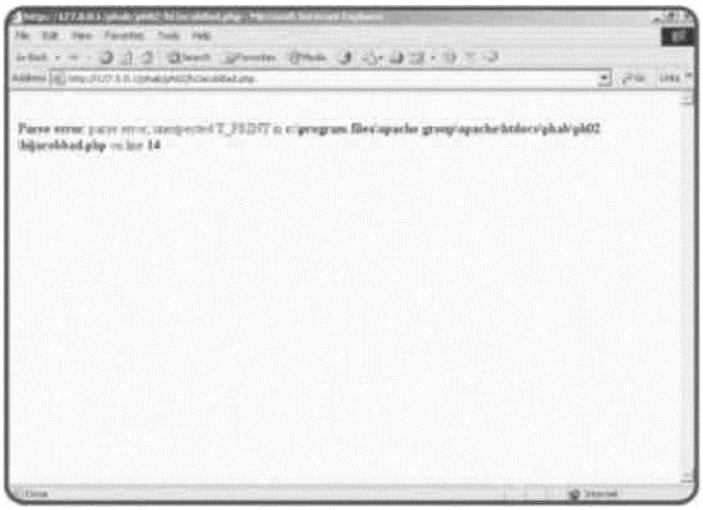


Figure 1.6: This error will occur if you forget to add a semicolon to the end of every line.

If you see this particular message, look back at your code to ensure you've

remembered to add a semicolon at the end of the line.

While the HiJacob program was interesting, there was no real advantage to using a variable. Now you will see another use of variables that shows how useful they can be.

Building the "Row Your Boat" Page

Figure 1.7 shows the "Row Your Boat" page.

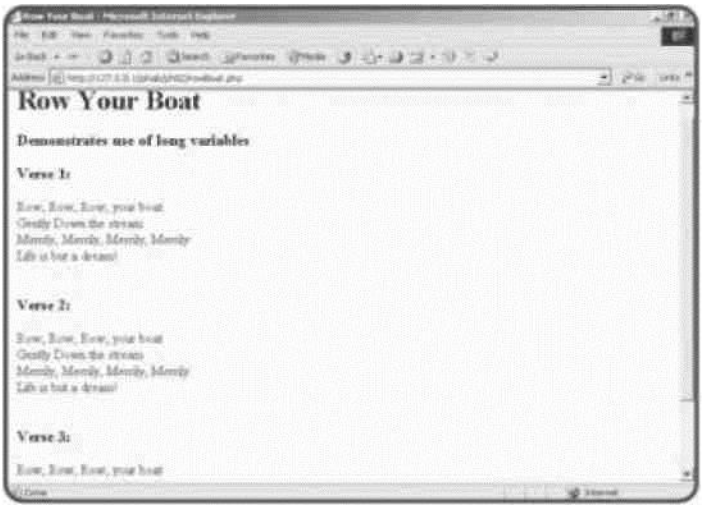


Figure 1.7: This program shows the words to a popular song. They sure repeat a lot.

I chose this song in particular because it repeats the same verse three times. If you look at the original code for the rowBoat.php program, you'll see I used a special trick to save some typing.

```
<html>
<head>
<title>Row Your Boat</title>
</head>
<body>
<h1>Row Your Boat</h1>
<h3>Demonstrates use of long variables</h3>
```

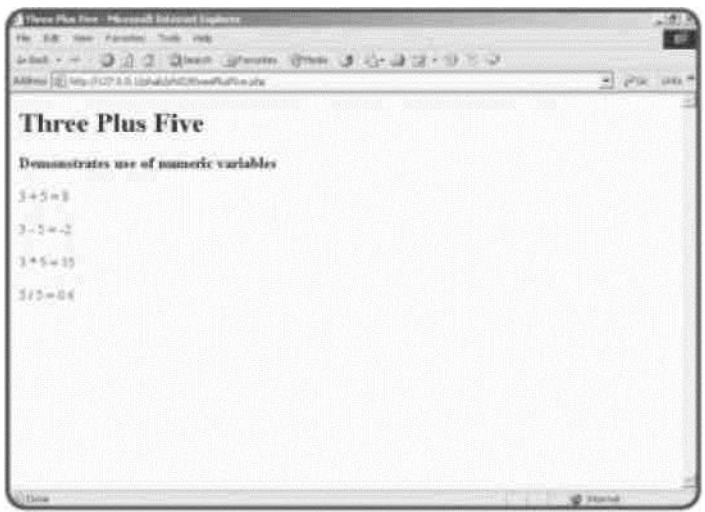
```
<?
$verse = <<<HERE
Row, Row, Row, your boat<br>
Gently down the stream<br>
Merrily, merrily, merrily, merrily<br>
Life is but a dream!<br>
<br><br>
HERE;
print "<h3>Verse 1:</h3>"; print $verse;
print "<h3>Verse 2:</h3>"; print $verse; print
"<h3>Verse 3:</h3>"; print $verse;
?>
</body> </html>
```

1.2 Working with Numeric Variables

Computers ultimately store information in on/off impulses. These very simple data values can be converted into a number of more convenient kinds of information. The PHP language makes most of this invisible to you, but it's still important to know that string (text) is handled differently in memory than numeric values, and there are two main types of numeric values.

Making the ThreePlusFive Program

As an example of how PHP works with numbers, consider the



ThreePlusFive.php program illustrated in Figure 1.8.

Figure 1.8: This program does basic math on variables containing the values 3 and 5.

All the work in the ThreePlusFive program is done with two variables called $x and $y. (I know, I recommended that you assign variables longer, descriptive names, but these variables are commonly used in arithmetic problems, so these very short variable names are OK in this instance.) The code for the program looks like this:

```
<html>
<head>
<title>Three Plus Five</title>
</head>
<body>
<h1>Three Plus Five</h1>
<h3>Demonstrates use of numeric variables</h3>
<?
$x = 3;
$y = 5;
print "$x + $y = "; print $x + $y; print
"<br><br>";
print "$x - $y = "; print $x - $y; print
"<br><br>";
print "$x * $y = "; print $x * $y; print
"<br><br>";
print "$x / $y = "; print $x / $y; print
"<br><br>";
?>
</body> </html>
```

Assigning Numeric Values

You create a numeric variable like any other variable in PHP. Simply assign a value to a variable, and the variable is created. Notice that numeric values do not require quotes. I created variables called $x and $y and assigned appropriate values to these variables.

Using Mathematical Operators

For each calculation, I wanted to print the problem as well as its solution. The line that says

print "$x + $y = ";

prints out the values of the $x and $y variables with the plus sign between them. In this particular case (since $x is 3 and $y is 5), it prints out the literal value

3 + 5 =

Because the plus sign and the equals sign are inside quotes, they are treated as ordinary text elements and PHP doesn't do any calculation (such as addition or assignment) with them.

The next line print $x + $y;

does not contain any quotes. It calculates the value of $x + $y and prints the result of this calculation (8) to the Web page.

Most of the math symbols you are familiar with also work with numeric variables. The plus sign (+) is used for addition, the minus sign (-) indicates subtraction, the asterisk (*) is used for multiplication, and the forward slash (/) is used for division. The remainder of the program illustrates how PHP does subtraction, multiplication, and division.

Creating a Form to Ask a Question

It's very typical for PHP programs to be made of two or more separate documents. An ordinary HTML page contains a form, which the user fills out. When the user presses the submit button, the information in all the form elements is sent to a program specified by a special attribute of the form. This program processes the information from the form and returns a result, which looks to the user like an ordinary Web page. To illustrate, look at the whatsName.html page illustrated in Figure 1.9.

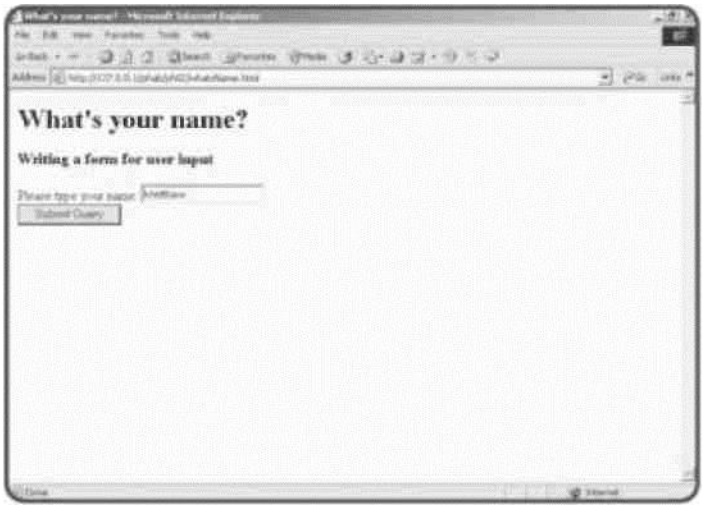


Figure 1.9: This is an ordinary HTML page containing a form.

The whatsName.html page does not contain any PHP at all. It's simply an HTML page with a form on it. When the user clicks on the Submit Query button, the page sends the value in the text area to a PHP program called hiUser.php.

Figure 1.10 shows what happens when the hiUser.php program runs:

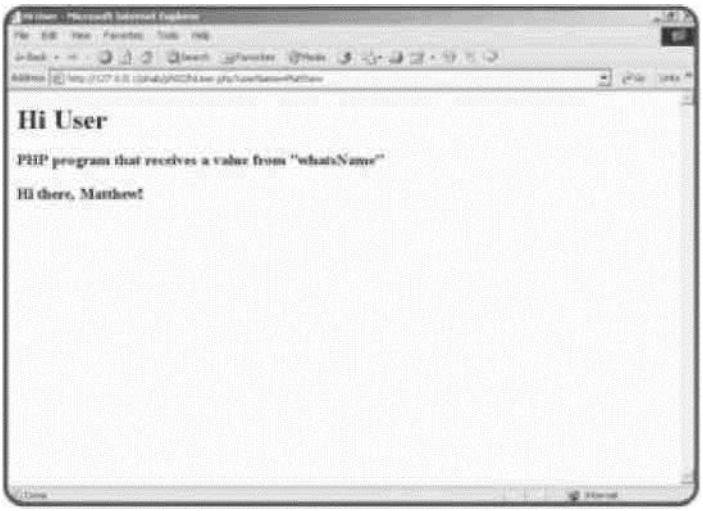


Figure 1.10: The resulting page uses the value from the original HTML form.

It's important to recognize that two different pages are involved in the transaction. In this section, you'll learn how to link an HTML page to a particular script, and how to write a script that expects certain form information.

Building an HTML Page with a Form

Forms are very useful when you want to get information from the user. To illustrate how this is done, look at the code for the whatsName.html file.

```
<html>
<head>
<title>What's your name?</title>
</head>
<body>
<h1>What's your name?</h1>
<h3>Writing a form for user input</h3>
<form method = "post"
      action = "hiUser.php"> Please type your name:
<input type = "text"
       name = "userName"
```

```
value = ""> <br> <input type =
"submit">
</form> </body> </html>
```

There is only one element of this page that may not be familiar to you. Take a careful look at the form tag. It contains two new attributes. The method attribute indicates how the data will be sent to the browser. There are two primary methods, get and post. The post method is the most powerful and flexible, so it is the one I use most often in this book. However, you'll see some interesting ways to use the get method later in this part in the section called "Sending Data without a Form."

Setting the Action Attribute to a Script File

The other attribute of the form tag is the action attribute. This is used to determine the URL of a program that will interpret the form. This attribute is used to connect a Web page to a program to read the page and respond with another page. The URL can be an absolute reference (which begins with http:// and contains the entire domain name of the response program), or they can be relative references (meaning the program will be in the same directory as the original Web page).

The whatsName.html page contains a form with its action attribute set to hiUser.php. Whenever the user clicks on the submit button, the values of all the fields (there's only one in this case) will be packed up and sent to a program called hiUser.php, which is expected to be in the same directory as the original whatsName.html page.

1.3 Writing a Script to Retrieve the Data

The code for hiUser.php is specially built. The form that called the hiUser.php code is expected to have an element called userName. Take a look at the code for hiUser.php and you'll see what I mean.

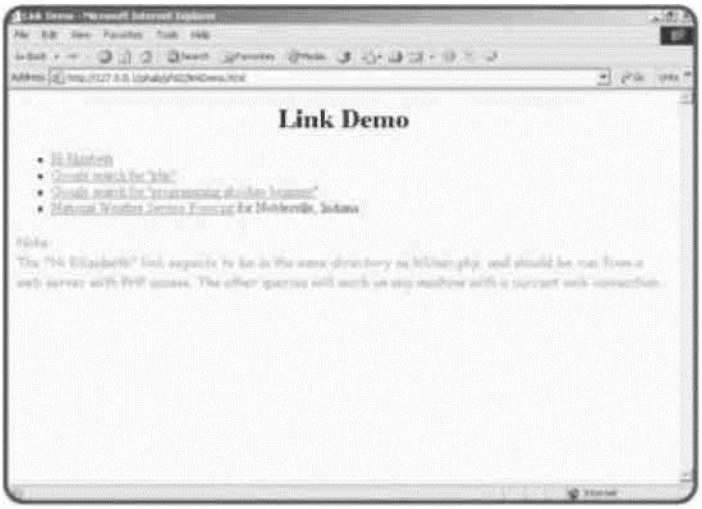
```
<html>
<head>
<title>Hi User</title>
</head>
<body>
<h1>Hi User</h1>
<h3>PHP program that receives a value from "whatsName"</h3>
<?
   print "<h3>Hi there, $userName!</h3>";
?>

</body> </html>
```

Like many PHP pages, hiUser.php is mainly HTML. The only thing that's different is the one print statement. This statement incorporates the variable $userName. The puzzling thing is there is no other mention of the variable anywhere in the code.

When a user submits a form to a PHP program, the PHP processor automatically creates a variable with the same name as every form element on the original HTML page. Since the whatsName.html page has a form element called userName, any PHP program that whatsName.html activates will automatically have access to a variable called $userName. The value of that variable will be whatever the user has entered into the field before pressing the Submit button.

Sometimes it can be very handy to send data to a server-side program without necessarily using a form. This is a little-known trick that can really



enhance

Figure 1.11 Sending data with link

your Web pages without requiring a lick of PHP programming. The Link Demo page (linkDemo.html) shown in illustrate this phenomenon.

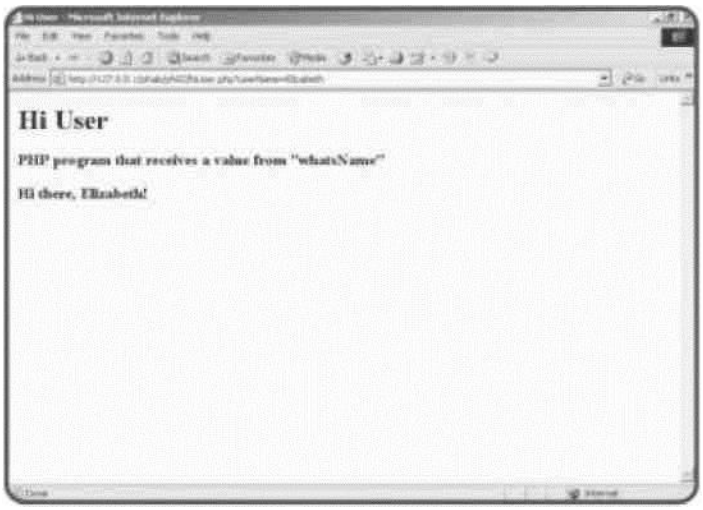


Figure 1.12: The links on this page appear ordinary, but they are unusually powerful.

When I clicked on the "Hi Elizabeth" link, I was taken to the HiUser program with the value "Elizabeth" automatically sent to the program!

Understanding the get Method

All the links in the linkDemo.html page use a similar trick. As you recall from earlier in the part, form data can be sent to a program through two different methods. The post method is the technique you'll usually use in your forms, but you've actually been using the get method all along, because normal HTML requests actually are get requests. The interesting thing about that is that you can send form data to any program that knows how to read get requests by embedding the request in your URL. As an experiment, switch the method attribute of whatsName.html so the form looks like this:

```
<form method = "get"
    action = "hiUser.php">
```

Then run the page again. It will work the same as before, but the URL of the resulting page will look like this (presuming you said the user's name is "Andy"):

http://127.0.0.1/phab/ph02/hiUser.php?userName=Andy

The get method stashes all the form information into the URL using a special code. If you go back to the whatsName page and put in "Andy Harris," you'll get a slightly different result:

http://127.0.0.1/phab/ph02/hiUser.php?
userName=Andy+Harris

The space between "Andy" and "Harris" was converted to a plus sign because space characters cause a lot of confusion. When form data is transmitted, it often undergoes a number of similar transformations. In PHP programming, all the translation is automatic, so you don't have to worry about it.

Using a URL to Embed Form Data

If you understand how this works, you can use a similar technique to harness any server-side program on the Internet. (Presuming it's set up to take get-method data—some are not.) When I examined the URLs of Google searches, I could see

my search data in a field named "q" (for query, I suppose). I took a gamble that all the other fields would have default values, and wrote a hyperlink that incorporates a query. My link looked like this:

 Google search for "php"

Whenever the user clicks on this link, it sets up a get-method query to google's search program. The result is a nifty Google search. One fun thing you might want to do is figure out how to set up "canned" versions of your most common queries in various search engines so you can get updated results with one click. illustrates what happens when the user clicks on the "google php" link in the linkDemo page.

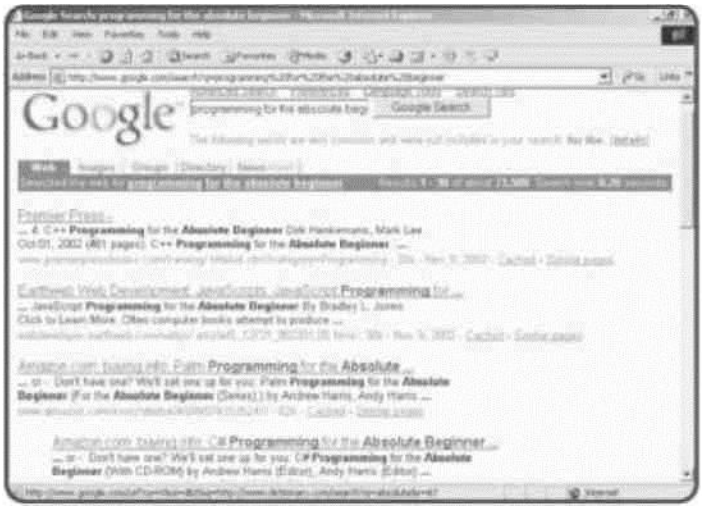


Figure 1.13 shows the results of this slightly more complex search.

The Google search for "Absolute Beginners Programming"
shows some really intriguing book offerings!

<a href =
"http://www.google.com/search?q=programming for the absolute beginner">
Google search for "programming absolute beginner"

Reading Input from Other Form Elements

A PHP program can read the input from any type of HTML form

element. In all cases, the name attribute of the HTML form object becomes a variable name in PHP. In general, the value of the PHP variable comes from the value property of the form object.

Introducing the borderMaker program

To examine most of the various form elements, I built a simple page to demonstrate various attributes of Cascading Style Sheet borders.

The HTML program is shown in Figure 1.14.

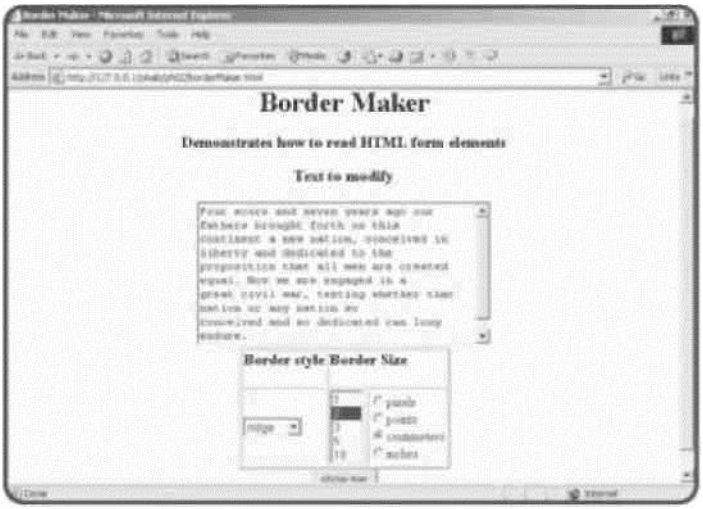


Figure 1.14: The borderMaker HTML page uses a text area, two list boxes, and a select group.

Building the borderMaker.html Page

The borderMaker.html page contains a very typical form with most of the major input elements in it. The code for this form is

```
<html>
<head>
<title>Font Choices</title>
</head>
<body>
<center>
```

<h1>Font Choices</h1>
<h3>Demonstrates how to read HTML form elements</h3>
<form method = "post"action = "borderMaker.php">
<h3>Text to modify</h3>
<textarea name = "basicText"
rows = "10" cols = "40">
Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation or any nation so conceived and so dedicated can long endure.
</textarea>
<table border = 2> <tr>
<td><h3>Border style</h3></td>
<td colspan = 2><h3>Border Size</h3></td> </tr>
<tr>
<td>
<select name = borderStyle>
<option value = "ridge">ridge</option> <option value = "groove">groove</option> <option value = "double">double</option> <option value = "inset">inset</option> <option value = "outset">outset</option>
</select>
</td>
<td>
<select size = 5
name = borderSize>
<option value = "1">1</option>
<option value = "2">2</option>
<option value = "3">3</option>
<option value = "5">5</option>
<option value = "10">10</option> </select> </td>

```
<td>
<input type = "radio"
name = "sizeType"
value = "px">pixels<br> <input type = "radio"
name = "sizeType"
value = "pt">points<br> <input type = "radio"
name = "sizeType"
value = "cm">centimeters<br> <input type = "radio"
name = "sizeType"
value = "in">inches<br> </td> </tr> </table>
<input type = "submit"
value = "show me">
</form>
</center>
</body>
</html>
```

The borderMaker.html page is designed to interact with a PHP program called borderMaker.php, as you can see by inspection of the action attribute. Note that I added a value attribute for each option element, and the radio buttons all have the same name but different values. The value attribute becomes very important when your program is destined to be read by a program, as you shall see very shortly. Finally, the Submit button is critical, because nothing interesting will happen until the user submits the form.

Reading the Form Elements

The borderMaker.php program expects input from borderMaker.html. When the user submits the HTML form, the PHP program produces results like those shown in Figure 1.15.

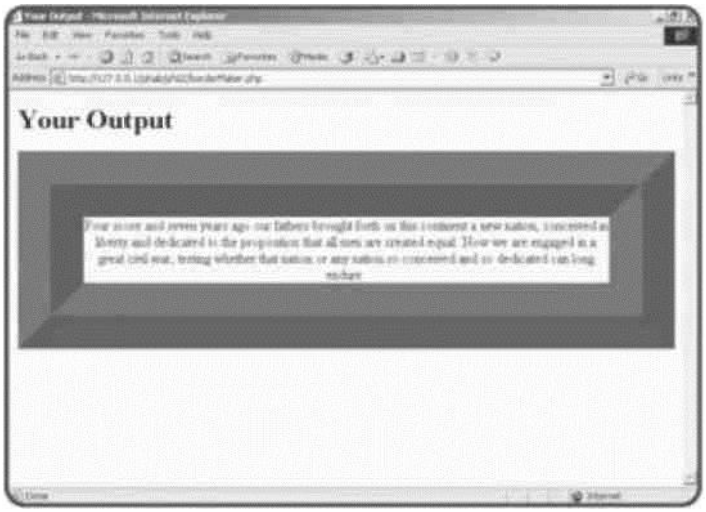


Figure 1.15: The borderMaker.php code reacts to all the various input elements on the form.

In general, it doesn't matter what type of element is used on an HTML form. The PHP interpreter simply looks at the name of each element and the value. By the time the information gets to the server, it doesn't really matter what type of input element was used. PHP automatically creates a variable corresponding to each form element. The value of that variable will be the value of the element. The code used in borderMaker.php illustrates:

```
<html>
<head>
<title>Your Output</title>
</head>
<body>
<h1>Your Output</h1>
<center>
<? $theStyle = <<<HERE
```

```
"border-width:$borderSize$sizeType;
border-style:$borderStyle;
border-color:green"
HERE;
print "<div style = $theStyle>"; print $basicText; print
"</span>";
?> </center>
</body> </html>
```

In the case of text boxes and text areas, the user types the value directly in. In borderMaker.html, there is a text area called basicText. The PHP interpreter creates a variable called $basicText. Anything typed into that text box (as a default the first few lines of the Gettysburg Address) becomes the value of the $basicText variable.

Reading Select Elements

Recall that both drop-down lists and list boxes are created with the select object. That object has a name attribute. Each of the possible choices in the list box is an option object. Each option object has a value attribute.

The name of the select object will become the name of the variable. For example, borderMaker.html has two select objects, borderSize and borderStyle. The PHP program can expect to find two corresponding variables, $borderSize and $borderStyle. Because there is no place for the user to type a value into a select object, the values it can return must be encoded into the structure of the form itself. The value of whichever option the user selected will be sent to the PHP program as the value of the corresponding variable. For example, if the user chose groove as the border style, the $borderStyle variable will have the value groove in it.

CSS allows the developer to indicate sizes with a variety of measurements. This is an ideal place for a group of radio buttons because only

one unit of measure is appropriate at a time. Even though there are four different radio
buttons on the borderDemo.html page with the name sizeType, the PHP program will only see one $sizeType variable. The value associated with whichever option is selected will become the value of the $sizeType variable. Note that like option elements, it is possible for the value of a radio button to be different than the text displayed beside it.

Building the HTML Page

With the basic outline from , it becomes clear how the story program should be created. It should have two parts. The first is an HTML page that prompts the user for all the various words. Here's the code for my version:

```
<html>
<head>
<title>Story</title>
</head>
<body>
<h1>Story</h1>
<h3>Please fill in the blanks below, and I'll tell
      you a story</h3> <form method = "post"
         action = "story.php">
<table border = 1> <tr>
   <th>Color:</th> <th>
<input type = "text" name = "color" value = "">
</th> </tr>
<tr>
   <th>Musical Instrument</th> <th>
      <input type = "text"
name = "instrument" value = ""> </th> </tr>
<tr>
```

<th>Animal</th> <th>
<input type = "text" name = "anim1" value = "">
</th> </tr>
<tr>
<th>Another animal</th> <th>
<input type = "text" name = "anim2" value = "">
</th> </tr>
<tr>
<th>Yet another animal!</th> <th>
<input type = "text" name = "anim3" value = ""> </th> </tr>
<tr>
<th>Place</th> <th>
<input type = "text" name = "place" value = ""> </th> </tr>
<tr>
<th>Vegetable</th> <th>
<input type = "text"
name = "vegetable" value = ""> </th> </tr>
<tr>
<th>A structure</th> <th>
<input type = "text"
name = "structure" value = ""> </th> </tr>
<tr>
<th>An action</th> <th>
<select name = "action">
<option value = "fast asleep">fast asleep</option>
<option value = "drinking cappuccino">drinking cappuccino</option> <option value = "wandering around aimlessly">wandering around aimlessly</opti <option value = "doing nothing in particular">doing nothing in particular</op </select> </th> </tr>
<tr>

```
<td colspan = 2> <center>
<input type = "submit"
value = "tell me the story"> </center> </td> </tr> </table>
</form> </body> </html>
```

There's nothing terribly exciting about the HTML. In fact, since I had the plan, I knew exactly what kinds of things I was asking for and created form elements to ask each question. I used a list box for the last question so I could put in some interesting suggestions. Note that I changed the order a little bit just to throw the user off.

There are a few things to check when you're writing a page that will connect to a script. First, ensure you've got the correct action attribute in the form tag. (for that matter, make sure you've added an action attribute.) Make sure each form element has an appropriate name attribute. If you have radio or option objects, make sure each one has an appropriate value. Finally, be sure there is a Submit button somewhere in your form.

Checking the Form

I actually wrote two different scripts to read this form. The first one I wrote simply checked each element to make sure it received the value I expected. Here's the first program, called storySimple.php.

```
<html>
<head>
<title>Little Boy Who?</title>
</head>
<body>
<h1>Little Boy Who?</h1>
<h3>Values from the story page</h3>
<table border = 1> <tr>
  <th>Variable</th>
```

```
   <th>Value</th> </tr>
<tr>
   <th>color</th>


   <td><? print $color ?></td> </tr>
<tr>
   <th>instrument</th>
   <td><? print $instrument ?></td> </tr>
<tr>
   <th>anim1</th>
   <td><?  print  $anim1 ?></td> </tr>
<tr>
   <th>anim2</th>
   <td><?  print  $anim2 ?></td> </tr>
<tr>
   <th>anim3</th>
   <td><? print $anim3 ?></td> </tr>
<tr>
   <th>place</th>
   <td><? print $place ?></td> </tr>
<tr>
   <th>vegetable</th>
   <td><? print $vegetable ?></td> </tr>
<tr>
   <th>structure</th>
   <td><? print $structure ?></td> </tr>
<tr>
```

```
  <th>action</th>
  <td><?  print  $action ?></td></tr>
</table>
<form>
</html>
```

Building the Final StoryThe story itself is very simple to build if you've made a plan and ensured that the variables are working right. All I had to do was write out the story as it was written in the plan, with the variables incorporated in the appropriate places. Here's the code for the finished story.php page:

```
<html>
<head>
<title>Little Boy Who?</title>
</head>
<body>
<center>
<h1>Little Boy Who?</h1>
<?
print <<<HERE
<h3>
Little Boy $color, come blow your $instrument!<br>
The $anim1's in the $place, the $anim2's in the $vegetable.<br>
Where's the boy that looks after the $anim3?<br>
He's under the $structure, $action.
</h3>
HERE;
?>
</center>
</body> </html>
```

It might astonish you that the final program is quite a bit simpler than the test

program. Neither is very complicated, but once you have created the story, set up the variables, and tested that all the variables are being sent correctly, the story program itself turns out to be almost trivial. Most of the story.php code is plain HTML. The only part that's in PHP is one long print statement. This uses the print <<<HERE syntax to print out a long line of HTML text with PHP variables embedded inside. The story itself is this text..

Creating a Condition

On the surface, the behavior of the Ace program is very straightforward: it does something interesting only if the die roll is one, and it doesn't do that interesting thing in any other case. While it is a simple idea, the implications are profound.

The same simple mechanism used in the Ace program is the foundation of all complicated computer behavior, from flight simulators to heart monitors. Take a look at the code for the Ace program and see if you can spot the new element:

```
<html>
<head>
<title>Ace!</title>
</head>
<body>
<h1>Ace!</h1>
<h3>Demonstrates if statement</h3>

<?
$roll = rand(1,6);
print "You rolled a $roll";

if ($roll == 1){
   print "<h1>That's an ace!!!!!</h1>"; } // end if

print "<br>";
print "<img src = die$roll.jpg>";
?>
```


Refresh this page

</body> </html>

The secret to this program is the segment that looks like this:

```
if ($roll == 1){
   print "<h1>That's an ace!!!!!</h1>"; } // end if
```

The line that prints "That's an ace!" will not happen every time the program is run. It will only happen if a certain condition is true. The if statement sets up a condition for evaluation. In this case, the condition is read "$roll is equal to one." If that condition is true, all the code between the left brace ({) and the right brace (}) will evaluate. If the condition is not true, the code between the braces will be skipped altogether.

A condition can be thought of as an expression that can be evaluated as true or false. Any expression that can return a true or false value can be used as a condition. Most conditions look much like the one in the Ace program. This condition checks the variable $roll to see if it is equal to the value 1.

Note that equality is indicated by two equals signs (==).

This is important, because computer programs are not nearly as flexible as humans. We humans often use the same symbol for different purposes. While computer languages can do this, it often leads to problems. The single
equals sign is reserved for assignment. You should read this line

```
$x = 5;
```

as x gets five, indicating that the value five is being assigned to the variable $x. The code fragment

```
$x == 5;
```

should be read as x is equal to five, as it is testing equality. It is essentially asking whether x is equal to five. A condition such as $x == 5 does not stand on its own. Instead, it is used inside some sort of other structure, such as an if

statement.

TABLE COMPARISON OPERATORS

Equality (==) is not the only type of comparison PHP allows. There are several other ways to compare a variable and a value or two variables. These comparison operators are described in table 3.1

Operator	**Description**
==	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
!=	not equal to

These comparison operators work on any type of data, although the results might be a little strange. For example, if you have a condition like

"a" < "b"

you would get the result true because alphabetically, the letter "a" is earlier than "b," so it has a "smaller" value.

In other words, the program does one thing when the condition is true and something else when the condition is false.

Using the else Clause

The code for the aceOrNot program shows how the else clause can be used to allow for multiple behavior.

```
<html>
<head>
<title>Ace or Not</title>
</head>
<body>
<h1>Ace or Not</h1>
```

```
<h3>Demonstrates if statement with else clause</h3>

<?
$roll = rand(1,6);
print "You rolled a $roll";
print "<br>";

if ($roll == 1){          an   ace!!!!!</h1>";
   print    "<h1>That's an ace...";                }
else {
   print "That's not } // end if

print "<br>";
print "<img src = die$roll.jpg>";
?>
<br>
Refresh this page in the browser to roll another die.

</body> </html>
```

The interesting part of this code comes near the if statement:

```
if ($roll == 1){
   print "<h1>That's an ace!!!!!</h1>"; } else {
   print "That's not an ace..."; } // end if
```

If the condition $roll == 1 is true, the program prints "That's an ace!!!!!" If the condition is not true, the code between else and the end of the if structure is executed instead. Notice the structure and indentation. One chunk of code (between the condition and the else statement, encased in braces) occurs if the condition is true. If the condition is false, the code between else and the end of the if structure (also in braces) is executed. You can put as much code in either segment as you wish. Only one of the segments will run (based on the condition), but you are guaranteed that one will execute.

`

There appear to be some new PHP functions. it called the verse1() function, then the chorus() function, and so on. There are some new functions, but they weren't shipped with PHP. Instead, I made them as part of the page. You can take a set of instructions and store them with a name. This essentially builds a new temporary command in PHP, so you can combine simple commands to do complex things. Building a function is simple. Use the keyword function followed by the function's name and a set of parentheses. Keep the parentheses empty for now.

Use a pair of braces ({}) to combine a
series of code lines into one function. Don't forget the right brace (}) to end the function definition. It's smart to indent everything between the beginning and end of a function.

The chorus() function is especially handy in this program because it can be re-used. It isn't necessary to re-write the code for the chorus each time when you can simply call a function instead.

1.4 Creating New Functions

Functions are meant to be self-contained. This is good because the entire program can be too complex to understand. If you break the complex program into smaller functions, each function can be set up to work independently. When you work inside a function, you don't have to worry about anything outside the function. If you create a variable inside a function, that variable dies as soon as you leave the function. This prevents many errors that can otherwise creep into your code. The bad side of functions being so self-contained is you often want them to work with data from the outside program. There are a couple of ways to do this. You can send a parameter to a function, which allows you to determine one or more values sent to the function as it starts. Each function can also have a return value. An example will make this more clear. The param program, shown in Figure 1.17 illustrates another form of the This Old Man song. Although again the user might not be aware of it, there are some important differences between this more sophisticated program and the first This Old Man program.

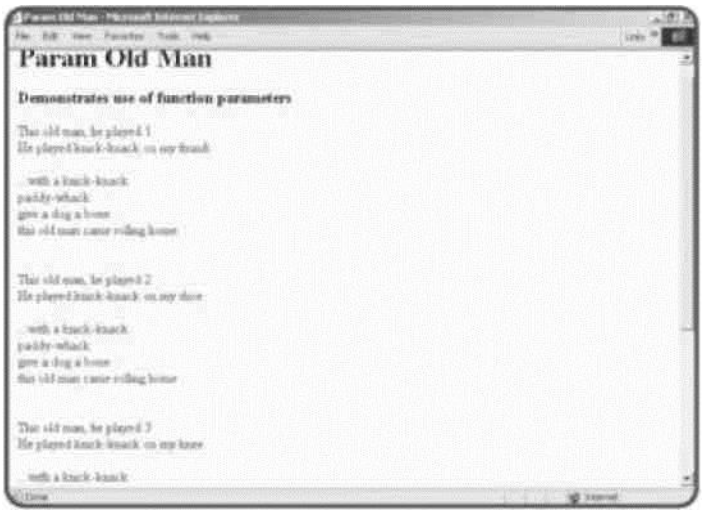


Figure 1.18: While the output looks similar to Figure 1.17, the program that produced this page is much more efficient.

Notice that the output of Figure 1.22 is longer than that of 1.21, but the code that generates this longer output is shorter and more efficient.

```
<html>
<head>
<title>Param Old Man</title>
```

```
</head>
<body>
<h1>Param Old Man </h1>
<h3>Demonstrates use of function parameters</h3>
<?
print verse(1);
print chorus();
print verse(2);
print chorus();
print verse(3);
print chorus();
print verse(4);
function verse($stanza){ switch ($stanza){ case 1:
$place = "thumb";
break; case 2:
$place = "shoe";
break; case 3:
$place = "knee";
break; case 4:
$place = "door";
break; default:
$place = "I don't know where"; } // end switch
$output = <<<HERE
This old man, he played $stanza<br>
He played knick-knack on my $place<br><br> HERE;
return $output; } // end verse
function chorus(){
$output = <<<HERE
...with a knick-knack<br>
```

```
paddy-whack<br>
give a dog a bone<br>
this old man came rolling home<br>
<br><br> HERE;
return $output; } // end chorus
?>
</body>
</html>
```

Looking at Encapsulation in the Main Code Body

This code features a number of improvements over the previous version. First, look at the main body of the code that looks like this:

```
print verse(1); print chorus(); print verse(2); print chorus(); print verse(3); print chorus(); print verse(4); print chorus();
```

The main code body is very easy to understand. The program is to print the first verse, then the chorus, then the second verse, then the chorus, and so on. The details of how all these things are to be generated is left to the individual functions. This is an example of encapsulation. Encapsulation is good, because it allows you to think about problems in multiple levels. At the highest level, you're interested in the main ideas (print the verses and chorus) but you're not so concerned about the exact details. You use the same technique when you talk about your day: "I drove to work, had some meetings, went to lunch, and taught a class." You don't usually describe each detail of each task. Each major task can be broken down into its component tasks later. (If somebody asks, you could really describe the meeting: "I got some coffee, appeared to be taking notes furiously on my palm pilot, got a new high score on Solitaire while appearing to take notes, scribbled on the agenda, and dozed off during a presentation.")

Returning a Value: The chorus() Function

Another interesting thing about the main section of code is the use of the print() function. In the last program, I simply said chorus() and the program printed the

verse. In this program, I did it a little differently. The chorus() function doesn't actually print anything to the screen. Instead, it creates the chorus as a big string and sends that value back to the program, which can do whatever it wants with it. This behavior isn't really new to you. Think about the rand() function. It always returns a value back to the program. The functions in this program work in the same way. Take another look at the chorus() function to see what I mean.

```
function chorus(){
  $output = <<<HERE
  ...with a knick-knack<br>
  paddy-whack<br>
  give a dog a bone<br>
  this old man came rolling home<br>
  <br><br> HERE;
  return $output; } // end chorus
```

I began the function by creating a new variable called $output. You can create variables inside functions by mentioning them, just like you can in the main part of the program. However, a variable created inside a function loses its meaning as soon as the function is finished. This is good, because it means the variables inside a function belong only to that function. You don't have to worry about whether the variable already exists somewhere else in your program. You also don't have to worry about all the various things that can go wrong if you mistakenly modify an existing variable. I assigned a long string (the actual chorus of the song) to the $output variable with the <<<HERE construct.

The last line of the function uses the return statement to send the value of $output back to the program. Any function can end with a return statement. Whatever value follows the keyword return will be passed to the program. This is one way your functions can communicate to the main program.

Accepting a Parameter in the verse() Function

The most efficient part of this newer program is the verse() function. Rather

than having a different function for each verse, I wrote one function that can work for all the verses. After careful analysis of the song, I noticed that each verse is remarkably similar to the others. The only thing that differentiates each verse is what the old man played (which is always the verse number) and where he played it (which is something rhyming with the verse number). If there is some way to indicate which verse to play, it should be easy enough to produce the correct verse. Notice that when the main body calls the verse

function, it always indicates a verse number in parentheses. For example, it makes a reference to verse(1) and verse (3). These commands both call the verse function, but they send different values (1 and 3) to the function. Take another look at the code for the verse () function to see how the function responds to these inputs.

```
function verse($stanza){ switch ($stanza){ case 1:
$place = "thumb";
break; case 2:
$place = "shoe";
break; case 3:
$place = "knee";
break; case 4:
$place = "door";
break; default:
$place = "I don't know where"; } // end switch
$output = <<<HERE
This old man, he played $stanza<br>
He played knick-knack on my $place<br><br> HERE;
return $output; } // end verse
```

```
This old man, he played $stanza<br>
He played knick-knack on my $place<br><br> HERE;
return $output; } // end verse
```

In this function, I indicated $stanza as a parameter in the function definition.

A parameter is simply a variable associated with the function. If you create a function with a parameter, you are required to supply some sort of value whenever you call the function. The parameter variable automatically receives the value from the main body. For example, if the program says verse(1), the verse function will be called, and the $stanza variable will contain the value 1.

I then used a switch statement to populate the $place variable based on the value of $stanza. Finally, I created the $output variable using the $stanza and $place variables and returned the value of $output.

You have learned some ways to have your main program share variable information with your functions. In addition to parameter passing, sometimes you'll want your functions to have access to variables created in the main program. This is especially true because all the variables automatically created by PHP (such as those coming from forms) will be generated at the main level. You must tell PHP when you want a function to use a variable that has been created at the main level.

Take a look at the code for the Scope Demo, and you'll see how it works.

```
<html>
<head>
<title>Scope Demo</title>
</head>
<body>
<h1>Scope Demo</h1>
<h3>Demonstrates variable scope</h3>
<?
$a = "I have a value"; $b = "I have a value";
print <<<HERE
   outside the function, <br> \$a is "$a", and<br>
   \$b is "$b"<br><br> HERE;
myFunction();
function myFunction(){
```

```
  //make $a global, but not $b global $a;
  print <<<HERE
    inside the function, <br>
    \$a is "$a", and<br>
    \$b is "$b"<br><br> HERE; } // end
myFunction
?>
</body> </html>
```

For this demonstration, I created two variables, called $a and $b. I gave them both the value "I have a value." As a test, I printed out the values for both $a and $b.

At the beginning of this part, I showed you the Petals Around the Rose game. This game uses all the skills you have learned so far, including the new concepts you learned in this part. If you haven't already done so, play the game now so you can see how it works, because it won't be as much fun once you know the secret.

Here's the basic plan of the Petals game: Each time the page is drawn, it randomly generates five dice, and calculates the correct number of petals based on a super-secret formula. The page includes a form that has a text area called guess for the user to guess the right answer. The form also includes a hidden field called numPetals, which tells the program what the correct answer was. Starting HTML

Like most PHP programs, the Petals game uses some HTML to set everything up. The HTML is pretty basic because most of the interesting HTML will be created by the PHP code.

```
<HTML>
<head>
<title>Petals Around the Rose</title>
</head>
<body bgcolor = "tan">
<center>
```

<font face = "Comic Sans MS">

<h1>Petals Around the Rose</h1>

I decided on a tan background with a whimsical font. This should give the program a light feel.

Main Body Code

The main PHP code segment has three main jobs to do. These jobs are (appropriately enough) stored in three different functions. One goal of encapsulation is to make the main code body as clean as possible. This goal has been achieved in the Petals game.

```
<?
printGreeting();
printDice();
printForm();
```

All the real work is passed off to the various functions, which will each be described shortly. Even before you see the functions themselves, you have a good idea what each function will do, and you also have a good sense of the overall flow of the program. If you encapsulate your code and name your functions well, it makes your code much easier to read and repair.

The printGreeting() Function

The purpose of this function is to print a greeting to the user. There are three possible greetings. If the user has never called this program before, the program should provide a welcome. If the user has been here before, he or she has guessed the number of petals. That guess might be correct (in which case a congratulatory message is appropriate) or incorrect, requiring information about what the correct answer was. The printGreeting() function uses a switch statement to handle the various options.

```
function printGreeting(){ global
  $guess, $numPetals; if
  (empty($guess)){
    print "<h3>Welcome to Petals Around the Rose</h3>"; }
  else if ($guess == $numPetals){
    print "<h3>You Got It!</h3>"; } else {
    print <<<HERE
      <h3>from last try: </h3> you
      guessed: $guess<br><br>
      -and the correct answer was: $numPetals petals around the rose<br> HERE;
  } // end if
} // end printGreeting
```

This function refers to both the $guess and $numPetals variables, which are both automatically created. You can use one global statement to make more than one variable global by separating the variables with commas.

The $guess variable will be empty if this is the first time the user has come to the program. If $guess is empty, I print a welcoming greeting. If $guess is equal to $numPetals, the user has guessed correctly, so I print an appropriate congratulations. If neither of these conditions is true (which will be most of the time), the function will print out a slightly more complex string indicating the user's last guess and the correct answer. This should give the user enough information to finally solve the riddle.

The else if structure turned out to be the easiest option here for handling the three possible conditions I wanted to check.

The printDice() Function

After the program prints out a greeting, it does the important business of generating

the random dice. It's relatively easy to generate random dice, as you saw from earlier in this part. However, I also wanted to be efficient and calculate the correct number of petals. To make the printDice() function more efficient, you'll see that it calls some other custom functions.

```
function printDice(){ global $numPetals;
print "<h3>New Roll:</h3>"; $numPetals = 0;
$die1 = rand(1,6);
$die2 = rand(1,6);
$die3 = rand(1,6);
$die4 = rand(1,6);
$die5 = rand(1,6);
showDie($die1);
showDie($die2);
showDie($die3);
showDie($die4);
showDie($die5);
   print "<br>";
   calcNumPetals($die1);
   calcNumPetals($die2);
   calcNumPetals($die3);
   calcNumPetals($die4);
   calcNumPetals($die5);
} // end printDice
```

The printDice() function is very concerned with the $numPetals variable, but doesn't need access to $guess. It requests access to $numPetals from the main program. After printing out the "New Roll" message, it resets $numPetals to zero. The value of $numPetals will be recalculated each time the dice are rolled.

I got new dice values by calling the rand(1, 6) function six times. I stored each result

in a different variable, named $die1 to $die6. To print out an appropriate graphic for each die, I called the showDie() function (which will be described next). I printed out a line break, then called the calcNumPetals() function (which will also be described soon) once for each die.

The showDie() Function

The showDie() function is used to simplify repetitive code. It accepts a die value as a parameter, and generates the appropriate HTML code for drawing a die with the corresponding number of dots.

```
function showDie($value){ print <<<HERE
<img src = "die$value.jpg" height = 100 width = 100>
} // end showDie
```

The calcNumPetals Function

The printDice() function also calls calcNumPetals() once for each die. This function receives a die value as a parameter. It also references the $numPetals global variable. The function uses a switch statement to determine how much to add to $numPetals based on the current die's value.

Here's the trick. The center dot of the die is the rose. Any dots around the center dot are the petals. The value one has a rose but no petals. 2, 4, and 6 have petals, but no rose. 3 has two petals, and 5 has four. If the die roll is 3, $numPetals should be increased by 2, and if the roll is 5, $numPetals should be increased by 4. function calcNumPetals($value){

```
  global $numPetals; switch ($value) { case
  3:
    $numPetals += 2; break; case 5:
$numPetals += 4; break; } // end switch
} // end calcNumPetals
```

The += code is a shorthand notation. The line $numPetals +=

2;

is exactly equivalent to $numPetals = $numPetals + 2;

The first style is much shorter and easier to type, so it's the form most programmers prefer.

The printForm() Function

The purpose of the printForm() function is to print out the form at the bottom of the HTML page. This form is pretty straightforward except for the need to place the hidden field for $numPetals.

```
function printForm(){ global $numPetals;
   print <<<HERE
   <h3>How many petals around the rose?</h3>
   <form method = "post">
<input type = "text" name = "guess" value = "0">
 <input type = "hidden" name = "numPetals" value = "$numPetals"> <br>
   <input type = "submit"> </form> <br>
   <a href = "petalHelp.html" target = "helpPage">
   give me a hint</a> HERE;
} // end printForm
```

This code places the form on the page. I could have done most of the form in plain HTML without needing PHP for anything but the hidden field, but when I start using PHP, I like to have much of my code in PHP. It helps me see the flow of things more clearly (print greeting, print dice, print form).

The Ending HTML Code

The final set of HTML code closes everything up. It completes the PHP segment, the font, the centered text, the body, and finally the HTML itself.

```
?>
</font>
```

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

Create MySQL queries from PHP

Data is such an important part of modern programming that entire programming languages are devoted to manipulating databases. The primary standard for database languages is Structured Query Language (SQL). SQL is a standardized language for creating databases, storing information into databases and retrieving this information. Special applications and programming environments specialize in interpreting SQL data and acting on it.

Often a programmer will begin by creating a data structure in SQL, then will write a program in some other language (such as PHP) to allow access to that data. The PHP program can then formulate requests or updates to the data, which are passed on to the SQL interpreter. This approach has a couple of advantages. First, once you learn SQL, you can apply it easily to a new programming language. Also, you can easily add multiple interfaces to an existing data set because many programming languages have ways to access an SQL interpreter. Many relational database management systems are available, but the MySQL environment is especially well suited to working with PHP. However, the basic concepts of SQL remain the same no matter what type of database you are working on. Most of the SQL commands described in this part work without modification in Microsoft Access, Microsoft SQL Server, and Oracle, as well as a number of other RDBMS packages.

I'll begin this part by explaining how to create a simple database in MySQL. There are a number of ways to work with this package, but I'll start by showing you how to write a script that builds a database in a text file. I'll use the SQL language, which is different in syntax and style from PHP. Then I'll show you SQLyog, a wonderful front-end package for working with MySQL databases. In Part 8, "Connecting to Database Within PHP," I'll show you how to contact and manipulate your MySQL database from within PHP.

Build basic databases

Databases are described by a very specific organization scheme. To illustrate database concepts, I will create and view a simple phone list. The basic structure of the phone list is shown in Table 7.1.

Table 7.1: Phone list summary

id	**firstName**	**lastName**	**e-mail**	**phone**
0	Andy	Harris	<aharris@cs.iupui.ed	123-4567
1	Joe	Slow	<jslow@myPlace.net	987-6543

The phone list shows a very typical simple data table. Database people like to give special names to the parts of the database, so I'll use this simple phone list to illustrate. Each row of the table is called a record. Records describe discrete entities. The list of records is called a table. Each record in a table has the same elements, which are called fields, (or sometimes simply columns). Every record in the table has the same field definitions, but records can have different values in the fields. The fields in a table are defined in specific ways. Because of the way database tables are stored in files, the computer must always know how much room to allocate for each field, so the size and type of each field is important. This particular database is defined with five fields. The id field is an integer. All the other fields contain string data.

Creating a Table

RDBMS programs use a special language called Structured Query Language(SQL) to create and manipulate databases. SQL is usually pretty easy to understand, compared to full-blown programming languages. You can usually guess what's going on even without a lot of knowledge. As an example, look at the following SQL code:

```
USE part7;

CREATE TABLE phoneList (
```

```
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
  lastName VARCHAR (15),
  email VARCHAR(20),
  phone VARCHAR(15) );
```

```
DESCRIBE phonelist;
```

This code is an SQL script. It is like a PHP program in the sense it is a set of instructions for the computer to follow. However, the PHP interpreter doesn't directly interact with the SQL language. Instead, these commands are sent to another program. As a PHP programmer, you'll also be able to write code that sends commands to a database language. Just as your PHP code often writes code in HTML format for the browser to interpret, you'll be writing SQL code for the MySQL interpreter to use.

When this code is sent to an SQL-compliant database program (such as MySQL) it will create the database structure shown in Table 7.1

2.1Using a Database

It is possible that you will have several database projects working in the same relational database system. In my case, each part of this book that uses SQL has its own database. Sometimes your system administrator will assign you a database. In any case, you will probably need to invoke that database with the USE command.

Creating a Table

To create a table, you must indicate the name of the table as well as each field in the table. For each field, you must list what type of data is held in the field, and (at least for text data) how many characters long the field will be. As an example, the following code creates the phoneList table:

```
CREATE TABLE phoneList (
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
```

lastName VARCHAR (15),

email VARCHAR(20),

phone VARCHAR(15));

You can think of fields as being much like variables, but while PHP is easy-going about what type of data is in a variable, SQL is very picky about the type of data in fields. In order to create an efficient database, MySQL needs to know exactly how many bytes of memory to set aside for every single field in the database. The primary way it does this is to require the database designer to specify the type and size of every field in each table. Table 7.2 lists a few of the primary data types supported by MySQL.

Table 7.2: COMMON DATA TYPES IN MYSQL

Data type	**Description**
INT	Standard integer 2 billion (roughly)
BIGINT	Big integer 9 x 10 ^18th
FLOAT	Floating point decimal number 38 digits
DOUBLE	Double precision floating point 308 digits
CHAR(n)	text with n digits. If actual value is less than n,
VARCH	Text with n digits. Trailing spaces will
DATE	Date in YYYY-MM-DD format
TIME	Time in HH:MM:SS format
YEAR	Year in YYYY format

Table 7.2

You might notice that it is not necessary to specify the length of numeric types (although you can determine a maximum size for numeric types as well as the number of digits you want stored in float and double fields). The storage requirements for numeric variables are based on the type itself.

Working with String Data in MySQL

Text values are usually stored in VARCHAR fields. These fields must include the number of characters allocated for the field. Both CHAR and VARCHAR fields have fixed lengths. The primary difference between them is what happens when the field contains a value shorter than the specified length. If you declare a CHAR field to have a length of 10 with

firstName VARCHAR(10);

and then later store the value 'Andy' into the field, the field will actually contain 'Andy ' (that is, Andy followed by six spaces). CHAR fields pad any remaining characters with spaces. The VARCHAR field type removes any padded spaces. Usually you will use the VARCHAR field type to store all your string data.

DETERMINING THE LENGTH OF A VARCHAR FIELD

Data design is both a science and an art. Determining the appropriate length for your text fields is one of the oldest problems in data.

If you don't allocate enough room for your text data, you can cause a lot of problems for your users. I once taught a course called CLT SD WEB PRG because the database that held the course names didn't have enough room for the actual name of the course (Client-Side Web Programming). My students renamed it the "Buy a Vowel" course.

However, you can't simply make every text field a thousand characters long, either, because this would be wasteful of system resources. If you have a field that will usually contain only five characters and you allocate one hundred characters, you will still require room on the drive for the extra 95 characters. If your database has thousands of entries, this can be a substantial cost in drive space. In a distributed environment, you'll also have to wait for those unnecessary spaces to come across limited bandwidth. It takes experimentation and practice to determine the appropriate width for your string fields. You'll

need to test your application with real users before you can really be sure if you've made the right decision.

Finishing up the CREATE TABLE Statement

Once you understand field data types, the CREATE TABLE syntax makes a lot of sense. There are only a few more details to understand. Once you specify CREATE TABLE, use a pair of parentheses to indicate the field list. Each field has a name followed by its type (and length, if it's a CHAR or VARCHAR). The fields are separated by commas. You do not have to put each field on its own line or indent the field definitions, but I prefer to do so, because these practices make the code much easier to read and debug.

Creating a Primary Key

You might be curious about the very first field in the phone list database. Just to refresh your memory, the line that defines that field looks like this:

```
id INT PRIMARY KEY,
```

Most database tables have some sort of field like this that holds a numeric value. This special field is called the primary key.

The code presented so far can be entered directly into the MySQL program.

You can see the code and its results in Figure 1.20

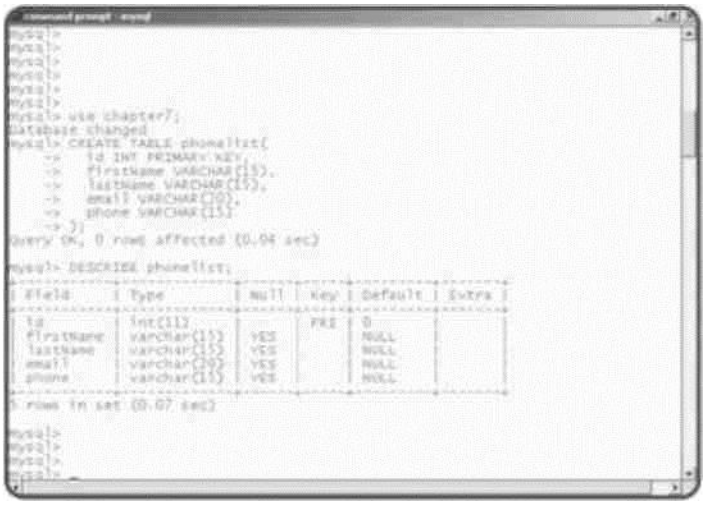


Figure 1.20: The MySQL command line tool after I created the phonelist table.

Using the DESCRIBE Command to Check the Structure of a Table

It can be useful to check the structure of a table, especially if somebody else created it or you don't remember exactly what types or sizes of fields are in the table. The DESCRIBE command lets you view the structure of a table.

Return a basic SQL query

Once you've created a table, you can begin to add data to it. The primary tool for adding records to a table is the INSERT command.

```
INSERT INTO phoneList VALUES (
   0, 'Andy', 'Harris', 'aharris@cs.iupui.edu', '123-4567' );
```

The INSERT statement allows you to add a record into a database. The values must be listed in exactly the same order the fields were defined. Each value is separated by a comma, and all VARCHAR and CHAR values must be enclosed in single quotes.

If you have a large amount of data to load into a database, you can also use the LOAD DATA command. This command accepts a tab-delimited text file with one row per record and fields separated by tabs. It then loads that entire file into the database. This is often the fastest way to load a database with test data. The following line loads data from a file called "addresses.txt" into the phoneList

table:

LOAD DATA LOCAL INFILE "addresses.txt" INTO TABLE phonelist; Figure 1.25 shows the MySQL tool after I have added one record to the table.

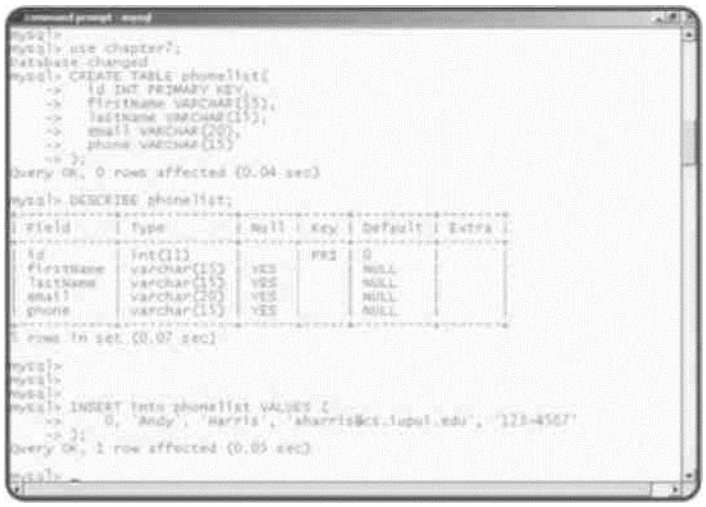


Figure 1.21: MySQL tells you the operation succeeded, but you don't get a lot more information.

Selecting Results

Of course, you'll want to see the results of all your table-building activities. If you want to see the data in a table, you can use the SELECT command. This is perhaps the most powerful command in SQL, but its basic use is quite simple. To see all of the data in the phonelist database, use this command:

SELECT * FROM phonelist

This command grabs all fields of all records of the phonelist database and displays them in a table format.

Figure 1.22shows what happens after I add a SELECT statement to get the results of the phone list.

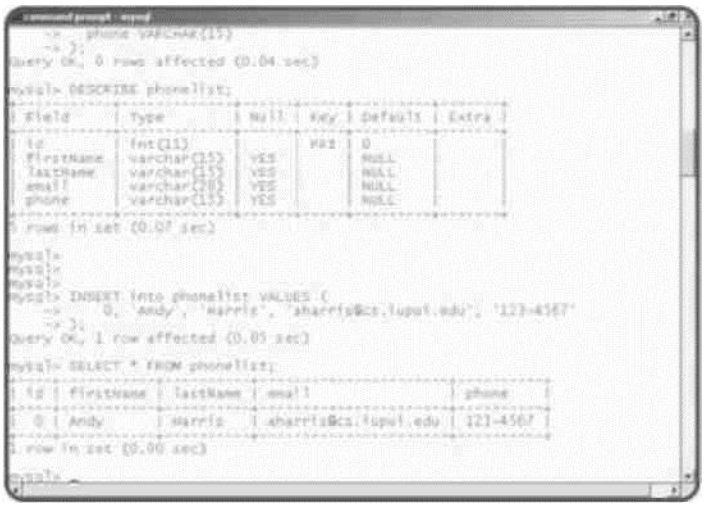


Figure 1.22 : The result of the SELECT statement is a table just like the original plan.

Writing a Script to Build a Table

It is very important to understand how to create tables by hand in SQL, because your programs will have to do this same work. However, it's very tedious to write your SQL code in the MySQL window directly. When you create real data applications, you'll often have to build and rebuild your data tables several times before you are satisfied with them, and this would be awkward directly in the command line interface. Also, as you are writing programs that work with your database, you will likely make mistakes that corrupt the original data. It's good to have a script ready that can easily rebuild the database with test data even if something goes wrong. Most programmers create a script of SQL commands with a text editor (you can use the same editor that you write your PHP code in) and use the SOURCE command to load that code in. Below is an SQL script for creating the phonelist database.

```
## build phone list ## for mySQL

USE part7;
DROP TABLE IF EXISTS phoneList;

CREATE TABLE phoneList (
```

```
  id INT PRIMARY KEY,
  firstName VARCHAR(15),
  lastName VARCHAR (15),
  email VARCHAR(20),
  phone VARCHAR(15) );
INSERT INTO phoneList VALUES (
  0, 'Andy', 'Harris', 'aharris@cs.iupui.edu', '123-4567' );
SELECT * FROM phoneList;
```

This code isn't exactly like the code I used in the interactive session, because there are a few more features that are especially handy when you create SQL code in a script.

SQL is actually a language. Although it isn't technically a programming language, it has many of the same features. Like PHP and other languages, SQL supports several types of comment characters. The # sign is often used to signify a comment in SQL. Comments are especially important when you save a group of SQL commands in a file for later reuse. These comments can help you remember what type of database you were trying to build. It's critical to put basic comments in your scripts so you will understand what they should do later.

Dropping a Table

It may seem strange to talk about deleting a table from a database before you've built one, but often (as in this case) a database is created using a script. Before you create a new table, you should check to see if it already exists, and if it does, delete it with the DROP command. The
DROP TABLE IF EXISTS phoneList;command does exactly that. If the phoneList table currently exists, it will be deleted, so there will be no confusion.

Running a Script with SOURCE

You can create an SQL script with any text editor. It is common to save SQL scripts with the .sql extension. Inside MySQL, you can use the SOURCE command to load and execute an external script. Figure 1.23 shows MySQL after I ran the buildPhonelist.sql script.

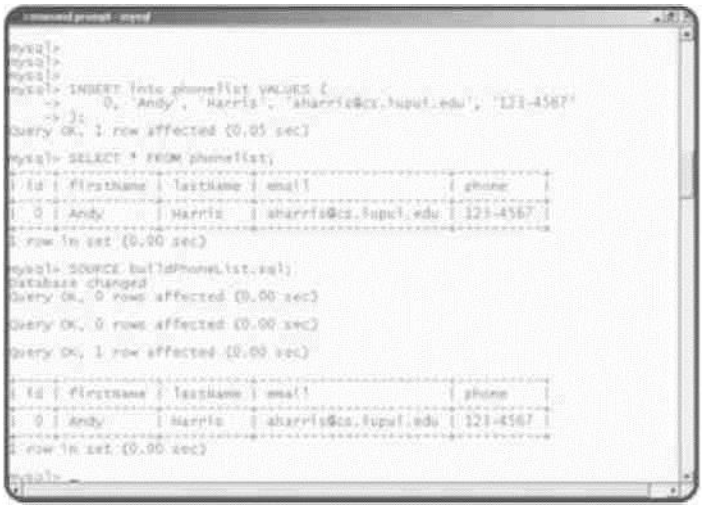


Figure 1.23: The SOURCE command allows you to read in SQL instructions from a file.

Connecting to the Hero Database

To show how this works, I'll build a simple PHP program that returns all the values in the hero database you created..

Figure 1.24 illustrates the Show Hero PHP program.

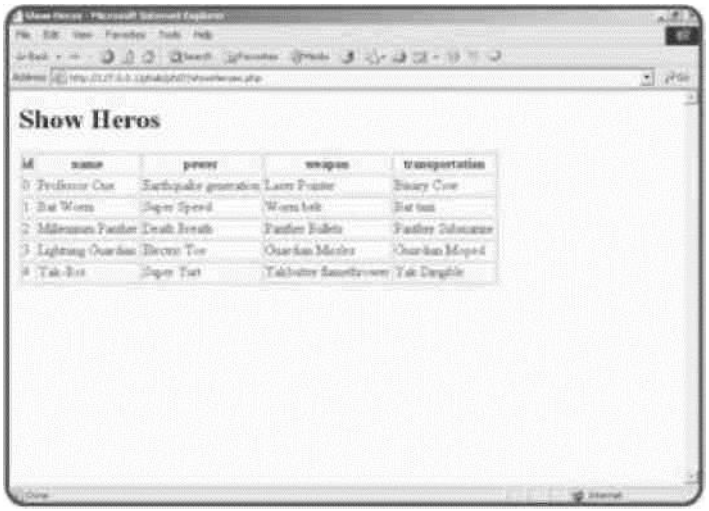


Figure 1.24 : This HTML table is generated by a PHP program reading the database.

The code that generates this page is shown below:

```
<body>
<h1>Show Heros</h1>
<?
//make the database connection
$conn = mysql_connect("localhost", "", "");
mysql_select_db("part7", $conn);

//create a query
$sql = "SELECT * FROM hero";
$result = mysql_query($sql, $conn);

print "<table border = 1>\n";

//get field names
print "<tr>\n";
```

```
while ($field = mysql_fetch_field($result)){
   print " <th>$field->name</th>\n"; } // end while print
"</tr>\n\n";

//get row data as an associative array while ($row =
mysql_fetch_assoc($result)){ print "<tr>\n";
  //look at each field foreach ($row as $col=>$val){
   print " <td>$val</td>\n"; } // end foreach print
"</tr>\n\n"; }// end while

print "</table>\n";
?>
</body>
</html>
```

Glance over the code, and you'll see it's mostly familiar except for a few new functions that begin with "mysql_" These functions are designed to allow access to MySQL databases. If you look through the PHP documentation, you'll see very similar functions for several other types of databases, including Oracle, Informix, mSQL, and ODBC. You'll find the process for connecting to and using other databases are pretty much the same no matter which database you're using.

2.3 Get a connection to a MySQL database

The first job is to get a connection between your PHP program and your MySQL server. You can connect to any server you have permission to use. The mysql_connect function arranges the communication link between MySQL and PHP. Here's the connect statement from the showHero program:

```
$conn = mysql_connect("localhost", "", "");
```

The mysql_connect() function requires three parameters: server name, username, and password. The server name is the name or URL of the MySQL server you wish to connect to (this will be localhost if your PHP and MySQL servers reside on the same machine, which is frequently the case). The username refers to the username in MySQL. Most database packages have user accounts.

You can use the same username and password you use to log into MySQL, and your program will have all the same access you do. Of course, you may want more restricted access for your programs, so you may want to create a special account, which has only the appropriate permissions, for users of your program. The mysql_connect() function returns an integer referring to the database connection. You can think of this identifier much like the file pointers you learned in "Working with Files." The data connection should be stored in a variable (I usually use something like $conn) because many of the other database functions will need to access the connection.

A data connection can have a number of databases connected to it. The mysql_set_db() function lets you choose a database. It works just like the USE command inside SQL. The mysql_set_db() function requires the name of a database and a data connection. This function returns the value FALSE if it was unable to connect to the specified database.

Creating a Query

Creating a query is very easy. The relevant code from showHero.php is reproduced here:

```
//create a query
$sql = "SELECT * FROM hero";
$result = mysql_query($sql, $conn);
```

You begin by placing SQL code inside a variable.

The mysql_query() function allows you to pass an SQL command through a connection to a database. You can send any SQL command to the database with mysql_query(), including table creation statements, updates, and queries. It returns a special element called a result set. If the SQL command was a query, the result variable will hold a pointer to the data, which we'll take apart in the next step. If it's a data definition command (the commands used to create, and modify tables) the result object will usually contain the string related to the success or failure of the operation.

I'll be printing the data out in an HTML table. I could create the table headings by hand, because I know what all the fields are, but it's better to get the field information directly from the query, because you won't always know which fields are being returned by a particular query. The next chunk of code manages this task.

```
print "<table border = 1>\n";

//get field names
print "<tr>\n";
while ($field = mysql_fetch_field($result)){
   print " <th>$field->name</th>\n"; } // end while print "</tr>\n\n";
```

The mysql_fetch_field() function expects a query result as its one parameter. It then fetches the next field and stores it in the $field variable. If there are no fields left in the result, the function returns the value FALSE. This allows the field function to also be used as a conditional statement.

The $field variable is actually an object. You haven't used PHP objects yet, but they're really not too difficult. The $field object in this case is much like an associative array. It has a number of properties (which can be thought of as the

attributes of the field). The field object has a number of attributes, listed in Table 8.1 .

Property	**Attribute**
max_length	How long the field is (Especially important in VARCHAR fields)
name	The name of the field
primary_ke	TRUE if the field is a primary key
table	Name of table this field belongs to
type	

By far the most common use of the field object is to determine the names of all the fields in a query. The other attributes can be useful in certain situations. You can see the complete list of attributes in the MySQL online help.

You use a slightly new syntax to refer to the properties of an object. Notice that I printed $field->name to the HTML table. This syntax simply refers to the name property of the field object. For now if you want to think of it as a fancy associative array, that would be reasonably accurate.

Creation and implementing part

```
<?php
include_once("conf.php");
function kun($kun, $dir, $yil)
{ //echo $kun."-".$dir." --".$yil;
$sql="SELECT DISTINCT(kunlik) FROM kunlik";
$r=mysql_query($sql);
if (mysql_num_rows($r)>0){return false;}
else
{return true;
}}
$day1=date("j")-1;
$month=date("m");
$ad1=$day1.".".$month;
$findday = "SELECT DISTINCT(kunlik) FROM kunlik";
$findday2=mysql_query($findday);
while($i=mysql_fetch_array($findday2)){
if ($ad1==$i['kunlik']){
$day=date("j")-1;
$month=date("m");
$ad=$day.".".$month;
$year=date("Y");
        break;}
else {
$ad=20.07;
$year=date("Y")-1;
}}
?>
<html><title>TATU Farg'ona filialiga topshirganlar royhati</title>
<body>
```

```
<link rel="stylesheet" href="bootstrap.css" />
<table align=center class="table table-hover table-striped" style = "width:80%; padding-top:150px; font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;">
<thead><tr align=center  ><td  colspan=16><?echo $ad."."$year; ?></td>
</tr>
<tr align=center >
        <th rowspan="3">Tr</th>
        <th rowspan="3">Ta'lim kodi</th>
        <th rowspan="3">Ta'lim yo'nalishi</th>
        <th rowspan="3">Qabul <br/>rejasi<br/> <?echo $year; ?><br/> yil</th>
        <th rowspan="3">O'tgan yilgi <br/>jami</th>
        <th colspan="3" rowspan="1"><? echo $year; ?> yilga jami</th>
        <th colspan="3" rowspan="1"><? echo $ad."."$year; ?> yil kuniga</th>
        <th colspan="5" rowspan="1" >xarbiy imtiyozlar</th>
</tr>
<tr align=center >
        <th rowspan=2>o'zbek</th>
        <th rowspan=2>rus</th>
        <th rowspan=2>jami</th>
        <th rowspan=2>o'zbek</th>
        <th rowspan=2>rus</th>
        <th rowspan=2>jami</th>
        <th rowspan=1 colspan=2>xarbiy </th>
        <th rowspan=1 colspan=3>jami <?echo $year; ?> yil</th>
</tr>
<tr align = center >
        <th>o'zbek</th>
        <th>rus</th>
        <th>o'zbek</th>
        <th>rus</th>
```

```
        <th>jami</th>
</tr>
</thead>
<tbody>
<?php
$query="select * from direction, kunlik, jadval where kunlik.kunlik=".$ad." anddirection.direct_id=kunlik.kunlik_directand kunlik.kunlik_direct=jadval.direction_id and kunlik.kunlik_yil=".$year." and jadval.yil=".$year." group by direction.direct_id";
$result=mysql_query($query) or die(mysql_error());
$sum ="select kunlik_direct,sum(kunlik_ozbek) as jamiuzbek,sum(kunlik_rus) as jamirus, sum(kunlik_xarbiy_o) as xarbiy_ozbek, sum(kunlik_xarbiy_r) as xarbiy_rus from kunlik where kunlik_yil='".$year."' group by kunlik_direct";
$sum = mysql_query($sum) or die (mysql_error());
$i=0;
while($row2=mysql_fetch_array($sum)){
$jamiuzbek[$i]=$row2['jamiuzbek'];
$jamirus[$i]=$row2['jamirus'];
$xarbiy_ozbek[$i]=$row2['xarbiy_ozbek'];
$xarbiy_rus[$i]=$row2['xarbiy_rus'];
$i++; }
if (mysql_num_rows($result)>0)
{ $i=0;         while ($row=mysql_fetch_object($result))  {
?>
<tr align = center class="color2" >
<td><?php echo $row->direct_id;?></td>
<td><?php echo $row->direct_number;?></td>
<td align=left><?php echo $row->direct_name;?></td>
<td><?php echo $qabul[$i]=$row->qabul_rejasi;?></td>
<td><?php echo $otgan_yilga_jami[$i]=$row->otgan_yilgi_jami;?></td>
```

```
<td><?php echo $jamiuzbek[$i];?></td>
<td><?php echo $jamirus[$i];?></td>
<td><?php echo $jami[$i]=$jamiuzbek[$i]+$jamirus[$i];?></td>
<td><?php echo $kunlik_ozbek[$i]=$row->kunlik_ozbek;?></td>
<td><?php echo $kunlik_rus[$i]=$row->kunlik_rus;?></td>
<td><?php echo $kunlik_jami[$i]=$row->kunlik_ozbek+$row->kunlik_rus;?></td>
<td><?php echo $kunlik_xarbiy_o[$i]=$row->kunlik_xarbiy_o;?></td>
<td><?php echo $kunlik_xarbiy_r[$i]=$row->kunlik_xarbiy_r;?></td>
<td><?php echo $xarbiy_ozbek[$i];?></td>
<td><?php echo $xarbiy_rus[$i];?></td>
<td><?php echo $jami2[$i]=$xarbiy_ozbek[$i]+$xarbiy_rus[$i];?></td>
</tr>
<?php
$i++;
}
$row=mysql_fetch_object($result);
echo "<tr align=center class='color2'>

                <td colspan=3><b> Jami</b></td>
                <td>".array_sum($qabul)."</td>
                <td>".array_sum($otgan_yilga_jami)."</td>
                <td>".array_sum($jamiuzbek)."</td>
                <td>".array_sum($jamirus)."</td>
                <td>".array_sum($jami)."</td>
                <td>".array_sum($kunlik_ozbek)."</td>
                <td>".array_sum($kunlik_rus)."</td>
                <td>".array_sum($kunlik_jami)."</td>
                <td>".array_sum($kunlik_xarbiy_o)."</td>
                <td>".array_sum($kunlik_xarbiy_r)."</td>
```

```
                <td>".array_sum($xarbiy_ozbek)."</td>
                <td>".array_sum($xarbiy_rus)."</td>
                <td>".array_sum($jami2)."</td>
                 </tr>
        </tbody>
                 </table>
</body>
</html>";
 }?>
```

20.07.2014

Tr	Ta'lim kodi	Ta'lim yo'nalishi	Qabul rejasi 2014 yil	O'tgan yilgi jami	2014 yilga jami			20.07.2014 yil kuniga			xarbiy imtiyozlar				
											xarbiy		jami 2014 yil		
					o'zbek	rus	jami	o'zbek	rus	jami	o'zbek	rus	o'zbek	rus	jami
1	5330501	Kompyuter injiniringi	194	1561	12	12	24	12	12	24	21	21	21	21	42
2	5350101	Telekommunikatsiya tehnologiyalari	54	557	12	12	24	12	12	24	1	21	1	21	22
3	5350401	AKT sohasida kasb talim	24	494	2	12	14	2	12	14	21	12	21	12	33
Jami			272	2612	26	36	62	26	36	62	43	54	43	54	97

Include(“conf.php”) ;

This php function upload this file to web site and use only this page. In the this file consists of connection to database name, username, password, email and other common variables.

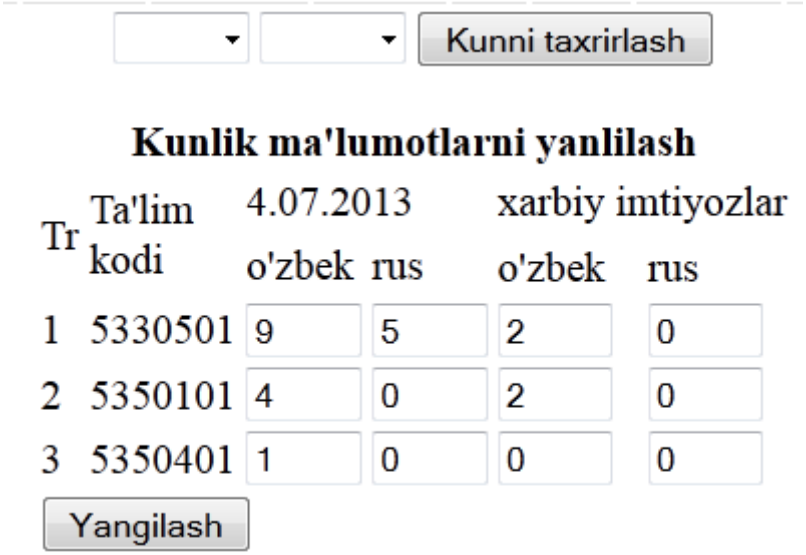
These codes get year, month and day from the server and it show

This files can user all web user and it changes daily when admin update tables. This table divided into three part they consist of : three faculty results. This table consists of two part, they are: how many applicants applied last year and until last day results, how many applicants had gone army. It is useful for all applicant they can check every day.

```
<?
session_start();
if ($_SESSION["adminlog"] == ""){
header("Location:index.php");
} else { include_once("admin.php");
?>
<link rel=stylesheet href="style.css" />
<form action="<? print $_SERVER['PHP_SELF']; ?>" method="POST" class
= "center">
<select name=year>
<option value=""></option>
<? include("conf.php");
$yil="SELECT DISTINCT(kunlik_yil) FROM kunlik";
$query=mysql_query($yil);
while($row=mysql_fetch_array($query)){
echo "<option value=".$row['kunlik_yil'].">".$row['kunlik_yil']."</option>" ;
} ?>
</select>
<select name=day>
    <option value=""></option>
    <?      $er = 1;
    $sql = "SELECT * from kunlik where kunlik_direct=".$er;
    $result = mysql_query($sql) or die (mysql_error());
    while($row=mysql_fetch_object($result)){    echo "<option value=".$row-
>kunlik.">".$row->kunlik."</option>";        }
    ?>
</select>
<input type="submit" value="Kunni taxrirlash" name="f1">
</form>
<?php
```

```
    if(isset($_POST['f1'])){
    include("conf.php");
    $year=$_POST['year'];
    $day=$_POST['day'];  ?>
   <form action="update.php" method="POST">
    <table  align=center cellspadding=5>
    <tr align=center><td colspan=6><strong>Kunlik ma'lumotlarni
yanlilash</strong></td></tr>
    <tr>
       <td rowspan='3'>Tr</td>
       <td rowspan='3'>Ta'lim<br/> kodi</td>
       <td colspan='2' rowspan='1'><? echo $day.".".$year; ?></td>
       <td colspan='2' rowspan='1'>xarbiy imtiyozlar</td>
    </tr>
    <tr>
       <td rowspan=2>o'zbek</td>
       <td rowspan=2>rus</td>
    </tr>
    <tr>
       <td>o'zbek</td>
       <td>rus</td>
    </tr>
    <tr>
    <? $query="select * from direction, kunlik where
kunlik.kunlik_direct=direct_id and kunlik=".$day." and kunlik_yil=".$year;
    $result=mysql_query($query) or die("Sorov bajarilmadi");
    echo "<input type='hidden' name='day' value=".$day.">";
    echo "<input type='hidden' name='year' value=".$year.">";
    if ($result>0){
    while($row=mysql_fetch_object($result)){
```

```
    ?>
    <td><input type="hidden" name="direct_id[]" value="<? echo $row-
>direct_id; ?>"> <?php  echo $row->direct_id; ?></td>
    <? echo "<td>".$row->direct_number."</td>";
    ?>
    <td><input type='text' name=ozbek[] size=4 value="<?php echo $row-
>kunlik_ozbek; ?>" required></td>
    <td><input type=text  name=rus[] size=4 value="<?php echo $row-
>kunlik_rus; ?>" required></td>
    <td><input type=text  name=xarbiy_o[] value="<?php echo $row-
>kunlik_xarbiy_o; ?>" size=4 required></td>
    <td><input type=text  name=xarbiy_r[] value="<?php echo $row-
>kunlik_xarbiy_r; ?>" size=4 required></td>
    </tr>
    <?
    } echo "<tr><td colspan=6><input type='submit' value='Yangilash'
name='update'></td>
    </tr>
    </table>
    </form>";
   }}}
    include_once("royhat.php");
    ?>
```



This picture is update daily information. Admin easy to use this table and control comfortable. Admin panel files in safety folder.

Defence of labour

Requirements to condition of the work on computer

Microclimate premisesses. The Parameters microclimate and their feature.

A Condition of the air ambience of the premises. The Qualitative composition of the air. The Contents of dust and other polluting material in worker to zone. An Electromagnetic fields and x-ray radiation. The Natural ventilation; The Artificial ventilation; The Air conditioning.

Illumination

The Natural illumination. The Influence to solar radiation on premiseses.

The Color decorating of the interior of the premises as method of protection solar radiation:

-An Artificial illumination. The System of the artificial illumination. The Choice lamp;

-A Standertization of the artificial illumination.

Production noise

-A Standartization of the production noise in computer premises;

-Barriering design and internal sewer of the premises.

Electrical safety

-A Danger of the defeat by electric current;

-An Earth of the computer equipment, requirements;

-A Facility collective and the individual safety when work with equipment;

-A Rendering first help before vrachebnoy at electrical trauma.

Fireman safety

A Reasons fire.

- A Primary facilities fire;

-A Way to evacuations in the event of arising the fire;

-A Fireman water-supply.

On each point, previously than proceed with development action, is conducted analysis existing bad and dangerous production factor, under

development actions must be concrete and provide the decision sanitary-hygenic, technical and organizing problems, which prevent the negative factors and create the comfortable of the condition of the work.

The Section is terminated by formation to instructions safety when functioning(working) on computer.

Calculation Intensity Noise

The Noise and vibration present itself professional bad if their intensity exceeds the certain level. For fight with noise are used general and the individual means of protection. Correct planning has Big importance and accommodation enterprise and their separate shop to other noisy enterprise. The Noise is in production premiseses possible vastly to reduce facing a sewer material. When designing and installation different mechanism equipments necessary to provide possibility of the reduction to vibrations and noise to account of the installing the equipment on special shock absorber, reduction revolving details, change the striking interaction unaccented In given problem necessary:

1. Define the total intensity of the noise from three sources on given worker a place;
2. Define the intensity of the noise if wall and ceiling cover material;
3. Formulate the findings.

The Methods of the calculation:

1. The Calculation of the change level to intensities of the noise with change the distance R from the source of the noise is produced on formula:

$$L_R \approx L_1 - 20 \lg R - 8, \partial Б, (4.1)$$

where LR and L1 - a level to intensities of the noise of the source on distance R metre and one metre accordingly.

If between source of the noise and worker revenge there is wall-barrier, level to intensities of the noise falls on N db

$$N = 14,5 \lg G = 15, \partial Б, (4.2)$$

where G - a mass one m2 wall-barriers

Level to intensities of the noise on worker place with account of the influence wall-barriers is defined as

$$L'_R = L_R - N, \partial Б, (4.3)$$

Total intensity of the noise two sources with level LA and LV, is defined as

$$L_\Sigma = L_A + \Delta L, \partial Б, (4.4)$$

where LA - most from two summiruemых level, db;

ΔL - an adjustment, hanging from difference level, is defined on table 4.1.

In table 4.2. consider the level to intensities of the noise, with provision for influences wall-barriers.

At determination of the total power several sources summation follows to conduct consecutively, as from the most intensive.

Follows to take into account that L_Σ it is defined for three sources of the noise and each source is considered with corresponding to wall-barrier.

The Parameters (the type of the material, thickness and mass 1 m2) wall-barriers to take from table 4.3.

Table 4.3

№	Material and designs	Thickness to designs, m	Mass 1/m2 barriers, kgs
1	Wall brick	0,12	250
2	Wall brick	0,25	470
3	Wall brick	0,38	690
4	Wall brick	0,52	934
5	Paperboard in several layers	0,02	12
6	Paperboard in several layers	0,04	24
7	Voylok	0,025	8
8	Voylok	0,05	16

9	Reinforced concrete	0,1	240
10	Reinforced concrete	0,2	480
11	Wall from shlakobetona	0,14	150
12	Stena iz shlakobetona	0,28	300
13	Partiton from boards by thickness 0,02 m, with two sides	0,06	70
14	Partiton from boards by thickness 0,01 m, with two sides	0.18	95
15	Gypsum partiton	0,11	117

At determination of the intensities of the noise after covering sewer and ceiling material for simplicity is allowed neglect the action direct sound fine, consider that wall- barriers are found inwardly premiseses and on influences do not render.

Total sewer and ceiling is defined as.

$$M = S_{nm} \cdot \alpha = S_c \cdot \beta = S_{nm.} \cdot \gamma, ед.погл., (4.5)$$

where Snm, Ss - accordingly area of the ceiling and sewer of the premises.

Table 4.4

	Penultimate numeral of the number of the student ticket									
	1	2	3	4	5	6	7	8	9	0
S_{nm}, m^2	100	150	200	250	300	350	400	450	500	550
S_c ,m^2	160	180	200	220	250	260	280	300	320	340
α_1,10^{-3}	20	25	30	35	40	45	40	35	30	25
α_2 10^{-2}	95	90	85	80	75	70	75	80	85	90
β_1,10^{-3}	34	33	32	31	30	31	32	33	34	35
β_2,10^{-2}	75	80	85	90	95	90	85	80	75	70

α, β, γ - accordingly factors of the absorption material, which cover ceiling, wall and floor.

Is it Here taken into account that area flap and ceiling of the premises are. Reduction to intensities of the noise will form

$$K = 10 \lg \frac{M_2}{M_1}, \partial Б \ , (4.6)$$

where M1, M2,- accordingly without covering sewer and ceiling special material (M1) and after covering such material (M2), ed. has bent.

Importance M1, is calculated with use factor a1 and M2 - with use a2, and the Floor usually material is not covered and at calculation to take that floor parquet.

Level to intensities of the noise on worker place with account of the covering sewer and ceiling material will form

$$L'_{\Sigma} = L_{\Sigma} - K \ , \partial Б \ , (4.7)$$

Conclusion

My full name is Turg'unaliyev Bekzod. I am a fours year student of Tashkent University of Information Technologies. I made graduation qualification work on this theme "Creating reception of interactive service for hight educational establishments" and its useful to applicants and high establishments. This interactive service increases web users.

Interactive web services are increasingly replacing traditional static web pages. Producing web services seems to require a tremendous amount of laborious low-level coding due to the primitive nature of PHP programming. We present ideas for an improved runtime system for interactive web services built on top of Common Gateway Interface running on virtually every combination of browser and HTTP/CGI server. The runtime system has been implemented and used extensively a tool for producing interactive web services.

An interactive web service consists of a global shared state (typically a database) and a number of distinct sessions that each contain some local private state and a sequential, imperative action. A web client may invoke an individual thread of one of the given session kinds. The execution of this thread may interact with the client and inspect or modify the global state.

It consists of three main parts, they are: analytical part, designing part and creation and implementing part. Also one additional part that is Introduction part. The analytical part includes main information of the diploma work. While the creating diploma work I learned completely working in PHP5.

BIBLIOGRAPHY LIST

20.07.2014

Tr	Ta'lim kodi	Ta'lim yo'nalishi	Qabul rejasi 2014 yil	O'tgan yilgi jami	2014 yilga jami			20.07.2014 yil kuniga			xarbiy imtiyozlar				
											xarbiy		jami 2014 yil		
					o'zbek	rus	jami	o'zbek	rus	jami	o'zbek	rus	o'zbek	rus	jami
1	5330501	Kompyuter injiniringi	194	1561	12	12	24	12	12	24	21	21	21	21	42
2	5350101	Telekommunikatsiya tehnologiyalari	54	557	12	12	24	12	12	24	1	21	1	21	22
3	5350401	AKT sohasida kasb talim	24	494	2	12	14	2	12	14	21	12	21	12	33
Jami			272	2612	26	36	62	26	36	62	43	54	43	54	97

admin

●●●

Kirish

Kunlik ma'lumotlar qoshish | Kunlik ma'lumotlarni taxrirlash | Chiqish

20.07.2013

Tr	Ta'lim kodi	Ta'lim yo'nalishi	Qabul rejasi 2013 yil	O'tgan yilgi jami	2013 yilga jami			20.07.2013 yil kuniga			xarbiy imtiyozlar				
											xarbiy		jami 2013 yil		
					o'zbek	rus	jami	o'zbek	rus	jami	o'zbek	rus	o'zbek	rus	jami
1	5330501	Kompyuter injiniringi	190	1398	1073	305	1378	92	37	129	1	3	34	23	57
2	5350101	Telekommunikatsiya tehnologiyalari	50	381	301	104	405	17	16	33	0	3	25	8	33
3	5350401	AKT sohasida kasb talim	25	370	343	91	434	26	15	41	1	0	5	2	7
Jami			265	2149	1717	500	2217	135	68	203	2	6	64	33	97

Kunlik ma'lumotlarni qo'shish

18.06 yil kuniga xarbiy imtiyozlar

Tr	Ta'lim kodi	o'zbek	rus	o'zbek	rus
1	5330501				
2	5350101				
3	5350401				

Qoshish

Kunlik ma'lumotlar qoshish | Kunlik ma'lumotlarni taxrirlash | Chiqish

20.07.2013

Tr	Ta'lim kodi	Ta'lim yo'nalishi	Qabul rejasi 2013 yil	O'tgan yilgi jami	2013 yilga jami			20.07.2013 yil kuniga			xarbiy imtiyozlar				
					o'zbek	rus	jami	o'zbek	rus	jami	xarbiy		jami 2013 yil		
											o'zbek	rus	o'zbek	rus	jami
1	5330501	Kompyuter injiniringi	190	1398	1073	305	1378	92	37	129	1	3	34	23	57
2	5350101	Telekommunikatsiya tehnologiyalari	50	381	301	104	405	17	16	33	0	3	25	8	33
3	5350401	AKT sohasida kasb talim	25	370	343	91	434	26	15	41	1	0	5	2	7
		Jami	265	2149	1717	500	2217	135	68	203	2	6	64	33	97

Kunni taxrirlash

Kunni taxrirlash

Kunlik ma'lumotlarni yanlilash

4.07.2013 xarbiy imtiyozlar

Tr	Ta'lim kodi	o'zbek	rus	o'zbek	rus
1	5330501	9	5	2	0
2	5350101	4	0	2	0
3	5350401	1	0	0	0

Yangilash

LIST OF REFERENCE

1. “Uzbekistan on the Threshold of the 21st Century: Threats to the Security, Conditions and Guarantees of Progress”, I.A.Karimov. Tashkent: "Uzbekistan", 1997.

2. “There Is No Future Without Historical Memory”, I.A.Karimov. Tashkent: "Uzbekistan", 1999.

3. “WICKED COOL PHP REAL-WORLD SCRIPTS THAT SOLVE DIFFICULT PROBLEMS” by william Steinmetz with Brain Ward 2009

4. “LEARNING PHP MYSQL AND JAVASCRIPT”
Robin Nixon 2011

5. “Php Building Responsive Web Applications “
Cristian Darie, Bogdan Brinzarea, Filip Cherecbes-Tose, Mibai Bucica 2006

6. “HTML, Javascript PHP and MySQL” with cd Nikolay Proxorenok 2010

7. “Learning PHP beginners ” Laura Tomson 2006

Internet resource

1. http://www.myrusakov.ru
2. http:/forum.php.ru
3. http:/.phpbb.ru
4. http://php-forum.com
5. http://fluxbb.org
6. http://www.sitepoint.com/forums/php
7. http://webdeveloper.com/forum
8. http://forums.php.freaks.com/
9. http://phorum.org
10. http://google.com